



**HAL**  
open science

# Efficient Enumeration of Fixed Points in Complex Boolean Networks Using Answer Set Programming

Van-Giang Trinh, Belaid Benhamou, Sylvain Soliman

► **To cite this version:**

Van-Giang Trinh, Belaid Benhamou, Sylvain Soliman. Efficient Enumeration of Fixed Points in Complex Boolean Networks Using Answer Set Programming. 29th International Conference on Principles and Practice of Constraint Programming (CP 2023), Aug 2023, Toronto, Canada. pp.35:1–35:19, 10.4230/LIPIcs.CP.2023.35 . hal-04209296

**HAL Id: hal-04209296**

**<https://amu.hal.science/hal-04209296v1>**

Submitted on 22 Sep 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# 29th International Conference on Principles and Practice of Constraint Programming


CP 2023, August 27–31, 2023, Toronto, Canada

Edited by

Roland H. C. Yap



*Editors*

**Roland H. C. Yap** 

National University of Singapore, School of Computing, 13 Computing Drive, Singapore  
ryap@comp.nus.edu.sg

*ACM Classification 2012*

Theory of computation → Constraint and logic programming

**ISBN 978-3-95977-300-3**

*Published online and open access by*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern,  
Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-300-3>.

*Publication date*

September, 2023

*Bibliographic information published by the Deutsche Nationalbibliothek*

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed  
bibliographic data are available in the Internet at <https://portal.dnb.de>.

*License*

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0):

<https://creativecommons.org/licenses/by/4.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work  
under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.CP.2023.0

ISBN 978-3-95977-300-3

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

## LIPICs – Leibniz International Proceedings in Informatics

LIPICs is a series of high-quality conference proceedings across all fields in informatics. LIPICs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

### *Editorial Board*

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University, Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Pierre Senellart (ENS, Université PSL, Paris, FR)

**ISSN 1868-8969**

**<https://www.dagstuhl.de/lipics>**



## ■ Contents

Preface	
<i>Roland H.C. Yap</i> .....	0:ix–0:x
Senior Program Committee	
.....	0:xi
Program Committee	
.....	0:xii
Additional Reviewers	
.....	0:xiii
Authors	
.....	0:xv–0:xx

### Invited Talks

Beyond Optimal Solutions for Real-World Problems	
<i>Maria García de la Banda</i> .....	1:1–1:4
A Tale of Two Cities: Teaching CP with Story-Telling	
<i>Jimmy H.M. Lee</i> .....	2:1–2:1
The CP-SAT-LP Solver	
<i>Laurent Perron, Frédéric Didier, and Steven Gay</i> .....	3:1–3:2
Coupling CP with Deep Learning for Molecular Design and SARS-CoV2 Variants Exploration	
<i>Thomas Schiex</i> .....	4:1–4:3
CP Solver Design for Maximum CPU Utilization	
<i>Petr Vilím</i> .....	5:1–5:1

### Regular Papers

Optimization of Short-Term Underground Mine Planning Using Constraint Programming	
<i>Younes Aalian, Gilles Pesant, and Michel Gamache</i> .....	6:1–6:16
Exploiting Configurations of MaxSAT Solvers	
<i>Josep Alòs, Carlos Ansótegui, Josep M. Salvía, and Eduard Torres</i> .....	7:1–7:16
Symmetries for Cube-And-Conquer in Finite Model Finding	
<i>João Araújo, Chaiwah Chow, and Mikoláš Janota</i> .....	8:1–8:19
Guiding Backtrack Search by Tracking Variables During Constraint Propagation	
<i>Gilles Audemard, Christophe Lecoutre, and Charles Prud'homme</i> .....	9:1–9:17
Incremental Constrained Clustering by Minimal Weighted Modification	
<i>Aymeric Beauchamp, Thi-Bich-Hanh Dao, Samir Loudni, and Christel Vrain</i> ....	10:1–10:22

Simplifying Step-Wise Explanation Sequences <i>Ignace Bleukx, Jo Devriendt, Emilio Gamba, Bart Bogaerts, and Tias Guns</i> .....	11:1–11:20
Towards More Efficient Local Search for Pseudo-Boolean Optimization <i>Yi Chu, Shaowei Cai, Chuan Luo, Zhendong Lei, and Cong Peng</i> .....	12:1–12:18
Boosting Decision Diagram-Based Branch-And-Bound by Pre-Solving with Aggregate Dynamic Programming <i>Vianney Coppé, Xavier Gillard, and Pierre Schaus</i> .....	13:1–13:17
Fast Matrix Multiplication Without Tears: A Constraint Programming Approach <i>Arnaud Deza, Chang Liu, Pashootan Vaezipoor, and Elias B. Khalil</i> .....	14:1–14:15
Probabilistic Inference by Projected Weighted Model Counting on Horn Clauses <i>Alexandre Dubray, Pierre Schaus, and Siegfried Nijssen</i> .....	15:1–15:17
A CP Approach for the Liner Shipping Network Design Problem <i>Yusra El Ghazi, Djamel Habet, and Cyril Terrioux</i> .....	16:1–16:21
Optimization Models for Pickup-And-Delivery Problems with Reconfigurable Capacities <i>Arnoosh Golestanian, Giovanni Lo Bianco, Chengyu Tao, and J. Christopher Beck</i>	17:1–17:17
Preprocessing in SAT-Based Multi-Objective Combinatorial Optimization <i>Christoph Jabs, Jeremias Berg, Hannes Ihalainen, and Matti Järvisalo</i> .....	18:1–18:20
An Efficient Constraint Programming Approach to Preemptive Job Shop Scheduling <i>Carla Juvin, Emmanuel Hebrard, Laurent Houssin, and Pierre Lopez</i> .....	19:1–19:16
Horizontally Elastic Edge Finder Rule for Cumulative Constraint Based on Slack and Density <i>Roger Kameugne, Sévérine Fetgo Betmbe, Thierry Noulamo, and Clémentin Tayou Djamegni</i> .....	20:1–20:17
Exploring Hydrogen Supply/Demand Networks: Modeller and Domain Expert Views <i>Matthias Klapperstueck, Frits de Nijs, Ilankaikone Senthoooran, Jack Lee-Kopij, María García de la Banda, and Michael Wybrow</i> .....	21:1–21:18
Binary Constraint Trees and Structured Decomposability <i>Petr Kučera</i> .....	22:1–22:19
Large Neighborhood Beam Search for Domain-Independent Dynamic Programming <i>Ryo Kuroiwa and J. Christopher Beck</i> .....	23:1–23:22
MDD Archive for Boosting the Pareto Constraint <i>Steve Malalel, Arnaud Malapert, Marie Pelleau, and Jean-Charles Régim</i> .....	24:1–24:15
Learning a Generic Value-Selection Heuristic Inside a Constraint Programming Solver <i>Tom Marty, Tristan François, Pierre Tessier, Louis Gautier, Louis-Martin Rousseau, and Quentin Cappart</i> .....	25:1–25:19

Proof Logging for Smart Extensional Constraints <i>Matthew J. McIlree and Ciaran McCreesh</i> .....	26:1–26:17
Improving Conflict Analysis in MIP Solvers by Pseudo-Boolean Reasoning <i>Gioni Mezi, Timo Berthold, Ambros Gleixner, and Jakob Nordström</i> .....	27:1–27:19
Using Canonical Codes to Efficiently Solve the Benzenoid Generation Problem with Constraint Programming <i>Xiao Peng and Christine Solnon</i> .....	28:1–28:17
Distribution Optimization in Constraint Programming <i>Guillaume Perez, Gaël Glorian, Wijnand Suijlen, and Arnaud Lallouet</i> .....	29:1–29:19
The p-Dispersion Problem with Distance Constraints <i>Nikolaos Ploskas, Kostas Stergiou, and Dimosthenis C. Tsouros</i> .....	30:1–30:18
Partially Preemptive Multi Skill/Mode Resource-Constrained Project Scheduling with Generalized Precedence Relations and Calendars <i>Guillaume Pováda, Nahum Alvarez, and Christian Artigues</i> .....	31:1–31:21
Assembly Line Preliminary Design Optimization for an Aircraft <i>Stéphanie Roussel, Thomas Polacsek, and Anouck Chan</i> .....	32:1–32:19
SAT-Based Learning of Compact Binary Decision Diagrams for Classification <i>Pouya Shati, Eldan Cohen, and Sheila McIlraith</i> .....	33:1–33:19
Constraint Programming with External Worst-Case Traversal Time Analysis <i>Pierre Talbot, Tingting Hu, and Nicolas Navet</i> .....	34:1–34:20
Efficient Enumeration of Fixed Points in Complex Boolean Networks Using Answer Set Programming <i>Van-Giang Trinh, Belaid Benhamou, and Sylvain Soliman</i> .....	35:1–35:19
Guided Bottom-Up Interactive Constraint Acquisition <i>Dimosthenis C. Tsouros, Senne Berden, and Tias Guns</i> .....	36:1–36:20
Addressing Problem Drift in UNHCR Fund Allocation <i>Sameela Suharshani Wijesundara, Maria Garcia de la Banda, and Guido Tack</i> ...	37:1–37:18
From Formal Boosted Tree Explanations to Interpretable Rule Sets <i>Jinqiang Yu, Alexey Ignatiev, and Peter J. Stuckey</i> .....	38:1–38:21
Searching for Smallest Universal Graphs and Tournaments with SAT <i>Tianwei Zhang and Stefan Szeider</i> .....	39:1–39:20
FastMapSVM for Predicting CSP Satisfiability <i>Kezin Zheng, Ang Li, Han Zhang, and T. K. Satish Kumar</i> .....	40:1–40:17
Improving Local Search for Pseudo Boolean Optimization by Fragile Scoring Function and Deep Optimization <i>Wenbo Zhou, Yujiao Zhao, Yiyuan Wang, Shaowei Cai, Shimao Wang, Xinyu Wang, and Minghao Yin</i> .....	41:1–41:18



## Short Papers

Predict-Then-Optimise Strategies for Water Flow Control <i>Vincent Barbosa Vaz, James Bailey, Christopher Leckie, and Peter J. Stuckey</i> . . . .	42:1–42:10
Constraint Programming Models for Depth-Optimal Qubit Assignment and SWAP-Based Routing <i>Kyle E. C. Booth</i> . . . . .	43:1–43:10
Constraint Model for the Satellite Image Mosaic Selection Problem <i>Manuel Combarro Simón, Pierre Talbot, Grégoire Danoy, Jędrzej Musiał, Mohammed Alswaitti, and Pascal Bouvry</i> . . . . .	44:1–44:15
Partitioning a Map into Homogeneous Contiguous Regions: A Branch-And-Bound Approach Using Decision Diagrams <i>Nicolas Golenvoux, Xavier Gillard, Siegfried Nijssen, and Pierre Schaus</i> . . . . .	45:1–45:10
Constraint Programming to Improve Hub Utilization in Autonomous Transfer Hub Networks <i>Chungjae Lee, Wirattawut Boonbandansook, Vahid Eghbal Akhlaghi, Kevin Dalmeijer, and Pascal Van Hentenryck</i> . . . . .	46:1–46:11
A New Approach to Finding $2 \times n$ Partially Spatially Balanced Latin Rectangles <i>Renee Mirka, Laura Greenstreet, Marc Grimson, and Carla P. Gomes</i> . . . . .	47:1–47:11
Proven Optimally-Balanced Latin Rectangles with SAT <i>Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider</i> . . . . .	48:1–48:10
Enumerative Level-2 Solution Counting for Quantified Boolean Formulas <i>Andreas Plank, Sibylle Möhle, and Martina Seidl</i> . . . . .	49:1–49:10

## ■ Preface

This volume contains the proceedings of the *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, which was held in Toronto, Canada, August 27–31, 2023. More details of the conference can be found at <https://cp2023.a4cp.org/index.html>.

Held annually, CP is the premier international conference on constraint programming. CP is concerned with all aspects of computing with constraints, including, but not restricted to: theory, algorithms, environments, languages, models, systems, and applications.

As is customary for CP, papers could be submitted to multiple tracks. CP 2023 had the following tracks:

- |                               |                       |
|-------------------------------|-----------------------|
| ■ Technical Track             | Chair: Roland Yap     |
| ■ Applications Track          | Chair: Helmut Simonis |
| ■ Machine Learning Track      | Chair: Tias Guns      |
| ■ Operations Research Track   | Chair: Gilles Pesant  |
| ■ Trustworthy Decision Making | Chair: Peter Stuckey  |

In addition, there was a SAT Fast Track to synchronize with the SAT 2023 conference.

A total of 109 papers (excluding abstracts) were submitted to these tracks. Authors could submit either a full paper with a maximum length of 15 pages (references excluded) or a short paper with a maximum length of 8 pages (references excluded). Each paper was reviewed with a senior Program Committee together with Program Committees and additional reviewers recruited by the Program Committee. The track chairs managed the review process for their respective tracks. All papers had at least three reviews. Authors had the opportunity to answer questions from reviewers in an author response phase. Based on extensive discussion on the papers from reviewers, program and senior program committee, and track chairs taking into account reviews and author responses, a total of 44 papers were accepted. A meta-review was prepared for each paper by a senior program committee member summarizing the decision with suggestions to authors. The Senior Program Committee and Chairs nominated papers for the best paper prizes. A select committee from the Senior Program Committee together with the Program Chair awarded the Best Paper Prize to Mathew J. McIlree and Ciaran McCreesh for “Proof Logging for Smart Extensional Constraints” and the best application paper prize to Matthias Klapperstueck, Frits De Nijs, Ilankaikone Senthooran, Jack Lee-Kopij, Maria Garcia De La Banda and Michael Wybrow for “Exploring Hydrogen Supply/Demand Networks: Modeller and Domain Expert views”.

In addition to the paper tracks, the conference had a number of satellite events. Lars Kotthoff (University of Wyoming) organized the workshops on the first day of the conference with five workshops. The doctoral program, also on the first day, was organized by Xavier Gillard (Université catholique de Louvain) and forms an important part of the conference to give students an environment to present their research with discussions with senior researchers, and networking activities combined with a poster presentation during the conference reception. CP features two tutorials organized by Emir Demirović (TU Delft) on the timely topics of explainable constraint solving and machine learning for solvers. The conference program featured four invited talks given by Maria Garcia de la Banda, Jimmy Lee, Laurent Perron, Thomas Schiex and Petr Vilím. The talks were selected to showcase a range of research in CP and include constraint solver design, designing proteins with applications to the SARS-Cov2 virus, teaching of CP, and making optimisation technology more usable.



Many people have contributed to make the conference a success. The conference would not be possible for the hard work of the authors in submitting high-quality scientific work which forms the basis of the conference proceedings. The program and senior committee together with track chairs had the challenging tasks of selecting papers as well as providing authors with suggestions on paper improvements. A special thanks goes to Andre Cire and Eldan Cohen, the conference and local chairs respectively, both at the University of Toronto, who made CP possible in Toronto. In addition to the above mentioned conference organizers, I would like to thank the following co-organizers. The diversity, equity, and inclusion (DEI) chairs, Maria Andreina Francisco Rodriguez (Uppsala University) and Andrea Rendl (Satalia). Arvind Raghunathan (Mitsubishi Electric Research Laboratories), the sponsorship chair. Anna Latour (National University of Singapore) was instrumental in managing the website and also handled the publicity and social media.

I would also like to thank the Association for Constraint Programming (ACP) which makes the CP conference series possible. The conference is grateful to the ACP president, David Bergman (University of Connecticut) and the ACP conference coordinator, H el ene Verhaeghe (KU Leuven) for their support and feedback.

The conference acknowledges the generous support of all our sponsors:

- ACP
- The Artificial Intelligence Journal (Elsevier)
- Cosling
- Department of Management, University of Toronto Scarborough
- Google
- IBM
- MERL
- The Optimization Firm
- ScheduleOpt

July 2023, Singapore

Roland Yap

## ■ Organization

### Senior Program Committee

Christian Bessiere	CNRS, University of Montpellier, France
Mats Carlsson	RISE Research Institutes of Sweden, Sweden
Berthe Choueiry	University of Nebraska-Lincoln, USA
David Cohen	Royal Holloway, University of London
Maria Garcia De La Banda	Monash University, Australia
Ian Gent	University of St Andrews, United Kingdom
Philip Kilby	Data61 & The Australian National University
Christophe Lecoutre	CRIL, University of Artois, France
Jimmy Lee	The Chinese University of Hong Kong, Hong Kong
Kuldeep Meel	National University of Singapore, Singapore
Ian Miguel	University of St Andrews, United Kingdom
Michela Milano	University of Bologna, Italy
Barry O'Sullivan	University College Cork, Ireland
Justin Pearson	Uppsala University, Sweden
Thomas Schiex	Universite Fédérale de Toulouse, ANITI, INRAE, France
Laurent Simon Labri	Bordeaux Institute of Technology, France
Christine Solnon	INSA Lyon, France
Peter J. Stuckey	Monash University, Australia
Michael Trick	Carnegie Mellon University, USA
Neil Yorke-Smith	Delft University of Technology, Netherlands



### Program Committee

Carlos Ansotegui	Universitat de Lleida, Spain
Christopher Beck	University of Toronto, Canada
Nicolas Beldiceanu	IMT Atlantique (LS2N), France
David Bergman	University of Connecticut, USA
Armin Biere	University of Freiburg, Germany
Nikolaj Bjorner	Microsoft, USA
Ken Brown	University College Cork
Quentin Cappart	Ecole Polytechnique de Montréal, Canada
Martin Cooper	IRIT - Université Paul Sabatier, France
Simon de Givry	INRA - MIAT, France
Jean-Guillaume Fages	COSLING, France
Pierre Flener	Uppsala University, Sweden
Emmanuel Hebrard	LAAS, CNRS, France
Alexey Ignatiev	Monash University, Australia
George Katsirelos	MIA Paris, INRAE, AgroParisTech, France
Zeynep Kiziltan	University of Bologna, Italy
T. K. Satish Kumar	University of Southern California, USA
Mikael Lagerkvist	Optischedule, Sweden
Nadjib Lazaar	UM2-LIRMM, France
Chu-Min Li	Université de Picardie Jules Verne, France
Michele Lombardi	University of Bologna, Italy
Ciaran McCreesh	University of Glasgow, United Kingdom
Laurent Michel	University of Connecticut, USA
Nysret Musliu	TU Wien, Austria
Peter Nightingale	University of York, United Kingdom
Jakob Nordstrom	University of Copenhagen & Lund University
Marie Pelleau	Université Côte d'Azur, France
Guillaume Perez	University of Nice-Sophia Antipolis/I3S, France
Laurent Perron	Google France, France
Steve Prestwich	Insight Centre for Data Analytics, Ireland
Patrick Prosser	Glasgow University, United Kingdom
Charles Prud'Homme	IMT Atlantique, LS2N, France
Claude-Guy Quimper	Laval University, Canada
Jean-Charles Regin	University Nice-Sophia Antipolis/I3S/CNRS, France
Pierre Schaus	UCLouvain, Belgium
Paul Shaw	IBM, France
Mohamed Siala	INSA Toulouse & LAAS-CNRS, France
Mate Soos	National University of Singapore, Singapore
Kostas Stergiou	University of Western Macedonia, Greece
Guido Tack	Monash University, Australia
Cyril Terrioux	LIS - UMR CNRS 7020 - Aix-Marseille Université, France
Gilles Trombettoni	LIRMM, University of Montpellier, France
Charlotte Truchet	Université de Nantes, France
Willem-Jan Van Hoeve	Carnegie Mellon University, USA
Hélène Verhaeghe	KU Leuven, Belgium
Petr Vilím	ScheduleOpt
Mark Wallace	Monash University, Australia
Ruiwei Wang	National University of Singapore, Singapore
Armin Wolf	Fraunhofer Institute, Germany
Lebbah Yahia	University of Oran 1, Algeria
Neng-Fa Zhou	CUNY Brooklyn College & Graduate Center, USA

**Additional Reviewers**

Josep Alòs  
Valentin Antuori  
Noureddine Aribi  
Federico Baldo  
Nassim Belmecheri  
Hendrik Bierlee  
Andrea Borghesi  
Touati Chahira  
Vianney Coppé  
Timothy Curry  
Toby O. Davies  
Guillaume Derval  
Julien Ferry  
Romain Fontaine  
Andrea Formisano  
Matteo Francobaldi  
Xavier Gillard  
Ramiz Gindullin  
Luca Giuliani  
Arthur Gontier  
Cristian Grozea  
Matthias Horn  
Victor Jung  
Anthony Karahalios  
Amina Kemmar  
Lucas Kletzander  
Stepan Kochemazov  
Tanguy Lapègue  
Shuolin Li  
Matthew J. McIlree  
Claude Michel  
Florian Mischek  
Eleonora Misino  
Pierre Montalbano  
Antonio Morgado  
Bertrand Neveu  
Andy Oertel  
Jolan Philippe  
Philippe Refalo  
András Z. Salamon  
Noah Schutte  
Jean-Baptiste Sciau  
Dimosthenis C. Tsouros  
Marc Vinyals  
Damien T. Wojtowicz  
Oleg Zaikin  
Yuanlin Zhang



## ■ List of Authors

Younes Aalian (6)

Département de mathématiques et de génie industriel, Polytechnique Montréal, Québec, Canada

Vahid Eghbal Akhlaghi (46)

H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

Mohammed Alswaitti (44)

University of Luxembourg, Luxembourg; Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg

Nahum Alvarez (31)

Airbus (AI Research), Toulouse, France

Josep Alòs (7)

Logic & Optimization Group (LOG), University of Lleida, Spain

Carlos Ansótegui (7)

Logic & Optimization Group (LOG), University of Lleida, Spain

João Araújo (8)

Universidade Nova de Lisboa, Lisbon, Portugal

Christian Artigues (31)

LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Gilles Audemard (9)

CRIL, Univ. Artois & CNRS, France

James Bailey (42)

The University of Melbourne, Australia

Vincent Barbosa Vaz (42)

The University of Melbourne, Australia; the Australian Research Council OPTIMA ITTC, Melbourne, Australia

Aymeric Beauchamp (10)

University of Orléans, INSA Centre Val de Loire, LIFO EA 4022, France

J. Christopher Beck (17, 23)

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Belaïd Benhamou (35)

LIS, Aix-Marseille University, Marseille, France

Senne Berden (36)

KU Leuven, Belgium

Jeremias Berg (18)

HIIT, Department of Computer Science, University of Helsinki, Finland

Timo Berthold (27)

Fair Isaac Deutschland GmbH, Berlin, Germany; TU Berlin, Germany

Sévérine Fetgo Betmbe (20)

Faculty of Sciences, Department of Mathematics and Computer Science, University of Dschang, Cameroon

Giovanni Lo Bianco (17)

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Ignace Bleukx (11)

DTAI, KU Leuven, Belgium

Bart Bogaerts (11)

Artificial Intelligence Lab, VUB, Brussels, Belgium

Wirattawut Boonbandansook (46)

H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

Kyle E. C. Booth (43)

Amazon Quantum Solutions Lab, Seattle, WA, USA

Pascal Bouvry (44)

University of Luxembourg, Luxembourg; Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg

Shaowei Cai (12, 41)

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Quentin Cappart (25)

Polytechnique Montréal, Montreal, Canada

Anouck Chan (32)

ONERA, ONERA DTIS, Toulouse, Université de Toulouse, France

Choiwah Chow (8)

Universidade Aberta, Lisbon, Portugal

Yi Chu (12)

Institute of Software, Chinese Academy of Sciences, Beijing, China



- Eldan Cohen (33)  
Department of Mechanical and Industrial Engineering, University of Toronto, Canada
- Manuel Combarro Simón  (44)  
University of Luxembourg, Luxembourg;  
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg
- Vianney Coppé  (13)  
UCLouvain, Louvain-la-Neuve, Belgium
- Kevin Dalmeijer  (46)  
H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA
- Grégoire Danoy  (44)  
University of Luxembourg, Luxembourg;  
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg
- Thi-Bich-Hanh Dao  (10)  
University of Orléans, INSA Centre Val de Loire, LIFO EA 4022, France
- Frits de Nijs  (21)  
Department of Data Science and AI, Faculty of IT, Monash University, Clayton, VIC, Australia; ARC Industrial Training and Transformation Centre OPTIMA, Carlton, VIC, Australia
- Jo Devriendt  (11)  
DTAI, KU Leuven, Belgium
- Arnaud Deza (14)  
Department of Mechanical and Industrial Engineering, University of Toronto, Canada
- Frédéric Didier (3)  
Google, Paris, France
- Clémentin Tayou Djamegni  (20)  
Faculty of Sciences, Department of Mathematics and Computer Science, University of Dschang, Cameroon; UIT Fotso Victor of Bandjoun, Department of Computer Engineering, University of Dschang, Cameroon
- Alexandre Dubray  (15)  
Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), UCLouvain, Belgium
- Yousra El Ghazi  (16)  
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
- Tristan François (25)  
Ecole Polytechnique, Palaiseau, France
- Michel Gamache (6)  
Département de mathématiques et de génie industriel, Polytechnique Montréal, Québec, Canada
- Emilio Gamba  (11)  
Data Analytics Lab, VUB, Brussels, Belgium; DTAI, KU Leuven, Belgium
- Maria Garcia de la Banda  (1, 21, 37)  
Department of Data Science and AI, Faculty of IT, Monash University, Clayton, Victoria, Australia; ARC Industrial Training and Transformation Centre OPTIMA, Carlton, Victoria, Australia
- Louis Gautier (25)  
Ecole Polytechnique, Palaiseau, France
- Steven Gay (3)  
Google, Paris, France
- Xavier Gillard  (13, 45)  
UCLouvain, Louvain-la-Neuve, Belgium
- Ambros Gleixner  (27)  
HTW Berlin, Germany; Zuse Institute Berlin, Germany
- Gaël Glorian  (29)  
Huawei Technologies Ltd, CSI Paris, Boulogne-Billancourt, France
- Nicolas Golenvaux  (45)  
Institute for Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), Université catholique de Louvain, Louvain-la-Neuve, Belgium
- Arnoosh Golestanian  (17)  
Department of Mechanical and Industrial Engineering, University of Toronto, Canada
- Carla P. Gomes (47)  
Cornell University, Ithaca, NY, USA
- Laura Greenstreet (47)  
Cornell University, Ithaca, NY, USA
- Marc Grimson (47)  
Cornell University, Ithaca, NY, USA
- Tias Guns  (11, 36)  
DTAI, KU Leuven, Belgium; Data Analytics Lab, VUB, Brussels, Belgium
- Djamal Habet  (16)  
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
- Emmanuel Hebrard  (19)  
LAAS-CNRS, Université de Toulouse, France

- Laurent Houssin  (19)  
ISAE-SUPAERO, Université de Toulouse,  
France
- Tingting Hu (34)  
University of Luxembourg, Luxembourg
- Alexey Ignatiev  (38)  
Department of Data Science and AI, Monash  
University, Clayton, Australia
- Hannes Ihalainen  (18)  
HIIT, Department of Computer Science,  
University of Helsinki, Finland
- Peter J. Stuckey  (42)  
Monash University, Australia; the Australian  
Research Council OPTIMA ITTC, Melbourne,  
Australia
- Christoph Jabs  (18)  
HIIT, Department of Computer Science,  
University of Helsinki, Finland
- Mikoláš Janota  (8)  
Czech Technical University in Prague, Czech  
Republic
- Carla Juvin (19)  
LAAS-CNRS, Université de Toulouse, France
- Matti Järvisalo  (18)  
HIIT, Department of Computer Science,  
University of Helsinki, Finland
- Roger Kameugne  (20)  
Faculty of Sciences, Department of Mathematics  
and Computer Science, University of Maroua,  
Cameroon
- Elias B. Khalil (14)  
Department of Mechanical and Industrial  
Engineering, University of Toronto, Canada
- Matthias Klapperstueck  (21)  
Department of Human-Centred Computing,  
Faculty of IT, Monash University, Clayton, VIC,  
Australia
- T. K. Satish Kumar (40)  
University of Southern California, Los Angeles,  
CA, USA
- Ryo Kuroiwa  (23)  
Department of Mechanical and Industrial  
Engineering, University of Toronto, Canada
- Petr Kučera  (22)  
Department of Theoretical Computer Science  
and Mathematical Logic, Faculty of  
Mathematics and Physics, Charles University,  
Prague, Czech Republic
- Arnaud Lallouet  (29)  
Huawei Technologies Ltd, CSI Paris,  
Boulogne-Billancourt, France
- Christopher Leckie  (42)  
The University of Melbourne, Australia
- Christophe Lecoutre (9)  
CRIL, Univ. Artois & CNRS, France
- Chungjae Lee  (46)  
H. Milton Stewart School of Industrial and  
Systems Engineering, Georgia Institute of  
Technology, Atlanta, GA, USA
- Jimmy H.M. Lee (2)  
Department of Computer Science and  
Engineering, The Chinese University of Hong  
Kong, China
- Jack Lee-Kopij (21)  
Woodside Energy Ltd., Perth, Australia
- Zhendong Lei  (12)  
State Key Laboratory of Computer Science,  
Institute of Software, Chinese Academy of  
Sciences, Beijing, China
- Ang Li (40)  
University of Southern California, Los Angeles,  
CA, USA
- Chang Liu (14)  
Department of Mechanical and Industrial  
Engineering, University of Toronto, Canada
- Pierre Lopez  (19)  
LAAS-CNRS, Université de Toulouse, France
- Samir Loudni  (10)  
TASC (LS2N-CNRS), IMT Atlantique, France,  
GREYC, University of Caen Normandy, France
- Chuan Luo  (12)  
School of Software, Beihang University, Beijing,  
China
- Steve Malalel (24)  
Université Côte d'Azur, CNRS, I3S, Nice,  
France
- Arnaud Malapert (24)  
Université Côte d'Azur, CNRS, I3S, Nice,  
France

- Tom Marty  (25)  
Polytechnique Montréal, Montreal, Canada;  
Ecole Polytechnique, Palaiseau, France
- Ciaran McCreesh  (26)  
University of Glasgow, UK
- Sheila McIlraith (33)  
Department of Computer Science, University of  
Toronto, Canada; Vector Institute, Toronto,  
Canada
- Matthew J. McIlree  (26)  
University of Glasgow, UK
- Gioni Mexi  (27)  
Zuse Institute Berlin, Germany
- Renee Mirka (47)  
Cornell University, Ithaca, NY, USA
- Jedrzej Musial  (44)  
Poznan University of Technology, Poland
- Sibylle Möhle  (49)  
Max Planck Institute for Informatics,  
Saarbrücken, Germany
- Nicolas Navet  (34)  
University of Luxembourg, Luxembourg
- Siegfried Nijssen  (15, 45)  
Institute of Information and Communication  
Technologies, Electronics and Applied  
Mathematics (ICTEAM), UCLouvain, Belgium
- Jakob Nordström  (27)  
University of Copenhagen, Denmark; Lund  
University, Sweden
- Thierry Noulamo  (20)  
UIT Fotso Victor of Bandjoun, Department of  
Computer Engineering, University of Dschang,  
Cameroon
- Marie Pelleau (24)  
Université Côte d'Azur, CNRS, I3S, Nice,  
France
- Cong Peng  (12)  
Finovation in CCBFT, Beijing, China
- Xiao Peng (28)  
Univ Lyon, INSA Lyon, Inria, CITI, EA3720,  
69621 Villeurbanne, France
- Guillaume Perez  (29)  
Huawei Technologies Ltd, CSI Paris,  
Boulogne-Billancourt, France
- Laurent Perron (3)  
Google, Paris, France
- Vaidyanathan Peruvemba Ramaswamy  (48)  
Algorithms and Complexity Group, TU Wien,  
Austria
- Gilles Pesant (6)  
Département de génie informatique et génie  
logiciel, Polytechnique Montréal, Québec,  
Canada
- Andreas Plank  (49)  
Institute for Symbolic Artificial Intelligence,  
JKU Linz, Austria
- Nikolaos Ploskas  (30)  
University of Western Macedonia, Kozani,  
Greece
- Thomas Polacsek  (32)  
ONERA, ONERA DTIS, Toulouse, Université  
de Toulouse, France
- Guillaume Pováda  (31)  
Airbus (AI Research), Toulouse, France
- Charles Prud'homme  (9)  
TASC, IMT-Atlantique, LS2N-CNRS, France
- Louis-Martin Rousseau  (25)  
Polytechnique Montréal, Montreal, Canada
- Stéphanie Roussel  (32)  
ONERA, ONERA DTIS, Toulouse, Université  
de Toulouse, France
- Jean-Charles Régim (24)  
Université Côte d'Azur, CNRS, I3S, Nice,  
France
- Josep M. Salvía  (7)  
Logic & Optimization Group (LOG), University  
of Lleida, Spain
- Pierre Schaus  (13, 15, 45)  
UCLouvain, Louvain-la-Neuve, Belgium
- Thomas Schiex  (4)  
Université Fédérale de Toulouse, ANITI,  
INRAE, UR 875, 31326 Toulouse, France
- Martina Seidl  (49)  
Institute for Symbolic Artificial Intelligence,  
JKU Linz, Austria
- Hankaikone Senthoooran  (21)  
Department of Data Science and AI, Faculty of  
IT, Monash University, Clayton, VIC, Australia;  
ARC Industrial Training and Transformation  
Centre OPTIMA, Carlton, VIC, Australia

- Pouya Shati (33)  
Department of Computer Science, University of Toronto, Canada; Vector Institute, Toronto, Canada
- Sylvain Soliman  (35)  
Lifeware team, Inria Saclay, Palaiseau, France
- Christine Solnon (28)  
Univ Lyon, INSA Lyon, Inria, CITI, EA3720, 69621 Villeurbanne, France
- Kostas Stergiou  (30)  
University of Western Macedonia, Kozani, Greece
- Peter J. Stuckey  (38)  
Department of Data Science and AI, Monash University, Clayton, Australia; Australian Research Council OPTIMA ITTC, Clayton, Australia
- Wijnand Suijlen  (29)  
Huawei Technologies Ltd, CSI Paris, Boulogne-Billancourt, France
- Stefan Szeider  (39, 48)  
Algorithms and Complexity Group, TU Wien, Austria
- Guido Tack  (37)  
Department of Data Science and AI, Faculty of IT, Monash University, Clayton, Australia; ARC Industrial Training and Transformation Centre OPTIMA, Clayton, Australia
- Pierre Talbot  (34, 44)  
University of Luxembourg, Luxembourg; Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg
- Chengyu Tao  (17)  
Department of Mechanical and Industrial Engineering, University of Toronto, Canada
- Cyril Terrioux  (16)  
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
- Pierre Tessier (25)  
Ecole Polytechnique, Palaiseau, France
- Eduard Torres  (7)  
Logic & Optimization Group (LOG), University of Lleida, Spain
- Van-Giang Trinh  (35)  
LIS, Aix-Marseille University, Marseille, France
- Dimosthenis C. Tsouros  (30, 36)  
KU Leuven, Belgium
- Pashootan Vaezipoor (14)  
Department of Computer Science, University of Toronto, Canada
- Pascal Van Hentenryck  (46)  
H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA
- Petr Vilím  (5)  
ScheduleOpt, Nový Knín, Czech Republic
- Christel Vrain  (10)  
University of Orléans, INSA Centre Val de Loire, LIFO EA 4022, France
- Shimao Wang (41)  
School of Information Science and Technology, Northeast Normal University, Changchun, China
- Xinyu Wang (41)  
School of Information Science and Technology, Northeast Normal University, Changchun, China
- Yiyuan Wang  (41)  
School of Information Science and Technology, Northeast Normal University, Changchun, China; Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China
- Sameela Suharshani Wijesundara  (37)  
Department of Data Science and AI, Faculty of IT, Monash University, Clayton, Australia; ARC Industrial Training and Transformation Centre OPTIMA, Clayton, Australia
- Michael Wybrow  (21)  
Department of Human-Centred Computing, Faculty of IT, Monash University, Clayton, VIC, Australia
- Minghao Yin  (41)  
School of Information Science and Technology, Northeast Normal University, Changchun, China; Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China
- Jinqiang Yu  (38)  
Department of Data Science and AI, Monash University, Clayton, Australia; Australian Research Council OPTIMA ITTC, Clayton, Australia
- Han Zhang (40)  
University of Southern California, Los Angeles, CA, USA

Tianwei Zhang  (39)


Algorithms and Complexity Group, TU Wien,  
Austria

Yujiao Zhao  (41)

School of Information Science and Technology,  
Northeast Normal University, Changchun, China

Kexin Zheng (40)

University of Southern California, Los Angeles,  
CA, USA

Wenbo Zhou  (41)

School of Information Science and Technology,  
Northeast Normal University, Changchun,  
China; Key Laboratory of Applied Statistics of  
MOE, Northeast Normal University, Changchun,  
China; Key Laboratory of Symbolic  
Computation and Knowledge Engineering of  
MOE, Jilin University, Changchun, China

# Beyond Optimal Solutions for Real-World Problems

Maria Garcia de la Banda  

Department of Data Science and AI, Faculty of IT, Monash University, Clayton, Victoria, Australia  
ARC Industrial Training and Transformation Centre OPTIMA, Carlton, Victoria, Australia

---

## Abstract

Combinatorial optimisation technology has come a long way. We now have mature high-level modelling languages in which to specify a model of the particular problem of interest [18, 7, 24, 6]; robust complete solvers in each major constraint paradigm, including Constraint Programming (CP) [1, 19], MaxSAT [5, 11], and Mixed Integer Programming (MIP) [2, 3]; effective incomplete search techniques that can easily be combined with complete solvers to speed up the search such as Large Neighbourhood Search [23]; and enough general knowledge about modelling techniques to understand the need for our models to incorporate components such as global constraints [25], symmetry constraints [8], and more. All this has significantly reduced the amount of knowledge required to apply this technology successfully to the many different combinatorial optimisation problems that permeate our society.

And yet, not many organisations use such advanced optimisation technology; instead, they often rely on the solutions provided by problem-specific algorithms that are implemented in traditional imperative languages and lack any of the above advances. Further, while advanced optimisation technology is particularly suitable for the kind of complex human-in-the-loop decision-making problems that occur in critical sectors of our society, including health, transport, energy, disaster management, environment and finance, these decisions are often still made by people with little or no technological support. In this extended abstract I argue that to change this state of affairs, our research focus needs to change from improving the technology on its own, to improving it so that users can better trust, use, and maintain the optimisation systems that we develop with it. The rest of this extended abstract discusses my personal experiences and opinion on these three points.

## Trust

I highlight trust (which focuses on the user's point of view) rather than trustworthiness (which is a characteristic of the software itself) because I think it is the former rather than the latter that is at stake for the adoption of optimisation technology.

One of the biggest hurdles I have found for trust in the context of optimisation systems is for the domain experts to (feel like they) *understand the underlying model*. While many users will never do (or have to), I believe it is key for domain experts to have a high-level understanding of the constraints in the model, since their (dis)trust will likely spread through the organisation, impacting the adoption of the system. Thanks to the use of high-level modelling languages in CP, our group has achieved this [13] by documenting the constraints in a language the user knows (mathematics) and linking each constraint to the particular part of the model that implements it (via comments). While domain experts do not completely understand the model, the similarity between the format they understand (mathematics) and the model constraint has helped them verify our perception of their problem and improved their trust in the model. However, more needs to be done in this direction via the development of formal techniques. For example, our group is exploring the use of *domain-specific languages* [10] as a bridge between domain experts and modellers that helps both trust and maintenance (see later). This [27] and other approaches need to be explored.

A very significant source of trust for our domain experts (and of trustworthiness for the software) has been the development of two different models implemented by two different people for the same problem [13]. While this can be seen as a prohibitively expensive exercise, it did not take that long once the first model was mature, is a good way to onboard new optimisation team members, and has helped up detect not only bugs but also differences in the interpretation of domain expert information. For optimisation problems where it is not possible to verify the optimality (or even correctness) of the solution, we see such *redundant modelling* as the only solution for now. Interestingly, a significant



© Maria Garcia de la Banda;

licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 1; pp. 1:1–1:4

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1:2 Beyond Optimal Solutions

step forward in obtaining the trust of our domain experts has been the *generation of an optimality gap* whenever an optimal solution could not be found due to time constraints. While explaining this concept took time, once understood it has boosted their trust, particularly when tackling problems where the solution is not easy verifiable or when approximated models/data are used (needed for speed, see later). This makes it difficult to work with CP and SAT solvers, as they usually lack tight lower bounds. Finally, trust is often developed through the use of the system, which I discuss below.

### Use

Usability is known to be key for the deployment of software systems. By “system” in our context, I refer to the combination of the problem model(s), the associated solver(s) and, importantly, the User Interface (UI) that often integrates them and is fundamental to their success. In addition to the traditional usability characteristics of software systems, I believe an optimisation system requires particular care in the following areas. *Interaction*, i.e., the system must allow users to interact with the UI not only to provide and modify the input data, but also to modify the constraints (at the very least by turning some on/off) as well as explore and compare solutions, as argued in [17, 15]. Incremental compilers and solvers would significantly help in making this easier, as well as generic ways for the UIs to communicate with them. *Conflict resolution*, that is, ensuring the system can not only detect infeasible instances, but also support users in understanding the data/constraints that cause infeasibility and how to modify the instance to make it feasible. Any interactive optimisation system that has users, will likely have conflicts. Thus, it is mandatory for CP to improve its conflict resolution technology which, while existent [16, 14, 22], is not widespread and it is often still problem-dependent, overwhelming (in the number of constraints shown to the user) and slow. Without it, users will be “stumped” when (rather than if) infeasibility is reached. *Solution diversity*, that is, supporting users in obtaining a diverse set of (close-to-optimal) solutions, where diversity is measured by a user-provided metric modelled somehow. While some solver-independent technology has been developed and implemented for this [9, 20, 12], it should be easier to use and more widespread. Further, it requires sophisticated solution comparison capabilities and, importantly, for optimal solutions to be found in seconds rather than hours. This brings me to *speed*, an area where CP solvers are falling behind. Most of our research group applications now use MIP solvers due to the need for floats (which precludes us from using learning solvers such as Chuffed [4]), but also to the lack of effective warm-start processes that are available in MIP solvers. Interestingly, data and model approximations have been proved to achieve orders of magnitude speedups with small reductions in optimality [13]. Developing generic (i.e., problem independent) accurate approximations would be extremely useful for complex decision systems. Other areas where I think generic CP methods are worth investigating more include dealing with *uncertainty* and *online* problems, ensuring solution *fairness* (even if it is over time), and studying *predict + optimise* approaches.

### Maintain

I know very few papers devoted to the issue of maintenance in optimisation technology. While this may be due to my lack of knowledge, I suspect it is also due to the limited adoption of optimisation technology. While the issues in this area are again common to other software systems, I believe the solutions for CP require special attention. For example, the issue of changes in user requirements (that our research group calls *problem drift*) seems particularly prevalent in decision-making systems, as such problems can evolve rapidly due to unforeseen circumstances. This can make optimisation systems obsolete faster than expected. Our research group has proposed to tackle problem drift by developing a *requirements model* implemented in the above-mentioned MDSLs and created by both domain experts and modellers that, when modified re-generates parts of the model to support the modifications [27]. This and other approaches such as the creation of reusable models components [21, 26], or instantiatable classes for common problem domains, are worth investigating.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Theory of computation → Integer programming; Human-centered computing → Information visualization

**Keywords and phrases** Combinatorial optimisation systems, usability, trust, maintenance

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.1

**Category** Invited Talk

**Acknowledgements** Any good idea here is the result of working in many different projects with my colleagues at the Optimisation and at the Data Visualisation and Immersive Analytics research groups in the Faculty of IT, Monash University. I have learned a lot from them.

---

## References

- 1 GECODE - An open, free, efficient constraint solving toolkit. URL: <https://www.gecode.org/>.
- 2 Gurobi Optimizer Reference Manual. URL: <https://www.gurobi.com/documentation/9.5/refman/index.html>.
- 3 IBM ILOG CPLEX optimizer. URL: <https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer>.
- 4 Geoffrey Chu. Improving combinatorial optimization - extended abstract. In Francesca Rossi, editor, *23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*, pages 3116–3120. IJCAI/AAAI, 2013. URL: <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6687>.
- 5 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *17th International Conference on Principles and Practice of Constraint Programming - CP 2011*, Lecture Notes in Computer Science, pages 225–239. Springer, 2011. doi:10.1007/978-3-642-23786-7\_19.
- 6 Robert Fourer, David M Gay, and Brian W Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.
- 7 Alan M. Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel. Essence: A constraint language for specifying combinatorial problems. *Constraints*, 13(3):268–306, September 2008. doi:10.1007/s10601-008-9047-y.
- 8 Ian P. Gent, Karen E. Petrie, and Jean-François Puget. Symmetry in constraint programming. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 329–376. Elsevier, 2006. doi:10.1016/S1574-6526(06)80014-3.
- 9 Emmanuel Hebrard, Brahim Hnich, Barry O’Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. In Manuela M. Veloso and Subbarao Kambhampati, editors, *20th National Conference on Artificial Intelligence, AAAI 2005*, pages 372–377. AAAI Press / The MIT Press, 2005. URL: <http://www.aaai.org/Library/AAAI/2005/aaai05-059.php>.
- 10 Paul Hudak. Domain-specific languages. *Handbook of programming languages*, 3(39-60):21, 1997.
- 11 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. doi:10.3233/SAT190116.
- 12 Linnea Ingmar, Maria Garcia de la Banda, Peter J. Stuckey, and Guido Tack. Modelling diversity of solutions. In *34th Conference on Artificial Intelligence, AAAI 2020*, pages 1528–1535. AAAI Press, 2020. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/5512>.
- 13 Matthias Klapperstueck, Frits de Nijs, Ilankaikone Senthoooran, Jack Lee-Kopij, Maria Garcia de la Banda, and Michael Wybrow. Exploring hydrogen supply/demand networks: Modeller and domain expert views. In Roland Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming - CP23*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, to appear, 2023.



- 14 Niklas Lauffer and Ufuk Topcu. Human-understandable explanations of infeasibility for resource-constrained scheduling problems. In *ICAPS 2019 Workshop XAIP*, 2019.
- 15 Jie Liu, Tim Dwyer, Guido Tack, Samuel Gratzl, and Kim Marriott. Supporting the problem-solving loop: Designing highly interactive optimisation systems. *IEEE Trans. Vis. Comput. Graph.*, 27(2):1764–1774, 2021. doi:10.1109/TVCG.2020.3030364.
- 16 João Marques-Silva and Alessandro Previti. On computing preferred muses and mcscs. In Carsten Sinz and Uwe Egly, editors, *17th International Conference on Theory and Applications of Satisfiability Testing - SAT 2014*, Lecture Notes in Computer Science, pages 58–74. Springer, 2014. doi:10.1007/978-3-319-09284-3\_6.
- 17 David Meignan, Sigrid Knust, Jean-Marc Frayret, Gilles Pesant, and Nicolas Gaud. A review and taxonomy of interactive optimization methods in operations research. *ACM Trans. Interact. Intell. Syst.*, 5(3):17:1–17:43, 2015. doi:10.1145/2808234.
- 18 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In Christian Bessière, editor, *13th International Conference on Principles and Practice of Constraint Programming - CP 2007*, Lecture Notes in Computer Science, pages 529–543. Springer, 2007. doi:10.1007/978-3-540-74970-7\_38.
- 19 Laurent Perron and Vincent Furnon. Or-tools. URL: <https://developers.google.com/optimization/>.
- 20 Thierry Petit and Andrew C. Trapp. Finding diverse solutions of high quality to constraint optimization problems. In Qiang Yang and Michael J. Wooldridge, editors, *24th International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 260–267. AAAI Press, 2015. URL: <http://ijcai.org/Abstract/15/043>.
- 21 Sophia Saller and Jana Koehler. Easy, adaptable and high-quality modelling with domain-specific constraint patterns. *CoRR*, abs/2206.02479, 2022. doi:10.48550/arXiv.2206.02479.
- 22 Ilankaikone Senthoooran, Matthias Klapperstueck, Gleb Belov, Tobias Czauderna, Kevin Leo, Mark Wallace, Michael Wybrow, and Maria Garcia de la Banda. Human-centred feasibility restoration in practice. *Constraints*, to appear, 2023.
- 23 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael J. Maher and Jean-Francois Puget, editors, *4th International Conference on Principles and Practice of Constraint Programming - CP98*, Lecture Notes in Computer Science, pages 417–431. Springer, 1998. doi:10.1007/3-540-49481-2\_30.
- 24 Pascal Van Hentenryck. *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA, 1999.
- 25 Willem-Jan van Hoeve and Irit Katriel. Global constraints. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 169–208. Elsevier, 2006. doi:10.1016/S1574-6526(06)80010-6.
- 26 Toby Walsh. Constraint patterns. In Francesca Rossi, editor, *9th International Conference on Principles and Practice of Constraint Programming - CP 2003*, Lecture Notes in Computer Science, pages 53–64. Springer, 2003. doi:10.1007/978-3-540-45193-8\_4.
- 27 Sameela Suharshani Wijesundara, Maria Garcia de la Banda, and Guido Tack. Addressing problem drift in UNHCR fund allocation. In Roland Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming - CP23*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, to appear, 2023.

# A Tale of Two Cities: Teaching CP with Story-Telling

Jimmy H.M. Lee ✉

Department of Computer Science and Engineering, The Chinese University of Hong Kong, China

---

## Abstract

This presentation is all about story-telling. It tells the story, the pedagogical innovations and experience of the co-development of three MOOCs on the subject of “Modeling and Solving Discrete Optimization Problems” by The Chinese University of Hong Kong (CUHK) and the University of Melbourne, each with unique culture and tradition. The MOOCs feature the Fable-based Learning approach, which is a form of problem-based learning encapsulated in a story plot. Each MOOC video begins with an animation that follows a story adapted from a Chinese classic. The heroes of the story encounter various optimization problems requiring technical assistance from two professors from modern time via a magical tablet granted to the heroes by a genie old man. The animation thus sets the stage for lecturing modeling and solving techniques. The new pedagogy provides a movie-like immersive experience to the learners, and aims at increasing learners’ motivation and interests as well as situating them in a coherent learning context. In addition to scriptwriting, animation production and embedding the teaching materials in the story plot, another challenge of the project is the remote distance between the two institutions as well as the need to produce all teaching materials in both (Mandarin) Chinese and English to cater for different geographic learning needs. The project and production spanned across 2016 and 2017. The MOOCs have been running recurrently on Coursera since January, 2017. We present learner statistics and feedback, and discuss our experience and preliminary observations of adopting the online materials in a Flipped Classroom setting at CUHK.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming

**Keywords and phrases** Constraint Programming, MOOCs, Fable-based Learning

**Digital Object Identifier** 10.4230/LIPICs.CP.2023.2

**Category** Invited Talk



© Jimmy H.M. Lee;

licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



# The CP-SAT-LP Solver

Laurent Perron ✉

Google, Paris, France

Frédéric Didier ✉

Google, Paris, France

Steven Gay ✉

Google, Paris, France

---

## Abstract

The CP-SAT-LP solver is developed by the Operations Research team at Google and is part of the OR-Tools [8] open-source optimization suite. It is an implementation of a purely integral Constraint Programming solver on top of a SAT solver using Lazy Clause Generation [11]. It draws its inspiration from the chuffed solver [4], and from the CP 2013 plenary by Peter Stuckey on Lazy Clause Generation [12].

The CP-SAT-LP solver improves upon the chuffed solver [4] in two main directions. First, it uses a simplex alongside the SAT engine. Second, it implements and relies upon a portfolio of diverse workers for its search part.

The use of the simplex brings the obvious advantages of a linear relaxation on the linear part of the full model. It also started the integration of MIP technology into CP-SAT-LP. This is a huge endeavour, as MIP solvers are mature and complex. It includes presolve – which was already a part of CP-SAT –, dual reductions, specific branching rules, cuts, reduced cost fixing, and more advanced techniques. It also allows to integrate tightly the research from the Scheduling on MIP community [3, 1, 9] along with the most advanced scheduling algorithms [13]. This has enabled breakthroughs in solving and proving hard scheduling instances of the Job-Shop problems [5] and Resource Constraint Project Scheduling Problems [6, 2].

Using a portfolio of different workers makes it easier to try new ideas and to incorporate orthogonal techniques with little complication, except controlling the explosion of potential workers. These workers can be categorized along multiple criteria like finding primal solutions – either using complete solvers, Local Search [7] or Large Neighborhood Search [10] –, improving dual bounds, trying to reduce the problem with the help of continuous probing. This diversity of behaviors has increased the robustness of the solver, while the continuous sharing of information between workers has produced massive speedups when running multiple workers in parallel.

All in all, CP-SAT-LP is a state-of-the-art solver, with unsurpassed performance in the Constraint Programming community, breakthrough results on Scheduling benchmarks (with the closure of many open problems), and competitive results with the best MIP solvers (on purely integral problems).

**2012 ACM Subject Classification** Applied computing → Operations research

**Keywords and phrases** Constraint Programming, Operations Research, Sat Solver

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.3

**Category** Invited Talk

---

## References

- 1 David Applegate and William Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991. doi:10.1287/ijoc.3.2.149.
- 2 Christian Artigues, Sophie Demasse, and Emmanuel Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. ISTE/Wiley, 2008. URL: <https://hal.science/hal-00482946>.
- 3 Egon Balas. Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic Discrete Methods*, 6(3):466–486, 1985.



© Laurent Perron, Frédéric Didier, and Steven Gay;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 3; pp. 3:1–3:2

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- 4 Geoffrey Chu, Peter J. Stuckey, Andreas Schutt, Thorsten Ehlers, Graeme Gange, and Kathryn Francis. the chuffed solver, June 2023. URL: <https://github.com/chuffed/chuffed>.
- 5 Junwen Ding, Zhipeng Lü, Chu-Min Li, Liji Shen, Liping Xu, and Fred Glover. A two-individual based evolutionary algorithm for the flexible job shop scheduling problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2262–2271, 2019.
- 6 Rainer Kolisch and Arno Sprecher. Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216, 1997. doi:10.1016/S0377-2217(96)00170-1.
- 7 Bjørnar Luteberget and Giorgio Sartor. Feasibility jump: an lp-free lagrangian mip heuristic. *Mathematical Programming Computation*, 15, March 2023. doi:10.1007/s12532-023-00234-8.
- 8 Laurent Perron and Vincent Furnon. Or-tools, March 2023. URL: <https://developers.google.com/optimization/>.
- 9 Maurice Queyranne. Structure of a simple scheduling polyhedron. *Math. Program.*, 58:263–285, 1993. doi:10.1007/BF01581271.
- 10 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael J. Maher and Jean-Francois Puget, editors, *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer, 1998. doi:10.1007/3-540-49481-2\_30.
- 11 Peter J. Stuckey. Lazy clause generation: Combining the power of sat and cp (and mip?) solving. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 5–9, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- 12 Peter J. Stuckey. Those who cannot remember the past are condemned to repeat it. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, pages 5–6, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- 13 Petr Vilim. Timetable edge finding filtering algorithm for discrete cumulative resources. In Tobias Achterberg and J. Christopher Beck, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 230–245, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

# Coupling CP with Deep Learning for Molecular Design and SARS-CoV2 Variants Exploration

Thomas Schiex   

Universite Fédérale de Toulouse, ANITI, INRAE, UR 875, 31326 Toulouse, France

---

## Abstract

---

The use of discrete optimization, including Constraint Programming, for designing objects that we completely understand is quite usual. In this talk, I'll show how designing specific biomolecules (proteins) raises new challenges, requiring solving problems that combine precise design targets, approximate laws, and design rules that can be deep-learned from data.

**2012 ACM Subject Classification** Computing methodologies → Artificial intelligence; Computing methodologies → Machine learning; Theory of computation → Constraint and logic programming; Computing methodologies → Learning in probabilistic graphical models

**Keywords and phrases** graphical models, deep learning, constraint programming, cost function networks, random Markov fields, decision-focused learning, protein design

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.4

**Category** Invited Talk

## 1 Introduction

Proteins are biomolecules that support most mechanisms in living organisms, from viruses to human beings. They already have major commercial applications in green chemistry (as enzymes) but also in the health domain (e.g., the anti-CoViD Regeneron™ antibodies are proteins). Most commercially used proteins are either natural proteins or engineered versions of natural proteins. To go beyond the repertoire of natural proteins, it is important to be able to reliably and efficiently design new proteins, with new capacities [9]. Proteins are defined by their amino acid sequence, a discrete object defined over an alphabet of 20 characters. Once the sequence of a protein is fixed, it can be encoded into a suitable microbe, enabling the cheap manufacturing of these complex microscopic assemblies.

Optimization is often used to design objects such as schedules, assignments, time-tables or packing, which we completely understand. Instead, proteins are tiny physical objects that live in the realm of quantum physics. Their behavior is hard to formally, precisely and concisely capture. Designing new proteins therefore requires to combine knowledge, expressed as approximate laws of physics, with targeted design constraints and criteria, in the context of large sets of data of past successful designs (natural proteins) that also embody the many hidden laws which a successfully expressed protein must satisfy.

## 2 Designing proteins and SARS-CoV2 variants with CP

In this talk, we will see how Cost Function Networks (CFNs), a weighted variant of Constraint Networks/CP) can help us design new proteins [1, 6]. Alone, CFNs can already capture both logical information (constraints) and numerical information, enabling the simultaneous representation of approximate laws of physics and design targets. By solving suitable instances of Weighted Constraint Satisfaction Problems, one can already produce protein sequences that can be tested *in silico* (with e.g., AlphaFold2 [7]), characterized experimentally, and lead to successful designs [8].



© Thomas Schiex;

licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 4; pp. 4:1–4:3

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

By leveraging the exhaustive enumeration capabilities of exact discrete solvers, it becomes possible to tackle previously unsolvable questions. To infect us, the SARS-Cov2 virus relies on its own spike protein, designed by evolution to be stable and efficiently bind to the human ACE2 receptor. Using a protein structure produced in the early months of 2020, we exhaustively enumerated SARS-CoV2 variants that would, in theory, bind to ACE2 and kept those that remained sufficiently stable. After a drastic selection among tenths of millions of predicted variants, 59 sequences were tested experimentally for affinity, infectivity, and resistance to antibodies, resulting in a list of non-yet-existing infectious therapeutic-antibodies-resistant variants that could be used to design vaccines proactively [3].

### 3 Learning how to play the Protein Design and Sudoku games

Because the laws of physics and modeling assumptions used in such approaches lead to approximate results, it becomes crucial to exploit the massive amount of data that has been produced by experimentalists in terms of natural protein structures and sequences. This raises the exciting question of learning CFNs describing the “quality” of sequences for a given protein structure to eventually learn how to design proteins. This problem is reminiscent of learning “how to reason” or “how to play Sudoku” which has been addressed by various recent decision-focused learning architectures. By leveraging a usual probabilistic interpretation of CFNs, we recently proposed a simple scalable learning architecture [4] that combines Deep Learning with an exact CFN solver (toulbar2 [2]) to learn how to design proteins (or how to play Sudoku) which outperforms existing architectures in terms of training time, data-efficiency and accuracy. Because solving the WCSP is NP-hard, powerful polynomial time relaxations then become handy [5].

---

#### References

- 1 David Allouche, Isabelle André, Sophie Barbe, Jessica Davies, Simon de Givry, George Katsirelos, Barry O’Sullivan, Steve Prestwich, Thomas Schiex, and Seydou Traoré. Computational protein design as an optimization problem. *Artificial Intelligence*, 212:59–79, 2014.
- 2 David Allouche, Simon De Givry, George Katsirelos, Thomas Schiex, and Matthias Zytnicki. Anytime hybrid best-first search with tree decomposition for weighted CSP. In *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Proceedings 21*, pages 12–29. Springer, 2015.
- 3 Mireia Solà Colom, Jelena Vucinic, Jared Adolf-Bryfogle, James W Bowman, Sébastien Verel, Isabelle Moczygemba, Thomas Schiex, David Simoncini, and Christopher D Bahl. Deep evolutionary forecasting identifies highly-mutated SARS-CoV-2 variants via functional sequence-landscape enumeration. *Research Square*, pages rs–3, 2022.
- 4 M. Defresne, S. Barbe, and T. Schiex. Scalable coupling of deep learning with logical reasoning. In *Proc. of the 32<sup>nd</sup> IJCAI*, Macau, A.S.R., China, 2023.
- 5 Valentin Durante, George Katsirelos, and Thomas Schiex. Efficient low rank convex bounds for pairwise discrete graphical models. In *International Conference on Machine Learning*, pages 5726–5741. PMLR, 2022.
- 6 Mark A Hallen and Bruce R Donald. Protein design by provable algorithms. *Communications of the ACM*, 62(10):76–84, 2019.
- 7 John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

- 8 Hiroki Noguchi, Christine Addy, David Simoncini, Staf Wouters, Bram Mylemans, Luc Van Meervelt, Thomas Schiex, Kam YJ Zhang, Jeremy RH Tame, and Arnout RD Voet. Computational design of symmetrical eight-bladed  $\beta$ -propeller proteins. *IUCrJ*, 6(1):46–55, 2019.
- 9 Ilan Samish, editor. *Computational Protein Design*. Humana New York, NY, 2017. doi: 10.1007/978-1-4939-6637-0.





# CP Solver Design for Maximum CPU Utilization

Petr Vilím  

ScheduleOpt, Nový Knín, Czech Republic

---

## Abstract

In this talk, I explain how to improve the performance of a solver without focusing on algorithms, search, propagation or parallelism. Performance is achieved instead with better CPU utilization, efficient code and more precise design of the solver itself.

In the words of Fedor G. Pikus [1], the time of “performance taking care of itself” is over. In today’s hardware the number of cores is increasing while the CPU clock speed has reached a plateau. Main memory access is slow in comparison to the CPU. And despite multiple memory cache levels, the CPU can easily become idle waiting for data from the memory, slowing down the computation considerably. Unfortunately, those trends are probably not going to change in the near future.

For those reasons we are witnessing revived interest in efficient code and performance-centered software design, especially in areas where the performance is critical: computer games, compilers, internet browsers, language interpreters (e.g. JavaScript or Python), etc.

The good news is that many of the tricks used in the above-mentioned areas, can be used in constraint programming as well. The bad news is that the performance has to be taken into account from the very beginning of the design. It is not possible to add it easily later. Sometimes, better performance can be achieved only by radical shifts in the design such as from object-oriented to data-oriented programming.

The design of a CP solver is not an exception in this regard. Without the efficient core of the CP solver, it is not possible to write truly efficient propagation or search algorithms. On the other hand, all algorithms in the solver must take the design of the solver into account and leverage it.

In this talk, I will describe what I consider the most important aspects of the design of *ScheduleOpt* solver. I will concentrate on the performance, but I will also mention other aspects such as ease of use, maintainability, and testing.

**2012 ACM Subject Classification** Mathematics of computing → Solvers; Theory of computation → Constraint and logic programming

**Keywords and phrases** Constraint Programming, Software Design, Efficient Code

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.5

**Category** Invited Talk

---

## References

- 1 F.G. Pikus. *The Art of Writing Efficient Programs: An advanced programmer’s guide to efficient hardware utilization and compiler optimizations using C++ examples*. Packt Publishing, 2021.



© Petr Vilím;

licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 5; pp. 5:1–5:1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



# Optimization of Short-Term Underground Mine Planning Using Constraint Programming

Younes Aalian ✉🏠

Département de mathématiques et de génie industriel, Polytechnique Montréal, Québec, Canada

Gilles Pesant ✉

Département de génie informatique et génie logiciel, Polytechnique Montréal, Québec, Canada

Michel Gamache ✉

Département de mathématiques et de génie industriel, Polytechnique Montréal, Québec, Canada

---

## Abstract

Short-term underground mine planning problems are often difficult to solve due to the large number of activities and diverse machine types to be scheduled, as well as multiple operational constraints. This paper presents a Constraint Programming (CP) model to optimize short-term scheduling for the Meliadine underground gold mine in Nunavut, Canada, taking into consideration operational constraints and the daily development and production targets of the mine plan. To evaluate the efficacy of the developed CP short-term planning model, we compare schedules generated by the CP model with the ones created manually by the mine planner for two real data sets. Results demonstrate that the CP model outperforms the manual approach by generating more efficient schedules with lower makespans.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Computing methodologies → Planning and scheduling

**Keywords and phrases** Mine planning, Constraint Programming, Short-term planning, Underground mine, Scheduling

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.6

## 1 Introduction

The mining industry is an important component of Canada's economic vitality. In 2019, its economic contribution was estimated at \$ 109 billion, or 5 % of Canada's GDP [10]. Mining projects involve a variety of operations that handle significant amounts of material and require substantial investment. Even small reductions in costs or increases in ore yield can have a considerable economic impact. These projects can generate significant profits when they are managed efficiently. Furthermore, the mining industry is evolving and transitioning towards automated mining. With the advent of new communication and data collection tools, mining operation data is becoming more easily accessible. This creates opportunities to develop new optimization tools that can use the available data to enhance the operational efficiency in mines.

The model presented in this study is designed for an underground gold mine. The price of gold is set by the market and the same for all mining companies. Among other things, 47 % of the gold produced in Canada is purchased by the London Bullion Market, which trades gold worldwide. The only options for gold mines to increase their profits is to reduce their operating costs. One way to reduce operating costs is to make better use of available resources. Minimizing the makespan indirectly reduces operating costs by doing more activities with the same equipment and reducing downtime.

Short-term planning in underground mines plays a crucial role in ensuring the profitability and success of mining operations. It involves allocating resources to activities and determining the sequence and start time of activities during each work shift over a planning horizon



© Younes Aalian, Gilles Pesant, and Michel Gamache;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 6; pp. 6:1–6:16

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ranging from one to two weeks [1, 3]. Currently, scheduling decisions in underground mines are typically made manually based on the planner's experience. Planning has been done manually for several reasons. First, communication systems in the underground mines were virtually non-existent. As a result, the exchange of information between the planning teams was essentially done between shifts. In addition, the management systems are not yet standardized in the mines, which means that information on geology, equipment maintenance and production management are found in different systems and the transfer of one system to another is not always trivial. However, manual planning is prone to errors and often results in infeasible schedules with low accuracy and efficiency. Therefore, developing a decision tool to optimize short-term scheduling in underground mines can help achieve high-quality schedules, improve mine productivity, and reduce reliance on the planner's experience, while ensuring technical and safety requirements are met [17]. In this paper, a Constraint Programming (CP) model is presented for the short-term scheduling of activities at the Meliadine underground gold mine located in Nunavut, Canada. The model considers both operational constraints and the mine's development and production targets to generate more practical and reliable schedules.

### **1.1 Why CP?**

Previous research has shown that Constraint Programming is an effective and efficient method for solving scheduling problems across various industries, including planning, scheduling, transportation, and automated systems [9]. CP uses a wide variety of variable types, functions, and global constraints to offer modeling at a high level of abstraction, making it a more flexible and intuitive approach than other model-based methods such as Mixed Integer Programming (MIP)[4]. Consequently CP models are more concise and require fewer decision variables and constraints which makes them an attractive tool for addressing large-scale scheduling problems. In the context of underground mining, the use of CP functions (as described in detail in Section 3) makes it easier to model operational constraints in the short-term scheduling problem, resulting in a more compact and efficient model.

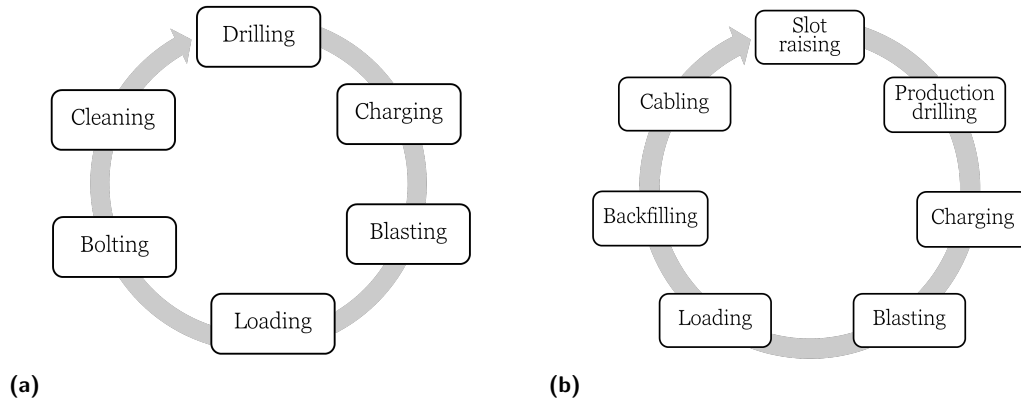
### **1.2 Plan of the Paper**

Section 2 describes the problem we address, Section 3 introduces the CP model we developed to solve it, and Section 4 discusses the outcomes of implementing the presented model on two actual data sets. Section 5 highlights the advantages of using CP for this short-term underground mine planning problem. Section 6 presents an overview of related computational approaches in the literature. Finally Section 7 concludes the paper.

## **2 Problem Description**

Underground mining operations involve two primary categories of activities: development and production. In order to access economically valuable ore deposits, development activities are conducted in waste rocks that lack financial value. Production activities take place in economically significant rocks located in areas referred to as *stopes* [5]. Mining activities occur in a cycle at one of several sites that serve as a workplace to perform these activities. Figures 1a and 1b show the development and production cycles with activities arranged in a sequence-dependent order. Table 1 provides the description of activities in the cycle, along with the required machine type. There are several machines available for each type of activity.

Each machine can be viewed as a renewable unitary resource, limited to performing one activity at a time. Short-term scheduling for underground mines includes assigning activities in the cycle to eligible machines and determining the start and end times for each activity [1].



■ **Figure 1** Typical development (a) and production (b) cycles in underground mining.

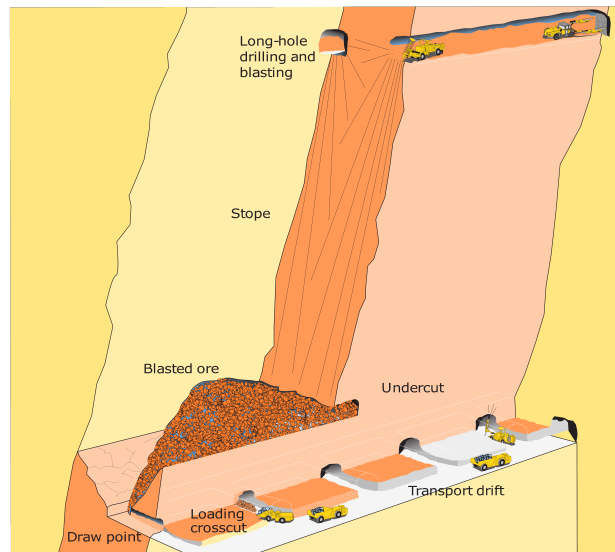
■ **Table 1** Activities and the required machine type in the cycle.

Activity	Machine	Description
Drilling	Drilling rigs	Drilling blast holes in the rock face
Charging	Anfo loader	Charging drilled holes with explosives
Loading	Scooptram	Removing broken rocks after blasting
Bolting	Bolter	Stabilizing drifts by installing bolts into the rock mass
Cleaning	Scooptram	Removing small rocks from the site (the gallery)
Cabling	Cabling machine	Reinforcing stope by installing steel cables into rock mass
Slot raising	Raise borer	Creating a vertical or inclined hole into the rock

There are several underground mining methods for extracting deep mineral deposits. The Meliadine mine uses the long-hole stoping method, which is one of the most commonly-used underground mining techniques that involve extracting a significant amount of material from each stope (Figure 2). This method is particularly suitable for large-scale and steeply dipping ore deposits with preferably tabular shapes. The long-hole stoping method begins with the development of main shafts or declines for transportation and ventilation purposes. Next, drill drives are excavated to access the intended location of the ore body and to create stopes. In each stope, production holes are drilled and charged with explosives. Once the blasting is completed, the fragmented rock is accessed through draw points developed at the bottom level of the stope. Scoop trams and trucks are used to collect the broken ore and transport it to the surface or other underground locations via drifts or ramps. In the final stage, the evacuated space in the stope is filled with a mixture of waste rocks and concrete to provide sufficient stability for the subsequent adjacent opening stopes [16].

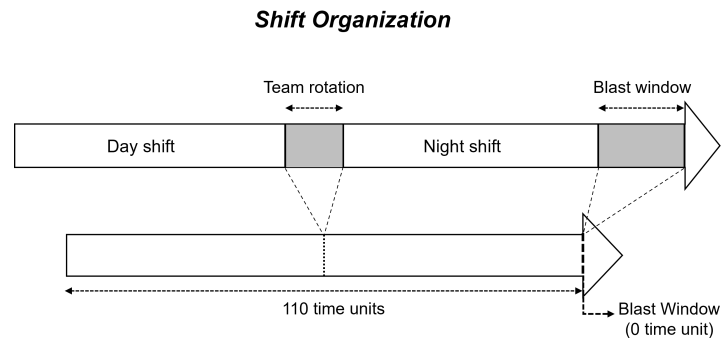
At Meliadine work is organized into a succession of day and night shifts, each lasting 55 time units. Blasting activities are performed only during designated blast windows. A blast window corresponds to the period between shifts in the morning, during which resources cannot be used by operators due to safety regulations. The team rotation takes approximately 1 hour and 30 minutes, and the blast window requires roughly 4 hours and 30 minutes, including the time needed for team rotation and gas clearance (18 working hours for both

## 6:4 Short-Term Underground Mine Planning Using CP



■ **Figure 2** Typical representation of the long-hole stoping operation [8].

day and night shifts + 1 hour 30 minutes for team rotation at the end of day shift + 4 hours 30 minutes for blast window at the end of night shift = 24 hours). The shift at the end of which team rotation occurs is referred to as the day shift, while the subsequent shift, which includes the blasting window at the end, is known as the night shift. Figure 3 illustrates the shift organization in the studied underground mine. Mining activities are preemptive as they can be interrupted at the end of each shift and continue in the next shift.



■ **Figure 3** Timeline of alternating day and night shifts including time to rotate the teams and to perform blasting (above). Its representation in the CP model (below).

Short-term planning at the Meliadine mine incorporates several key performance indicators (KPI) such as progress of development rounds in the drift, total length of production holes drilled in the stope, and total amount of material mucked from the stope to meet the medium-term planning goals. The KPI values vary monthly and are updated every three months by the medium-term mine planner. Development and production constraints will be introduced in our CP model to consider the defined KPIs in short-term scheduling.

### 3 How our Problem is Modeled in CP

An optimization model is developed using Constraint Programming (CP) for short-term underground mine scheduling, taking into account operational requirements of underground mining operations. Additional constraints are introduced to ensure that the mine planning development and production targets are met and that practical and reliable short-term schedules are generated. In other words, the produced schedule determines the detailed execution of mining activities in underground operations considering the required daily rates of development and production. It is important to note that the same model can be used for both development and production activities in underground mining, which ensures consistency and accuracy in short-term scheduling.

CP Optimizer (CPO) from IBM ILOG Optimization Studio [9] was used to create the model presented in this article. In this CPO model, *interval variables* are used to represent activities, each with several related *optional interval variables* depicting the choice of resource. Optional interval variables include a Boolean status reflecting the fact that the corresponding activity is present or absent from the solution (i.e. not considered by the constraints). The ordering of resources can be represented by a set of interval variables, known as a *sequence variable*. This sequence variable is used in the scheduling model to prevent activities in the sequence from overlapping in time. More formally, an interval variable  $a$  is defined by a start time  $s$  and an end time  $e$ , which are non-negative integer values, such that  $a \in \{[s, e] \mid s, e \in \mathbf{N}, e \geq s\}$ . Optional interval variable  $b$  is presented such that  $b \in \{\emptyset\} \cup \{[s, e] \mid s, e \in \mathbf{N}, e \geq s\}$ . Additionally the developed CPO model uses various functions and constraints that are described as follows [9]:

- **endOf**: A function that provides the end value of an interval variable if it exists, or else returns zero.
- **alternative**: This constraint ensures that if a given interval variable is present, then only one related optional interval variable is chosen with the same start and end values.
- **noOverlap**: This constraint is used to ensure that a set of interval variables defined by a sequence variable do not overlap, while maintaining a minimum distance between them as specified by a transition distance matrix.
- **endBeforeStart**: This constraint guarantees that if two interval variables are present, then the first ends before the second starts, with an optional minimum delay between them.
- **forbidExtent**: This constraint makes sure that an interval variable cannot overlap with a forbidden region where the value of the step function is zero. As a result, the interval variable must either end before the forbidden region or start after it.
- **stepAtEnd**: This step function returns an elementary cumulative function with a non-negative integer value at the end of an interval variable. Such functions model a known function of time, such as the resources used during a particular time period, by returning a non-negative integer value (height of the elementary function) within the range of the interval variable and zero outside of it.
- **cumulFunction**: This expression models a known function of time, such as the cumulative amount of resources used by an activity during a specific time period.
- **alwaysIn**: This constraint restricts the potential values of a cumulative function to a specific range during a time interval.



## 6:6 Short-Term Underground Mine Planning Using CP

Tables 2 and 3 present lists of sets, parameters, and variables used in the CP model, along with their corresponding descriptions.

■ **Table 2** Sets and parameters of the CP model.

Set	Description
$J$	Index set of activities
$M$	Index set of all available equipment
$M_j$	Index set of eligible machines to perform activity $j$
$A_j$	Index set of activities that must occur after activity $j$
$B$	Index set of blast activities
$T$	Index set of time windows (starting at 1)
Parameter	Description
$p_j$	Processing time of activity $j$
$D$	Matrix of transition time between sites where the value of its element is equal to 0 for the same site and greater than 0 otherwise
$d_j$	Development (meter) of activity $j$
$h_j$	Production hole drilling (meter) of activity $j$
$o_j$	Stope ore mucking (ton) of activity $j$
$s_t$	Starting time of time window $t$
$e_t$	Ending time of time window $t$
$\underline{d}$	Lower bound for daily development
$\bar{d}$	Upper bound for daily development
$\underline{h}$	Lower bound for daily hole drilling
$\bar{h}$	Upper bound for daily hole drilling
$\underline{o}$	Lower bound for daily ore mucking
$\bar{o}$	Upper bound for daily ore mucking
$Blast\_calendar$	The time periods during which only blasting activities are permitted (all activities except blasting are forbidden to be performed during these periods)

■ **Table 3** Decision variables of the CP model.

Variable	Description
$Y_j$	Interval variable for activity $j$
$X_{jm}$	Optional interval variable to perform activity $j$ using machine $m$
$S_m$	Sequence variable for machine $m$ ( $S_m = \{X_{jm} \mid j \in J\}$ )
$Q^d$	Integer variable for total development in drifts
$Q^h$	Integer variable for total amount of production hole drilling in stopes
$Q^o$	Integer variable for total amount of ore material mucked from stopes

The CP model is given as (1)-(11):

### Objective function

$$\text{Minimize } \max_{j \in J}(\text{endOf}(Y_j)) \quad (1)$$

### Constraints

$$\text{alternative}(Y_j, X_{jm} \mid m \in M_j) \quad \forall j \in J \quad (2)$$

$$\text{noOverlap}(S_m, D) \quad \forall m \in M \quad (3)$$

$$\text{endBeforeStart}(Y_j, Y_i) \quad \forall j \in J, i \in A_j \quad (4)$$

$$\text{forbidExtent}(Y_j, \text{Blast\_calendar}) \quad \forall j \in J \setminus B \quad (5)$$

$$\text{cumulFunction}(Q^d) = \sum_{j \in J} \text{stepAtEnd}(Y_j, d_j) \quad (6)$$

$$\text{alwaysIn}(Q^d, s_t, e_t, t \times \underline{d}, t \times \bar{d}) \quad \forall t \in T \quad (7)$$

$$\text{cumulFunction}(Q^h) = \sum_{j \in J} \text{stepAtEnd}(Y_j, h_j) \quad (8)$$

$$\text{alwaysIn}(Q^h, s_t, e_t, t \times \underline{h}, t \times \bar{h}) \quad \forall t \in T \quad (9)$$

$$\text{cumulFunction}(Q^o) = \sum_{j \in J} \text{stepAtEnd}(Y_j, o_j) \quad (10)$$

$$\text{alwaysIn}(Q^o, s_t, e_t, t \times \underline{o}, t \times \bar{o}) \quad \forall t \in T \quad (11)$$

Objective (1) of the CP model is to minimize the makespan. Constraint (2) ensures that only one optional variable is chosen for an interval variable i.e. only one machine (with the appropriate type) is used to perform a given activity. Constraint (3) prevents machines from being used simultaneously, meaning that each machine can only be assigned to one activity at a time. Constraint (4) takes into account the order in which activities must be performed at a site, with most activities having only one predecessor and some having none. It is important to note that the site where each activity must be carried out is predefined in the input data. Therefore, all activities can be executed in their respective predetermined sites.

Constraint (5) is used to ensure that only blasting activities occur during designated blast windows. In order to model the blasting constraint in the CP model, the day and night work shifts are compressed into a 110-time unit period (each shift consists of 55 time units), where each time unit represents 10 minutes in the real world. This compression allows for blasting activities with a length of zero time units to occur only at the end of compressed periods, every 110 time units (see Figure 3). Multiple blasting activities can be performed at the same time during each blasting window.

Constraints (6) and (7) are introduced to ensure that the progress of development rounds each day (measured in meters per day) is maintained within specific limits based on the defined development target. To model these development constraints, we define time windows  $[s_t, e_t)$  each representing a day in the schedule. For each time window, we establish cumulative upper and lower bounds (based on daily bounds) for total development in drifts (in meters) that must be achieved. Next, we use the `cumulFunction` to model the cumulative amount of development per meter and apply the `alwaysIn` constraint to ensure that the cumulative function stays within the target value bounds for each time window. The function `stepAtEnd(i, j)` returns an elementary cumulative function with a step of height  $j$  (a non-negative integer value) at the end of interval variable  $i$ . The presented development

constraints aim to achieve the desired daily development target in the generated schedule. Furthermore, by using the `cumulFunction` in this constraint, the model is able to flexibly compensate in the following days for any shortfall in achieving the daily development goal (see e.g. Figure 6a). This feature of the constraints closely resembles what is taken into account in actual short-term underground mine planning, making the model more practical for real-world operations.

Constraints (8) and (9) model the production drilling constraint to ensure that the amount of production holes drilled in the stope per day (measured in meters per day) is restricted within certain bounds, defined based on the production drilling objectives. Furthermore, Constraints (10) and (11) are used to model the stope mucking constraint, which ensures that the amount of ore material mucked from stopes each day (measured in tons per day) is maintained within specific limits determined based on the production target. These constraints (Constraints (8)-(11)) aim to achieve the production plan in the produced schedule by controlling the daily amount of production holes drilled and ore mucked. Similar to the development constraints, the production constraints also allow for making up for shortfalls in meeting the daily production goal. An activity can perform either development or production depending on the type of cycle. If the activity is part of a development cycle, it can have development ( $d_j$ ), and if it is involved in a production cycle, it can have either production hole drilling ( $h_j$ ) or stope mucking ( $o_j$ ). The development cycle includes activities that are performed in waste rocks lacking financial value to access economically valuable deposits, while the production cycle is conducted in valuable rocks to extract ore material from the stope.

In the presented short-term mine planning model, operational development and production targets (KPIs) are considered to achieve the tactical decisions made at the medium-term planning level. Specifically, tactical decisions in underground mine planning are typically associated with defining the extraction sequence over a planning horizon of one to three months [7].

## 4 Implementation and Results

The experiments were conducted on a computer featuring an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz and 16 GB of RAM. The CP models were solved using the Constraint Programming Optimizer in IBM ILOG CPLEX Optimization Studio version 12.8.0.

The model was tested on two real data sets collected from the Meliadine underground gold mine in Nunavut, Canada. Both data sets involve scheduling activities for a roughly one-week planning horizon. The first data set (Instance 1) relates to development operations, which consist of 15 machines and 291 activities to be performed across 18 sites. The total advancement achieved by all available development rounds in this instance is equal to 188 meters. Specifically, each round (cycle) results in approximately 4 meters of advancement in the development drift. The second data set (Instance 2) concerns production operations and includes 27 machines, 185 activities, and 27 sites. In this instance, a total of 1500 meters of production holes have been drilled across all accessible stopes, resulting in the extraction of 27,000 tons of ore material. The available resources are categorized into different equipment types, and the number of each type is reported in Table 4. Although the developed CP model takes into account both development and production activities, there was no data available (in the mine) that included both activities together. Therefore, we applied our model separately to two different datasets: one for development and another for production.

■ **Table 4** Number of machines per equipment type for Instances 1 and 2.

Equipment type	Instance 1	Instance 2
Scooptram	2	7
Bolter	6	-
Scooptram clean face	1	-
Jumbo	3	-
Anfo loader	3	3
Truck	-	7
Raise borer	-	1
Production drill rig	-	4
Cabling machine	-	5

The results obtained by implementing the CP model on Instances 1 and 2 are presented in the following subsections.

#### 4.1 Instance 1

Table 5 presents the results of schedules generated for Instance 1 with different daily development upper bounds ( $\bar{d}$ ) in the development constraint. All the models in the table are solved to optimality in a short amount of time. As the primary objective of the scheduling model is to minimize the makespan, the lower bound on daily development specified in the development constraint is readily satisfied. Therefore to evaluate the effect of different development targets on the resulting schedule, we only modify the upper bound value for the total amount of development to be accomplished per day.

■ **Table 5** Results of different CP models on Instance 1.

Model	Development upper bound ( $\bar{d}$ )	Makespan	Solving time
1	24	882	12 sec
2	28	772	13 sec
3	32	678	12 sec
4	36	635	13 sec
5	40	600	11 sec
6	44	600	10 sec
7	$\infty$	600	10 sec

As can be seen from Table 5, increasing the upper bound in the development constraint results in lower makespans in the produced schedule. Furthermore, the schedule makespan remains unchanged for bounds greater than 40. Therefore,  $\bar{d} = 40$  can be considered a suitable daily development target for generating a short-term schedule on Instance 1. Interestingly, this value coincides with the daily development target employed by the human planner at the mine – our model confirms this empirical choice. Figure 4 displays the location-based Gantt chart for the short-term schedule generated on Instance 1 with  $\bar{d} = 40$ .

The daily and cumulative development resulting from the schedule produced using Model 3 on Instance 1 with  $\bar{d} = 32$  are displayed in Figures 5a and 5b. As seen in Figure 5a, the maximum daily development limit of 30 meters is respected, resulting in a total cumulative development of 188 meters in six days (Figure 5b).

6:10 Short-Term Underground Mine Planning Using CP

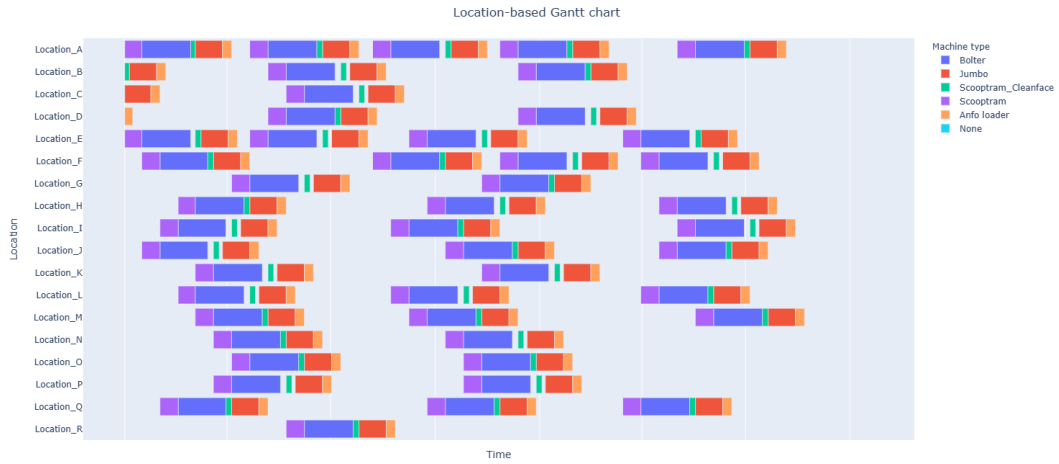


Figure 4 Location-based Gantt chart for the generated schedule on Instance 1.

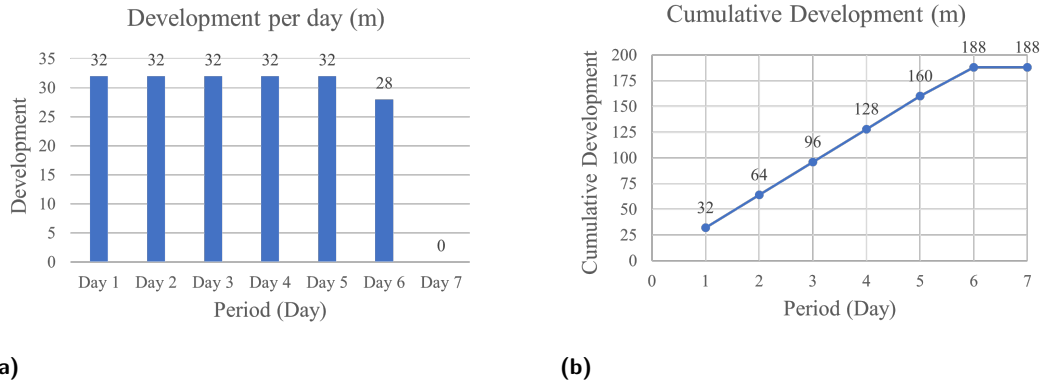
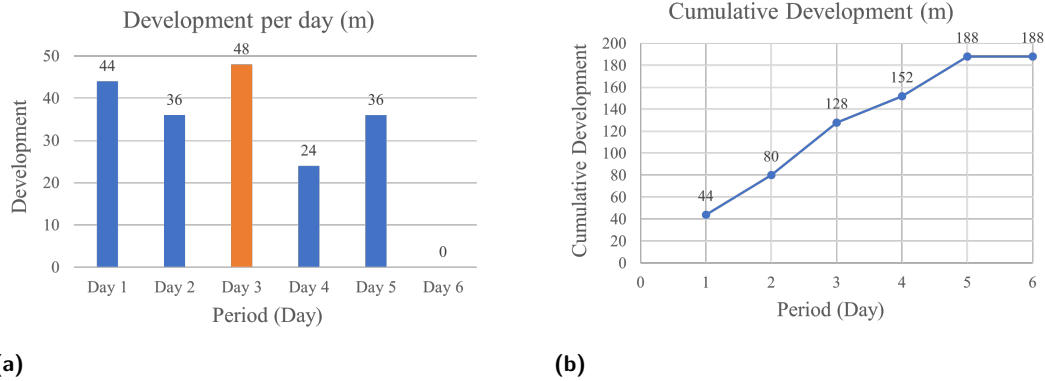


Figure 5 Daily (a) and cumulative (b) development in Model 3 ( $\bar{d} = 32$ ) on Instance 1.

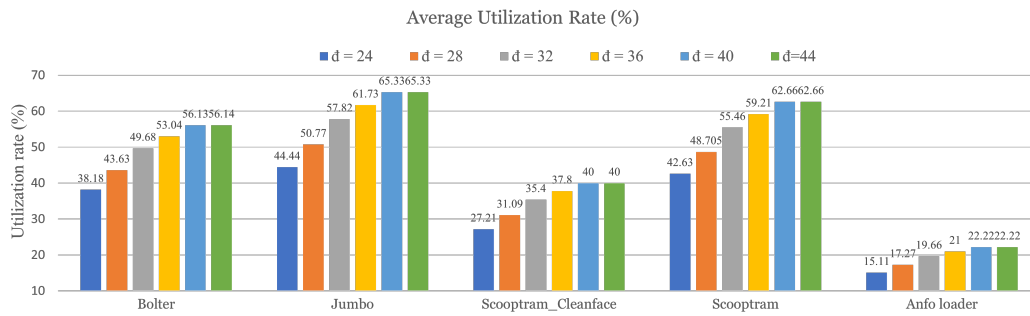
Figures 6a and 6b show the daily and cumulative development obtained from Model 6 on Instance 1 with  $\bar{d} = 44$ . According to Figure 6a, 36 meters of development are achieved on Day 2, which is lower than the maximum daily target of 44 meters. However, this shortfall is made up on Day 3 by completing 48 meters of development, above the maximum daily target. In other words, 48 meters of development are completed on Day 3 to compensate for the shortfall on Day 2. After Day 3, it is not possible to meet the maximum daily goal due to the limited number of drifts available. As shown in Figure 6b, the total cumulative development of 188 meters is reached in five days. This feature of the development constraints in the CP model can be practical for short-term planning in underground mines, where operational restrictions or a relatively small number of accessible drifts (sites) prevent the achievement of the development target on certain days.

Figure 7 shows the comparison of the average utilization rate of several machine types in schedules produced using CP models on Instance 1 with different  $\bar{d}$  for the development constraint. The utilization rate of a machine is the total amount of time units during which the machine was actively operating at the site relative to the total amount of time for which it was available for use. As can be seen from this figure, increasing  $\bar{d}$  results in a higher



■ **Figure 6** Daily (a) and cumulative (b) development in Model 6 ( $\bar{d} = 44$ ) on Instance 1.

average utilization rate of machines in the schedule. This is due to the fact that larger  $\bar{d}$  values lead to more compact schedules with lower makespans, which in turn, reduces waiting time for machines.



■ **Figure 7** Average utilization rate of machines in schedules with different development bounds ( $\bar{d}$ ) on Instance 1.

## 4.2 Instance 2

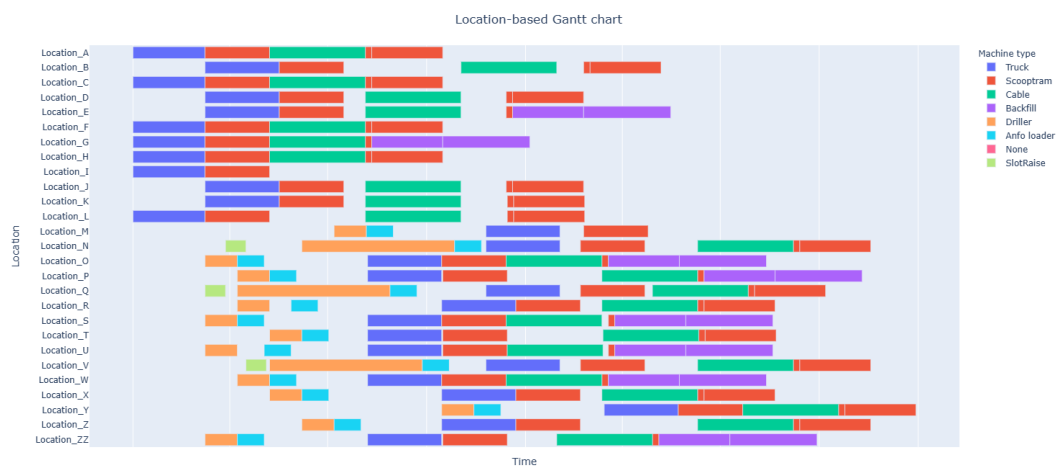
Table 6 displays the makespan of schedules generated by implementing different models on Instance 2, with distinct upper bounds for the daily production drilling ( $\bar{h}$ ) and stope mucking ( $\bar{o}$ ) in production constraints. According to Table 6, reducing the upper bound values in production constraints leads to longer makespans in the generated schedule. Specifically, for the production drilling constraint, the suitable  $\bar{h}$  value is 400 meters, as it leads to the lowest makespan value that remains unchanged for larger upper bounds. Similarly, for the stope mucking constraint,  $\bar{o} = 6,000$  appears to be an appropriate stope mucking target for the schedule generated on Instance 2. Figure 8 shows the location-based Gantt chart for the created schedule on Instance 2 with  $\bar{o} = 6,000$ .

Figures 9a and 9b present the daily and cumulative production drilling rates in the schedule generated using Model 5 with  $\bar{h} = 500$  on Instance 2. Figure 9a demonstrates that the drilling rate exceeds the daily limit by reaching 600 meters on Day 2 to compensate for the shortfall on Day 1. As depicted in Figure 9b, the total production drilling of 1500 meters is achieved within four days.

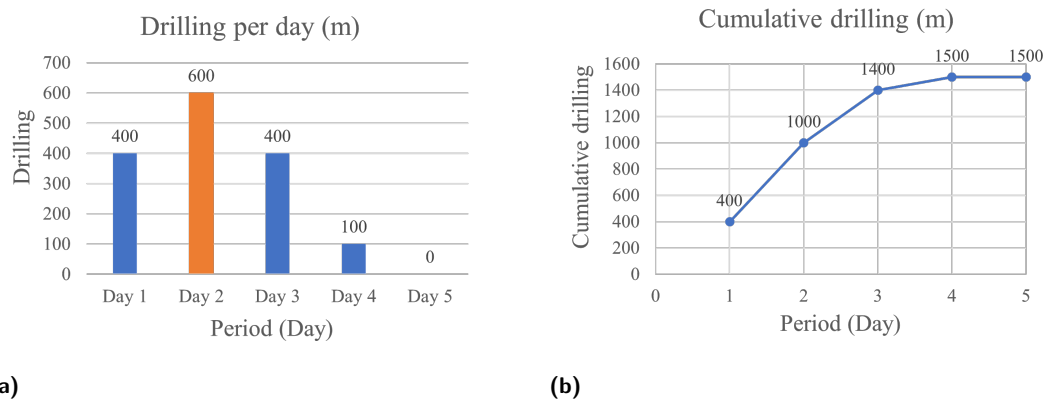
## 6:12 Short-Term Underground Mine Planning Using CP

■ **Table 6** Results of different CP models on Instance 2.

Model	Production drilling upper bound ( $\bar{h}$ )	Stope mucking upper bound ( $\bar{o}$ )	Makespan	Solving time
1	$\infty$	$\infty$	730	16 sec
2	200	–	1060	17 sec
3	300	–	840	16 sec
4	400	–	730	16 sec
5	500	–	730	16 sec
6	–	4000	881	17 sec
7	–	5000	771	16 sec
8	–	6000	730	17 sec
9	–	7000	730	17 sec



■ **Figure 8** Location-based Gantt chart for the generated schedule on Instance 2.



■ **Figure 9** Daily (a) and cumulative (b) drilling in Model 5 ( $\bar{h} = 500$ ) on Instance 2.

Figures 10 and 11 compare the average utilization rate of several machine types in schedules produced using CP models on Instance 2, with different values for  $\bar{h}$  and  $\bar{o}$ , respectively. According to these figures, the average utilization rate of machines increases for schedules with higher upper bounds in production constraints.

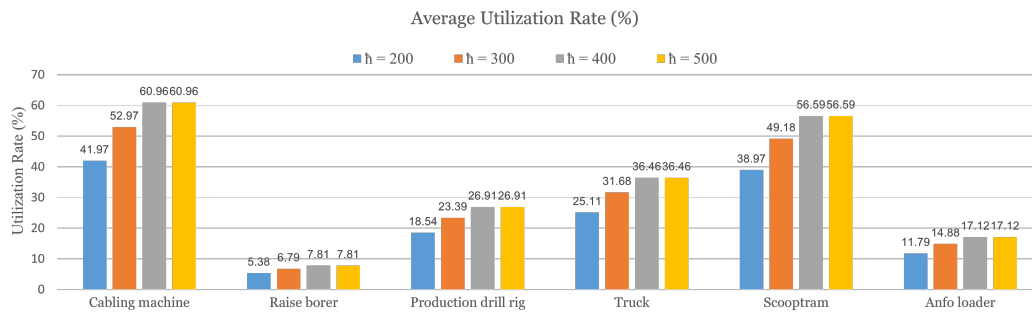


Figure 10 Average utilization rates of machines in schedules with different production drilling bounds ( $\bar{h}$ ) on Instance 2.

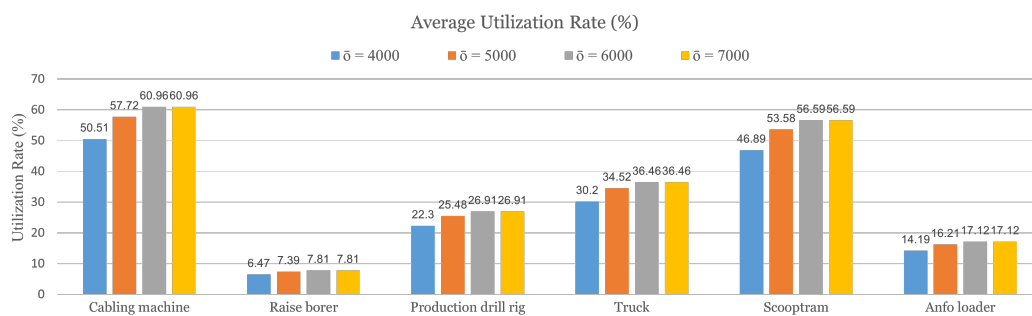


Figure 11 Average utilization rates of machines in schedules with different stope mucking bounds ( $\bar{d}$ ) on Instance 2.

## 5 Added Value of CP

Constraint Programming allowed us to efficiently address short-term underground mine planning by quickly producing optimal schedules minimizing the makespan. Moreover the model identifies  $\bar{d} = 40$  as the appropriate daily development target, which aligns with the value selected by the mine planner and confirms the practice of setting the development upper bound at 40. Additionally, the model can explore what-if scenarios by varying parameter values, such as the impact of changing daily development or production targets in the model on machine utilization rates in the generated schedule. These results demonstrate the practicality and efficiency of using the CP model for short-term scheduling in real-world underground mining operations.

### 5.1 Comparison of CP model and manual approach

In order to demonstrate the effectiveness of our optimization model, we compared the short-term schedules produced by the CP model with those manually created by the mine planner for the same instance. Since a detailed schedule of activities with similar time fidelity to the schedule produced using the CP model was not provided in the studied mine, we only compared the schedule makespan. In particular, we compared the number of shifts required to complete all activities in the generated short-term schedule using the CP model and manual approach for both Instances 1 and 2, as shown in Table 7. The development KPI considered for scheduling activities in Instance 1 is 40 meters per day (m/day). In Instance



## 6:14 Short-Term Underground Mine Planning Using CP

2, the production drilling KPI is 400 m/day, and the stope mucking KPI is 6,000 tons per day. As previously mentioned, each day consists of two working shifts, where each shift is equivalent to 55 time units in the CP model.

■ **Table 7** Comparison of schedule makespan between CP model and manual approach for Instances 1 and 2.

Instance	CP model	Manual approach
1	11 Shifts	14 Shifts
2	14 Shifts	16 Shifts

Table 7 shows that the CP approach outperforms the manual scheduling method on both Instances 1 and 2 by creating more compact schedules with lower makespans (based on the number of shifts) while satisfying the daily development and production targets (KPIs). Additionally, the CP models are quickly solved to optimality, making it an efficient tool for mine planners to rapidly generate updated short-term schedules whenever changes occur in the underground mine plan.

The results of this study demonstrate advantages of the developed CP model for optimizing short-term planning in underground mines and reducing the reliance on manual scheduling, which is highly dependent on the planner's experience. Moreover, the CP model can be easily adjusted to accommodate or exclude additional activity types and related constraints based on the specific requirements of underground mining operations.

## 6 Literature review

Short-term underground mine planning models are often difficult to solve (NP-hard) due to various operational constraints to consider and to the large number of variables involved. However there has been notable research interest in developing new mathematical models and algorithms to optimize short-term scheduling in underground mines.

Nehring et al. (2010) designed a MIP model to optimize the short-term scheduling and allocation of loader-trucks in sublevel stoping mines. The model allows for the reallocation of equipment in response to changes in underground operations. The proposed model was applied to a copper mine, demonstrating satisfactory results in terms of tonnage deviations from predetermined amounts throughout the planning period [11]. O'Sullivan and Newman (2015) introduced an Integer Programming (IP) model for scheduling activities in an Irish lead and zinc underground mine to maximize the discounted amount of produced metal. Both exact and heuristic solutions were used to reduce the number of variables in the model. Additionally, an optimization-based decomposition heuristic was developed to generate feasible schedules in less computation time for complicated problem instances [12]. Song et al. (2015) developed a decision support tool to determine the scheduling of activities in underground mines. The tool was tested on a real mine dataset in Finland and significantly decreased the makespan compared to manual scheduling methods, thereby improving operational performance. However, the proposed method did not take into account uncertainty related to unexpected activities in underground operations [15].

Schulze and Zimmermann (2017) introduced a solution approach for short-term production scheduling in underground mining. The developed approach assigns staff and machines to mining activities while considering operational constraints with the goal of minimizing deviations from targeted production in a potash mine. The method was tested on various instances and demonstrated superior performance when compared to manual scheduling [13]. Seifi et al. (2019) proposed a two-stage solution approach for scheduling machines and staff in an underground potash mine in Germany. The first step involves solving the relaxation

of the MIP model, and in the second step, a heuristic algorithm is used to modify the solutions obtained from the relaxation model to achieve feasible schedules. The experiments conducted on real-world datasets show that the developed approach outperforms the heuristic procedure presented by Schulze and Zimmermann (2017)[14]. Wang et al. (2020) utilized a genetic algorithm (GA) for optimizing the scheduling of equipment used in underground mining. A Non-Linear Programming (NLP) model is presented with a significant number of decision variables associated with multiple mining sites and equipment types [17]. A MIP model was presented by Campeau and Gamache (2020) to optimize short-term planning in underground mines. The goal was to maximize material extraction while ensuring a minimum ore production rate to keep the mill active. The model considers operational and resource constraints to generate feasible schedules. When applied to a gold mine data set, the model produced an optimal short-term schedule [5]. Campeau et al. (2022) introduced a novel MIP model to address short- and medium-term planning in underground mines. The model integrated continuous variables for time discretization, resulting in realistic schedules. The effectiveness of the model was demonstrated by applying it to a dataset from a Canadian gold mine, which produced promising results [7].

Over the last few years, several CP approaches have been proposed to tackle the short-term underground mine planning problem. A model using CP was suggested by Astrand et al. (2018) for scheduling a mobile fleet in underground operations, which was tested on data from an actual underground mine [2]. Astrand et al. (2020) extended the previously developed CP model by incorporating the time it takes for mobile machinery to travel between different sites in an underground cut-and-fill mine. They also proposed a revised CP model with compressed blasting time and post-processed solutions to obtain schedules for the primary problem. In order to improve the quality of schedules and reduce computation time, a specialized neighborhood definition was implemented in a Large Neighborhood Search (LNS) algorithm. The effectiveness of this algorithm was assessed using several instances of an underground mine in Sweden. The outcome showed that the suggested method successfully enhanced the initial feasible solution and generated high-quality schedules [3]. Campeau and Gamache (2022) presented a CP model for short- and medium-term planning in underground mining. They evaluated the model's ability to address long-term production planning objectives by testing it on five data sets from a Canadian underground gold mine, considering a planning horizon of up to one year. The outcomes revealed that the CP model was superior to the equivalent MIP model in terms of computational efficiency and application [6].

These previous CP approaches for short-term underground mine planning exhibit a limited ability to incorporate daily mine planning development and operational goals during the short-term scheduling process. To overcome this limitation, this paper introduced a CP model for the short-term scheduling of activities in underground mining that takes into consideration operational constraints and the development and production targets of the mine plan to generate more practical and reliable schedules.

## **7** Conclusion

This paper presented a CP model that takes into account various operational constraints and daily development and production targets for short-term scheduling optimization in underground mines. The model was tested on two data sets from the Meliadine gold mine using the long-hole stoping mining method. We conducted a comparative analysis of the schedules generated by our CP model and those created manually by the mine planner. The experiments showed that the CP model outperforms the manual approach, resulting

in more efficient schedules with lower makespans. Results highlight the potential benefits of implementing the CP model in actual underground mining operations to improve both development and production through optimized short-term mine planning. Underground mines are somewhat unpredictable environments which may affect how long an activity actually takes. For future work, it could be beneficial to incorporate uncertainty in activity durations which would improve the robustness of the short-term schedule.

---

## References

- 1 Max Åstrand, Mikael Johansson, and Jenny Greberg. Underground mine scheduling modelled as a flow shop: a review of relevant work and future challenges. *Journal of the Southern African Institute of Mining and Metallurgy*, 118(12):1265–1276, 2018.
- 2 Max Åstrand, Mikael Johansson, and Alessandro Zanarini. Fleet scheduling in underground mines using constraint programming. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15*, pages 605–613. Springer, 2018.
- 3 Max Åstrand, Mikael Johansson, and Alessandro Zanarini. Underground mine scheduling of mobile machines using constraint programming and large neighborhood search. *Computers & Operations Research*, 123:105036, 2020.
- 4 Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media, 2001.
- 5 Louis-Pierre Campeau and Michel Gamache. Short-term planning optimization model for underground mines. *Computers & Operations Research*, 115:104642, 2020.
- 6 Louis-Pierre Campeau and Michel Gamache. Short-and medium-term optimization of underground mine planning using constraint programming. *Constraints*, 27(4):414–431, 2022.
- 7 Louis-Pierre Campeau, Michel Gamache, and Rafael Martinelli. Integrated optimisation of short-and medium-term planning in underground mines. *International Journal of Mining, Reclamation and Environment*, 36(4):235–253, 2022.
- 8 Atlas Copco. Mining methods in underground mining. *Atlas Copco: Nacka, Sweden*, 2007.
- 9 Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. Ibm ilog cp optimizer for scheduling: 20+ years of scheduling with constraints at ibm/ilog. *Constraints*, 23:210–250, 2018.
- 10 B. Marshall. Facts and figures : The state of canada’s mining industry. Technical report, The Mining Association of Canada, 2020.
- 11 Micah Nehring, Erkan Topal, and Peter Knights. Dynamic short term production scheduling and machine allocation in underground mining using mathematical programming. *Mining Technology*, 119(4):212–220, 2010.
- 12 Dónal O’Sullivan and Alexandra Newman. Optimization-based heuristics for underground mine scheduling. *European Journal of Operational Research*, 241(1):248–259, 2015.
- 13 Marco Schulze and Jürgen Zimmermann. Staff and machine shift scheduling in a german potash mine. *Journal of Scheduling*, 20:635–656, 2017.
- 14 Cinna Seifi, Marco Schulze, and Jürgen Zimmermann. A two-stage solution approach for a shift scheduling problem with a simultaneous assignment of machines and workers. In *Mining Goes Digital*, pages 377–385. CRC Press, 2019.
- 15 Zhen Song, Håkan Schunnesson, Mikael Rinne, and John Sturgul. Intelligent scheduling for underground mobile mining equipment. *PloS one*, 10(6):e0131003, 2015.
- 16 Farzad Sotoudeh, Micah Nehring, Mehmet Kizil, Peter Knights, and Amin Mousavi. Production scheduling optimisation for sublevel stoping mines using mathematical programming: A review of literature and future directions. *Resources Policy*, 68:101809, 2020.
- 17 Hao Wang, Victor Tenorio, Guoqing Li, Jie Hou, and Nailian Hu. Optimization of trackless equipment scheduling in underground mines using genetic algorithms. *Mining, Metallurgy & Exploration*, 37:1531–1544, 2020.


# Exploiting Configurations of MaxSAT Solvers

Josep Alòs ✉ 

Logic & Optimization Group (LOG), University of Lleida, Spain

Carlos Ansótegui ✉ 

Logic & Optimization Group (LOG), University of Lleida, Spain

Josep M. Salvia ✉ 

Logic & Optimization Group (LOG), University of Lleida, Spain

Eduard Torres ✉ 

Logic & Optimization Group (LOG), University of Lleida, Spain

---

## Abstract

In this paper, we describe how we can effectively exploit alternative parameter configurations to a MaxSAT solver. We describe how these configurations can be computed in the context of MaxSAT. In particular, we experimentally show how to easily combine configurations of a non-competitive solver to obtain a better solving approach.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming

**Keywords and phrases** maximum satisfiability, maxsat evaluation, automatic configuration

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.7

**Funding** This work was supported by MCIN/AEI/10.13039/501100011033 (*Grant: PID2019-109137GB-C21*), Agència de Gestió d'Ajuts Universitaris i de Recerca (AGAUR), Departament d'Empresa i Coneixement de la Generalitat de Catalunya (*Grant: 2022 FI\_B 00010*)

**Acknowledgements** We want to thank Alexander Nadel for sharing the solver TT-Open-WBO with the configurable parameters exposed.

## 1 Introduction

Since 2006, the MaxSAT Evaluation (MSE) [5] has been held annually with the primary objective of advancing MaxSAT technology and assessing its current state-of-the-art. The evaluation consists of multiple solvers being tested on various benchmarks across different evaluation tracks. This event has undeniably spurred the MaxSAT community to create more cutting-edge solvers and enhance their competitiveness.

It is not surprising that solver performance depends on several factors, including the *power* of the algorithm implemented by the solver, proper configuration of solver parameters to unleash its full potential, and implementation issues. Therefore, we must interpret the MaxSAT Evaluation ranking results carefully and derive conclusions according to the goal of our analysis. For example, a similar or *weaker* algorithm could outperform other approaches thanks to better implementation of data structures or a previous tuning process of its input parameters.

From an industrial point of view, we mainly care about obtaining an effective solving approach that is ready for deployment for a particular problem subject to available resources (computing power, environment restrictions, licenses available, etc). From a research point of view, we are more interested in identifying the potential of new solving approaches that lead to further promising research avenues.



© Josep Alòs, Carlos Ansótegui, Josep M. Salvia, and Eduard Torres;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 7; pp. 7:1–7:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our aim is to satisfy both industrial and research perspectives by identifying the best possible solving approach that can be achieved from a single solver while adhering to certain restrictions. In particular, we treat the solver as a black box, meaning that we cannot access its source code, nor do we have any domain knowledge of the problem to be solved, meaning that we cannot utilize any specific structure feature.

Despite these constraints, our approach enables us to unleash the hidden potential of the solver and avoid incorrect rankings of better algorithms that have not been appropriately configured or restarted. Additionally, our study emphasizes the importance of being cautious when interpreting rankings based on the MaxSAT Evaluation, as we mentioned previously.

In this paper, we first show how to effectively configure MaxSAT solvers using Automatic Configuration (AC) tools (tuners), specifically GGA [4] and SMAC [14]. Then, we show that we can take advantage of not only the best configuration returned by the tuners but also a selection of the configurations seen by the tuner during the AC process. With these configurations, we can then build a simple portfolio that runs in parallel these configurations if enough computational resources are available.

We also demonstrate how to create a sequential portfolio that schedules the execution of different parametrizations of a single MaxSAT solver on a given number of cores within a specified timeout. This approach can be thought of as a restarting strategy, where a different configuration of the solver parameters is selected at each restart.

It is worth mentioning that all these approaches are agnostic of the structure of the instances. Otherwise, we should explore extending other approaches available in the literature such as ISAC++ [13].

Finally, we integrate all these building processes in the OptiLog framework [1]. With the new APIs, the user can provide an input MaxSAT solver and its parameters through the *BlackBox Module*, and OptiLog automatically generates a new solving approach for a given number of cores.

We conducted an extensive experimental investigation on the Weighted Incomplete track of the MaxSAT Evaluation 2022, with a particular focus on the highly configurable MaxSAT solver Loandra [6]. In this track, Loandra ranked sixth when restricted to a timeout of 60 seconds. Our approach involves the construction of parallel and sequential portfolios based solely on Loandra, which significantly improves its performance.

## 2 Preliminaries

MaxSAT is the optimization variant of the SAT decision problem. While for SAT the goal is to find an assignment to the Boolean variables (solution) that satisfies all the clauses in the input CNF formula, in MaxSAT we look for a solution that satisfies the maximum possible number of clauses. Since some of these clauses can be falsified we refer to them as *soft* clauses. Within the MaxSAT community, it is typical to reformulate the problem from a minimization perspective aiming to find a solution that falsifies the minimum possible number of soft clauses.

There are several variants of the MaxSAT problem. We can add weights to the soft clauses that represent the cost of falsifying the clause. In this case, we want to look for a solution that minimizes the aggregated cost of the *Weighted* soft clauses. Additionally, we can have *hard* clauses, i.e., clauses that cannot be falsified by the solution.

MaxSAT solvers have experimented a great success in the last decade. Among these solvers, we find complete (or exact) solvers and incomplete solvers. Complete solvers provide optimal solutions while incomplete solvers report solutions as good as possible, but are not

required to guarantee their optimality. These solvers can either refine a lower bound (lb) on the cost of the optimal solutions or an upper bound (ub), or both. In particular, incomplete solvers iteratively report (whenever possible) a better (smaller) upper bound on the optimal solution.

### 3 The MaxSAT Evaluation

The MaxSAT Evaluation 2022 was structured into three tracks: main track complete (unweighted and weighted variants), main track incomplete (unweighted and weighted variants), and the special incremental MaxSAT track. In this paper, we focus on the incomplete track for weighted MaxSAT instances with a timeout of 60 seconds.

The term *incomplete* refers to the type of MaxSAT solvers which are not required to be exact, i.e., they do not need to certify the optimum. Their goal is to report the best possible solution within a given timeout. The term *weighted* refers to the variant of MaxSAT instances. The weighted MaxSAT variant allows integer weights for the soft clauses plus the hard clauses.

We consider the timeout of 60 seconds useful for our study since it is a realistic scenario of industrial applications where we require a suboptimal solution in a short time window and because our automatic configuration process, given the computational resources we have available, can be restricted to two days (see Section 5).

The MaxSAT Evaluation 2022 incomplete (weighted) track involved 197 MaxSAT instances and 10 incomplete solvers: DT-HyWalk [18], noSAT-MaxSAT [15], NuWLS-c [7], Exact [8, 11], Loandra [6], Open-WBO-inc (two variants) [12], and TT-Open-WBO-Inc (three variants) [16].

Each solver  $s$  was ranked according to the scoring function  $score(s)$  shown in Equation 1.

$$score(s) = \frac{\sum_{i=1}^{i=n} score(s, i)}{n} \quad (1)$$

Given a set of  $n$  instances, the  $score(s)$  of a MaxSAT solver  $s$  is the average of the scores for each instance, computed by  $score(s, i)$  in Equation 2.

$$score(s, i) = \frac{1 + \text{best-known ub for instance } i}{1 + \text{ub for } i \text{ found by } s} \quad (2)$$

The  $score(s, i)$  function computes the ratio between the *best-known upper bound* of an instance  $i$  and the bound reported by the solver  $s$  on the same instance. Assuming that  $(\text{best-known ub}) \leq \text{ub}$  (which is the case for the MaxSAT Evaluation), the computed value ranges between 0 and 1, where higher values correspond to better upper bounds.

The competition has some specific rules about what is and is not allowed in the implementation of the solvers. In particular, the solvers are not allowed to employ triggers to modify their behavior, which is deemed to be specific to particular instances. However, solvers can concatenate the usage of different solving techniques.

In the most recent competition, the MaxSAT solvers used a variety of strategies and solvers. Some of these solvers are outlined below, along with the various approaches they employ in order to find improved solutions.

1. **NuWLS-c**: This solver adopts two solvers, the NuWLS solver, which is an improvement of SATLike, and the integration of TT-Open-WBO-Inc.
2. **TT-Open-WBO-Inc**: This solver uses four different strategies, including SATLike for preprocessing, a modified version of Mrs. Beaver for unweighted instances, BMO-clustering for weighted instances, and Polosat, a SAT-based local search method. This MaxSAT solver has three different distributions:

## 7:4 Exploiting Configurations of MaxSAT Solvers

- (g) Which incorporates the Glucose 4.1 SAT Solver.
  - (i) Which incorporates the new Intel SAT Solver.
  - (is) Which incorporates the new Intel SAT Solver and is tuned for short invocations.
3. **DT-HyWalk**: This solver employs three distinct strategies, including a direct call to a SatSolver, the SATLike solver for local search, and the use of another MaxSAT solver, TT-Open-WBO-Inc.
  4. **Loandra**: This solver utilizes two core algorithms, namely a Core-Based algorithm and a Linear algorithm.

Table 1, column “MSE”, shows the results of the MaxSAT Evaluation 2022 for the top six solvers at the incomplete weighted track with 60 seconds timeout. As we can see, the MaxSAT solver Loandra was not competitive within this category. In this paper, we propose an approach that is agnostic of the structure of instances and only allows the usage of alternative configurations of the same input MaxSAT solver. We experimentally show the goodness of our approach on the MaxSAT solver Loandra.

### 3.1 Reproducing the MaxSAT Evaluation for the Incomplete track

All of our executions of the MaxSAT solvers are run on a computation cluster composed of nodes with two AMD 7402 processors (each with 24 cores at 2.8 GHz) and 21 GB of RAM per core, managed by Sun Grid Engine (SGE). All the experiments are managed using the *Running Module* of the OptiLog framework.

Each execution is given 60s of CPU time and 32 GB of memory. As the memory requirements exceed the memory per core available, two slots are reserved and an affinity mask is set by SGE to restrict the execution to only one of the two cores. In contrast to the MaxSAT evaluation, each solver was evaluated with 50 different random seeds and we report results on the average score, and in some of the experiments, we also show the minimum and maximum scores, and the standard deviation.

In the course of developing our experiments, we detected two problems with some executions of the solvers: 1) some executions report a bound that does not correspond to the real cost of the solution reported, and 2) some executions report a solution that does not satisfy the *hard* clauses.

To address these issues we conduct a validation step executed after the solver exhausts the 60s of CPU time. In particular:

For 1), we trust the cost we compute from the solution reported, ignoring the bound reported by the solver.

For 2), we consider the solver was not able to find any solution at all.

This validation step is also conducted during the automatic configuration process when we evaluate a particular configuration of the solver on a given instance (see Section 5).

The score for each solver is computed using the MaxSAT evaluation rules. In particular, it is important to define which is the set of best-known upper bounds that we use to compute the score. Table 1 shows in column “MSE 2022”, the scores reported in the MaxSAT Evaluation 2022.

The rest of the columns present the results of the experimentation we conducted (in our cluster) using different sets of *best-known upper bounds*. “ $VBS_b$ ” uses the upper bounds found by the Virtual Best Solver of the solvers we executed, “ $MSE_b$ ” uses the set of best-known upper bounds provided by the MaxSAT Evaluation, and “ $LRUNS_b$ ” (Long Runs) uses a set of new *best-known upper bounds* we computed by running Loandra and NuWLS-c (both with the default parameters) with a timeout of 12 hours. We recall that the score presented

is the average score on 50 seeds in contrast to the MSE results where only 1 seed is used. As we can observe in Table 1, the relative ranking of the solvers is preserved although we can observe variations in the scores reported. We will use in the rest of the paper the best bounds from  $MSE_b + VBS_b + LRUNS_b$ .

■ **Table 1** Results of the MaxSAT Evaluation 2022 on the incomplete weighted track (60 seconds timeout) and reproduction of the Evaluation in our system with different sets of Best-Known Upper Bounds.

	MSE 2022	MSE 2022 on our system		
Best-known UBs	$MSE_b$	$VBS_b$	$VBS_b + MSE_b$	$VBS_b + MSE_b + LRUNS_b$
Solvers				
NuWLS-c	0.759	0.7831	0.7590	0.7524
DT-Hywalk	0.732	0.7625	0.7414	0.7351
TT-Open-WBO-inc (g)	0.728	0.7412	0.7221	0.7164
TT-Open-WBO-inc (is)	0.726	0.7354	0.7201	0.7141
TT-Open-WBO-inc (i)	0.720	0.7354	0.7178	0.7118
Loandra	0.693	0.7107	0.7003	0.6953

## 4 Automatic Configurators (AC)

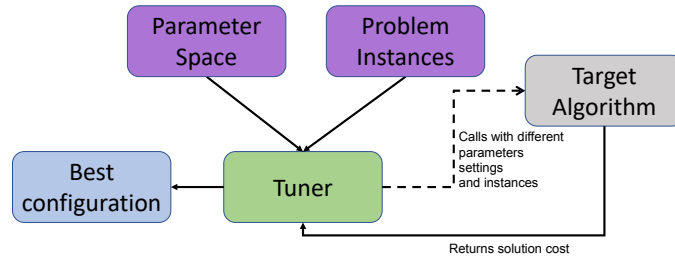
In this section, we review the Automatic Configuration Problem and two state-of-the-art automatic configuration algorithms or tuners.

### 4.1 The Automatic Configuration Problem

Given a target algorithm  $A$  with parameters  $\{p_1, \dots, p_n\}$  of domain  $d(p_i)$ . We define the parameter space  $\Theta$  of  $A$  as the subset  $d(p_1) \times \dots \times d(p_n)$  of *valid* parameter combinations. Depending on the parameter,  $d(p_i)$  can be categorical, a discrete domain of fixed values with no predefined order, or numerical, which represent integer or real values. Then, we define the *Automatic Algorithm Configuration* (AAC) problem as the optimization problem that consists of exploring  $\Theta$  to find a configuration  $\theta \in \Theta$  of  $A$ , which given a set of problem instances  $\Pi$ , minimizes a cost metric  $\hat{c} : \Theta \times \Pi \rightarrow \mathbb{R}$ , without exceeding a configuration budget  $B$ .

It is common for  $A$  to be a black box (target algorithm), meaning it accepts some inputs (the parameters) and provides some output (e.g.,  $\hat{c}$ ), but we cannot see its internal functionality. This allows AAC to generalize to any type of algorithm but makes it more challenging for algorithm tuners since they cannot use  $A$  to infer additional information about  $\Theta$ . In practice,  $A$  is implemented as a binary file that outputs its results in a format suitable for its domain but may not be suitable for the AAC tool. Moreover, it may be necessary to limit the resources that  $A$  can use to solve an instance, such as memory or CPU time. The standard way of addressing these issues in AAC tools is for the user to replace  $A$  with a wrapper script that handles these and any other necessary aspects. Figure 1 describes the automatic configuration process where the tuner is a solver for the AAC problem.





■ **Figure 1** Visualization of the Automatic Configuration process.

## 4.2 The GGA Automatic Configurator

The Gender-Based Genetic Automatic Algorithm Configuration (GGA) is a genetic algorithm that was introduced in [4] to search for high-quality configurations. It was one of the pioneering algorithms that supported continuous parameters and introduced the novel concept of *gender* to apply diverse selection pressures to the population’s individuals.

### ■ Algorithm 1 GGA.

**Input:** Target Algorithm  $A$ , Parameter Space  $\Theta$ , Instances  $\Pi$ , Performance Metric  $\hat{c}$ , # MiniTournaments  $N$ , Configuration Budget  $B$

```

1: function GGA( $A, \Theta, \Pi, \hat{c}, N, B$ )
2:    $pop \leftarrow \text{initPopulation}(\Theta)$ 
3:    $j = 0$ 
4:   while  $B$  not exhausted and threshold not achieved do
5:      $j = j + 1$ 
6:      $\Pi_j \leftarrow \text{selectInstances}(\Pi, j)$ 
7:      $\langle w_1, \dots, w_N \rangle \leftarrow \text{runMiniTournaments}(A, pop.comp, \Pi_j, \hat{c}, pop.comp/N)$ 
8:      $offspring \leftarrow \text{applyCrossoverAndMutate}(pop.noncomp, \langle w_1, \dots, w_N \rangle, \Theta)$ 
9:      $pop \leftarrow \text{agingAndDeath}(w_1, pop) \cup offspring$ 
return  $w_1$ 
  
```

Algorithm 1 shows the pseudocode of the GGA algorithm, which takes as input the target algorithm  $A$ , its parameter space  $\Theta$ , a set of training instances  $\Pi$ , a performance metric  $\hat{c}$  to optimize (e.g., time, accuracy, quality within a fixed timeout, etc), the number  $N$  of GGA mini-tournaments (which will be explained shortly), and a configuration time budget  $B$ .

GGA starts by initializing a *population* ( $pop$ ) of configurations (named *genomes*) as a subset of  $\Theta$  in line 2. This population is partitioned into a *competitive* group ( $pop.comp$ , which is directly evaluated on the target algorithm) and *non-competitive* group ( $pop.noncomp$ , which simply acts as a source of diversity).

The algorithm proceeds in a main loop that finishes when GGA reaches the configuration budget  $B$  or a threshold on the performance (line 4). At each iteration (which we call *generation*), GGA selects a subset of the instances  $\Pi_j$  to evaluate the genomes in line 6<sup>1</sup>. Then, in line 7, GGA evaluates the competitive genomes of the population over the selected instances  $\Pi_j$  using a parallel racing scheme called *mini-tournament*. This procedure returns

<sup>1</sup> There are different policies that can be applied to select the instances at each generation, see [4].

a set of  $N$  winners,  $\langle w_1, \dots, w_N \rangle$ , which will be the only competitive genomes that will generate new offspring in this generation (line 8). Finally, GGA applies an ageing policy in line 9 that is used to prevent population growth. The only exception is the overall best competitive genome ( $w_1$ ), which survives as long as it performs better than the other mini-tournament winners. At the end of the main loop, GGA returns the best competitive genome  $w_1$  of the last generation.

For more details on the GGA algorithm, we refer the reader to [4].

### 4.3 The SMAC Automatic Configurator

Sequential Model-Based Algorithm Configuration (SMAC) is an automatic configuration algorithm based on Bayesian optimization [10, 14]. In Bayesian optimization, we use a few evaluations of the target algorithm to train a *surrogate model* that predicts the performance of the algorithm for a given configuration. This fast-to-evaluate surrogate model is used to search for promising new configurations that will be executed on the training instances.

#### ■ Algorithm 2 SMAC.

---

**Input:** Target Algorithm  $A$ , Parameter Space  $\Theta$ , Instances  $\Pi$ , Performance Metric  $\hat{c}$ , Configuration Budget  $B$

- 1: **function** SMAC( $A, \Theta, \Pi, \hat{c}, B$ )
- 2:    $[R, \theta_{inc}] \leftarrow \text{initialize}(\Theta, \Pi)$
- 3:   **while**  $B$  not exhausted **do**
- 4:      $[M, t_{fit}] \leftarrow \text{fitModel}(R)$
- 5:      $[\vec{\Theta}_{new}, t_{select}] \leftarrow \text{selectConfigurations}(M, \theta_{inc}, \Theta)$
- 6:      $[R, \theta_{inc}] \leftarrow \text{intensify}(A, \vec{\Theta}_{new}, \theta_{inc}, R, \Pi, \hat{c})$

**return**  $\theta_{inc}$

---

Algorithm 2 shows the pseudocode of SMAC. This algorithm receives as input the target algorithm  $A$ , its parameter space  $\Theta$ , a set of training instances  $\Pi$ , a performance metric  $\hat{c}$  to optimize and a configuration time budget  $B$ . First, SMAC initializes a *best candidate* configuration  $\theta_{inc}$  and the history of conducted evaluations of different (*configuration, instance*) pairs  $R$  (which might be empty) in line 2.

As in GGA (see Section 4.2), SMAC has a main loop defined in line 3 that proceeds until the configuration budget  $B$  is reached. At each iteration, it fits a surrogate model  $M$  using the information in  $R$  in line 4. Then, it uses  $M$  to select a new set of promising candidate configurations  $\vec{\Theta}_{new}$  in line 5. Finally, it evaluates  $\vec{\Theta}_{new}$  and  $\theta_{inc}$  on instances from  $\Pi$  to determine the next best candidate  $\theta_{inc}$ , according to  $\hat{c}$  in line 6. Similar to the GGA algorithm, SMAC returns the best candidate configuration  $\theta_{inc}$ .

### 4.4 Support for Tuning into the OptiLog framework

In this section, we present an excerpt of the code that uses the OptiLog framework to generate the configuration tuner environment (from now on Tuning Scenario) of the solver Loandra for GGA and SMAC tuners.

```

1 # example_ac.py
2
3 from optilog.blackbox import *
4 from optilog.running import ParsingInfo
5 from optilog.tuning import *
6

```

## 7:8 Exploiting Configurations of MaxSAT Solvers

```
7 class LoandraBB(SystemBlackBox):
8     config = {
9         "weight-strategy": Int(0, 2, default=2),
10        "preprocess": Bool(default=True),
11        (...)
12    }
13    (...)
```

■ **Listing 1** Sample code to wrap the solver Loandra into an OptiLog BlackBox.

Listing 1 defines a custom BlackBox class named LoandraBB that inherits from SystemBlackBox. This class represents the binary that we want to optimize. The `config` dictionary defines the parameters of this binary that can be tuned by the optimization algorithm. We show the parameters “weight-strategy” and “preprocess”, with their respective types and default values. In particular, we need 18 lines of code to wrap Loandra, with 40 additional lines defining the parameters.

```
1 # scenario_gga.py
2
3 from optilog.tuning.configurators import *
4 from example_ac import LoandraBB
5
6 if __name__ == "__main__":
7     configurator = GGACConfigurator(
8         LoandraBB(),
9         input_data="/path/to/instances/*",
10        ...
11    )
12    configurator.generate_scenario("./scenario")
```

■ **Listing 2** Sample code to create a Tuning Scenario for the solver defined in Listing 1.

Listing 2 is a definition of a Tuning Scenario for the solver Loandra. It imports the custom LoandraBB class defined in Listing 1, and sets up a tuner (in this case GGA) to optimize the parameters of LoandraBB. The `input_data` parameter specifies the path to the instances used during the optimization process. Lastly, the `generate_scenario` method is called with the desired output path for the scenario that is being created.

Similar code to Listing 2 could be used to define a Tuning Scenario to be used with the SMAC AC tool, as OptiLog supports both GGA and SMAC.

## 5 Configuring MaxSAT Solvers

Although the AC tools (tuners) presented in the previous section have also parameters that impact the effectiveness of the configuration process, *tuning the tuner* is out of reach in this paper and we focus on providing a good cost function to be used during the tuning process.

Ideally, we would use the  $score(s, i)$  function from the MaxSAT Evaluation (Equation 2). Notice though that in the MaxSAT Evaluation we are trying to maximize this scoring function, whereas tuners minimize a cost (see Section 4). Therefore, we have to convert the *score* function to a *cost* function. Additionally, it is not guaranteed that the bounds found are equal or worse than the previously *best-known upper bounds* (see Section 3), and we cannot update the *best-known upper bounds* sets during the tuning process (otherwise previous results computed in the same tuning process would not be comparable).

We define the  $cost_{ac}(s, i)$  function, shown in Equation 3, as follows. First, we split the function in two cases: 1) the reported bound is worse (or equal) than the previous *best-known upper bound*, and 2) the bound reported is better.

For 1), we compute  $1 - score(s, i)$  to obtain a value in the range  $[0, 1)$ , where better bounds are closer to 0.

For 2), notice that we are breaking the assumption  $(best-known\ ub) \leq ub$ , which may lead to unbounded values that tend to  $\infty$ . To restrict the values to the range  $(-1, 0)$ , we use the inverse of the  $score(s, i)$  function, and then subtract 1.

The  $cost_{ac}(s, i)$  function returns values between  $(-1, 1)$ , where better bounds correspond to values closer to  $-1$ .

$$cost_{ac}(s, i) = \begin{cases} 1 - score(s, i), & \text{if } ub \text{ for } i \text{ found by } s \geq \text{best-known } ub \text{ for } i \\ \frac{1}{score(s, i)} - 1, & \text{otherwise} \end{cases} \quad (3)$$

As stated in Section 3.1, some executions might report a bound that does not match the reported solution. Thus, we integrate a validation step (see Figure 2) that certifies the real cost of the solution returned by the solver and reports it to the tuner.

Regarding the tuning environment, all experiments are conducted on the same computation cluster. Each tuning process is given a wall-time tuning budget of 48 hours, a memory limit of 32G per worker, and is allowed to use up to 50 parallel workers unless otherwise specified. Each configuration instance is given a CPU time limit of 60 seconds for the solver, and then a validation step is executed. As training instances for the tuning process, we will use the 197 instances from the MaxSAT Evaluation 2021 (incomplete weighted track, 60 seconds) and we will test the best configuration returned by GGA and SMAC (see Section 4) on the 151 instances from the MaxSAT Evaluation 2022.

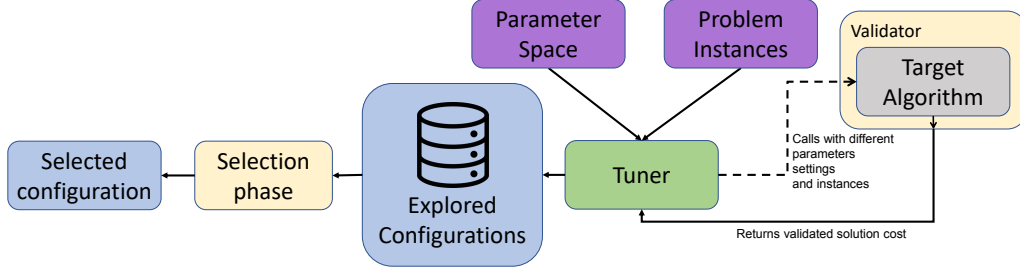
GGA allows for the selection of how many instances are used in each generation. Incrementally increasing the training set across generations till including the whole set of available instances is often recommended, as it facilitates discarding bad configurations with less effort, therefore more generations can be reached within the tuning budget. However, as discussed in Section 6, prioritizing the evaluation of more configurations on the whole training set within the same tuning budget may be preferable over having more generations. GGA also can preserve a set of *elite* configurations that are run at every generation. We define as an *elite* the default configuration of Loandra. Finally, we use the PyDGGA [2] (version 1.7.0) distribution of GGA which has support for distributed execution. The following non-default parameters were used for GGA: cost tolerance set to 0, population set to 100, generations set to 300, and minimum generations set to 50.

Regarding SMAC, although it can be executed in parallel, it does not report an overall winner in contrast to GGA. Instead, it reports as many winners as computation cores were used since it basically runs several sequential SMACs in parallel. Thus, after SMAC completes, we have to take all the winners from each SMAC sequential execution, which may not have been evaluated on all training instances, and perform the missing evaluations. Then, the winner with the best performance on the training set is selected to be the overall winner. We use the SMAC3 [14] (version 1.4.0) implementation of the algorithm.

As we have described earlier, the cost function used during the tuning process is not *strictly* the minimization version of the score we maximize according to the MSE 2022 (see Section 3). Therefore, one may argue that it would be better to return a winner for the training set with respect to the score function computed by the MSE. This is easy to do if the tuner provides the logs of each evaluation so, in the case of incomplete MaxSAT solvers, we can retrieve the best bound found by the solver on a given instance.

## 7:10 Exploiting Configurations of MaxSAT Solvers

Therefore, we add a *selection phase* (see Figure 2) after the tuning phase that recomputes the scores (according to the MSE) of the configurations traversed by the tool during the tuning process.



■ **Figure 2** Visualization of the Automatic Configuration process extended with the validator and a selection phase over the explored configurations.

In particular, for SMAC, we compute the MSE score of the 50 winners reported by the tuner on the training set and select the one with the highest score. Even though we have access to the logs of the evaluations of SMAC and could use those scores to select the winning configuration, we need to make sure that all the configurations are evaluated with all the instances.

For GGA, we order the configurations first by their ranking in a generation (according to the cost function in Equation 5), and within the same rank, we order by the most recent generation. Then, we select the first 50 distinct configurations. We look into their logs, recompute their MSE score according to Section 3, and report the winner<sup>2</sup>.

■ **Table 2** Comparison using GGA and SMAC to tune the Loandra solver (using  $VBS_b + MSE_b + LRUNS_b$  bounds).

	Mean	Median	Min	Max	Std
NuWLS-c	0.7524	0.7522	0.7484	0.7560	0.0017
Loa (GGA, all-i)	0.7393	0.7391	0.7313	0.7475	0.0037
Loa (GGA, incremental)	0.7353	0.7354	0.7275	0.7433	0.0038
DT-Hywalk	0.7351	0.7355	0.7288	0.7415	0.0030
Loa (SMAC)	0.7237	0.7234	0.7149	0.7355	0.0048
TT-Open-WBO-inc (g)	0.7164	0.7165	0.7128	0.7194	0.0015
TT-Open-WBO-inc (i)	0.7141	0.7142	0.7093	0.7188	0.0020
TT-Open-WBO-inc (is)	0.7118	0.7117	0.7098	0.7145	0.0008
Loandra	0.6953	0.6957	0.6872	0.7036	0.0037

Table 2 shows the result of the best configurations provided by GGA using all the instances from the first generation (“Loa (GGA, all-i)”), or adding them incrementally at each step (“Loa (GGA, incremental)”) and the best configuration provided by SMAC (see

<sup>2</sup> In our experiments, the winner reported by GGA was the same configuration as the best one found in the *selection phase*.

“*Loa (SMAC)*”) after the additional selections process described in the paragraph above. For the incremental approach of GGA, we use 20% of instances at the first generation and instruct GGA to use all the instances on generation 25. Those values were selected based on preliminary experiments taking into account the number of generations that GGA can do in the given time. It is clear by the results that the usage of all the instances from the beginning benefits GGA, allowing it to lift *Loandra* from the sixth position to the second one. In the next section, we will focus on the variant of GGA “*Loa (GGA, all-i)*”.

We also ran the variants “*Loa (GGA, all-i)*” and “*Loandra*” on another set of benchmarks, the MSE 2020 instances. The default parameters variant achieves a score of 0.755, and the tuned version a score of 0.777.

## 6 Exploiting Configurations Discarded by the Tuner

As it has been shown in the literature [9, 17], from the most pragmatic point of view, we can obtain an efficient parallel approach by just running the same non-deterministic solver with different seeds in parallel, or we can also run in parallel different configurations of the same solver.

In case resources are limited, we can also schedule the execution of different configurations of the same solver. In this section, we concrete and study these different approaches. We use OptiLog [1] to generate all the portfolios, as we explain in Section 6.3.

### 6.1 Parallel Portfolios of seeds and configurations

As we have already explained, tuners report the best configuration they have found. However, many other *potentially* good configurations are also explored and discarded during the automatic configuration process with respect to their performance on the particular training set. These configurations may exhibit good performance in different kinds of instances. As observed in [3] on SAT benchmarks, superior performance can be achieved by combining these complementary configurations.

The first approach we explored is the parallel execution of  $N$  different random seeds over a given MaxSAT solver. This approach can be applied to both the *default* MaxSAT solver and the best configuration obtained in the tuner.

Another approach is to extract  $N$  configurations of a MaxSAT solver from the ones traversed by the tuner and execute them in parallel. There are many strategies that we could follow to extract these configurations from the tuner. In particular, we use the set of configurations considered during the *selection phase* (after the *tuning phase*) process as explained in Section 5. Notice that we do not analyze any structure of the instances and we only incorporate configurations of the same solver.

Table 3 shows the results of the parallel portfolios that we explained. We tested parallel portfolios with 25, 30, 35, 40, 45, and 50 parallel executions. Each row shows the results of a parallel portfolio (rows marked with *(Seeds)* refer to a parallel portfolio of seeds, whereas the row marked with *(Configs)* refer to a parallel portfolio of configurations). We show the score as computed in the MaxSAT evaluation using the  $VBS_b + MSE_b + LRUNS_b$  upper bounds and the rank of each portfolio with respect to the others.

As we can observe, the portfolio over different seeds for the default *Loandra* (“*(Seeds) Loandra*” in Table 3) is not competitive while the portfolio of different seeds for the best configuration of *Loandra* computed by GGA (column “*(Seeds) Loa (GGA, all-i)*”) already outperforms NuWLS-c. Additionally, a portfolio of the best configurations provided by the selection phase (column “*(Configs) Loa (GGA, all-i)*”) systematically outperforms the rest of the approaches. These observations hold almost for any number of parallel executions.

■ **Table 3** Score and rank (#) for each parallel portfolio, given  $N$  parallel processes (using  $VBS_b + MSE_b + LRUNS_b$  bounds).

$N$	25		30		35		40		45		50	
	#	score	#	score	#	score	#	score	#	score	#	score
(Configs) Loa (GGA, all-i)	1	0.813592	1	0.818663	1	0.823986	1	0.825208	1	0.826448	1	0.827105
(Seeds) Loa (GGA, all-i)	2	0.806889	2	0.808078	2	0.809922	2	0.812589	2	0.813007	2	0.815263
(Seeds) NuWLS-c	3	0.768409	3	0.769302	3	0.769785	3	0.770293	3	0.771219	3	0.771263
(Seeds) DT-Hywalk	4	0.759271	4	0.764688	4	0.765736	4	0.765764	4	0.765862	4	0.766397
(Seeds) TT-Open-WBO-inc (g)	5	0.728901	5	0.728903	5	0.729766	5	0.729902	5	0.730165	5	0.730235
(Seeds) TT-Open-WBO-inc (i)	6	0.725971	6	0.726171	6	0.726253	6	0.726315	6	0.726510	6	0.727468
(Seeds) Loandra	8	0.717788	8	0.722663	7	0.723954	7	0.723996	7	0.724521	7	0.724648
(Seeds) TT-Open-WBO-inc (is)	7	0.722672	7	0.722704	8	0.722918	8	0.723174	8	0.723324	8	0.723489

## 6.2 Sequential Portfolios of configurations

In some settings, we will not have enough resources to run a parallel portfolio as described in Section 6.1. Potentially, we can have just one computation core available. In this case, we can schedule the sequential execution of different configurations of Loandra within the given timeout.

Let us describe how we construct this *sequential* portfolio. We assume we have a sequence of solvers (or configurations of a solver) ( $S$ ) that iteratively report better solutions, a time budget ( $TO$ ), and a maximum time budget a solver can exhaust between two consecutive reported solutions ( $MTBS$ ). The solvers are executed according to their order in the sequence until the time consumed globally by all the solvers exceeds  $TO$ .

Each solver is run as follows: first, we wait for the first solution reported by the solver. Once this first solution is reported, we start a timer of  $MTBS$  seconds. If the solver reports a new solution before this timer expires, we reset the timer and wait for a new solution. This is repeated until the solver is unable to report a new solution before the timer is consumed. At that point in time, the solver is stopped and the next one in the ordered list of solvers is executed. Note that, at any point in this process, a solver can also be stopped if the global time budget of  $TO$  seconds gets exhausted. A special case is the last solver of the sequence, which is allowed to run until the time budget expires (i.e. it is not stopped even if it took more than  $MTBS$  seconds to find a new solution). Obviously, we keep track of the best overall solution seen so far.

To identify which sequence of solvers  $S$  and  $MTBS$  value the portfolio should use, we carry out a simulation of sequential portfolios with the configurations provided by the *selection phase* (see Section 5) and their respective logs on the training instances computed during the *tuning phase*. In particular, we explore all sequences of up to size 3 and  $MTBS$  values of  $\{2, 3, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$  seconds. Once we identify the best *virtual* sequential portfolio for the training instances, we simulate again the execution of this *virtual* sequential portfolio on the test set. In Table 4 we present the results of this simulation.

To implement this *virtual* sequential portfolio we would need to take into account an additional thread that keeps track of the evolution of the solvers in the sequence, which may decrease the overall performance. Therefore, we see this *virtual* sequential portfolio as a restarting policy that MaxSAT developers could integrate into their solvers, with the added benefit that they may be able to reuse information computed by each solver in the sequence.

Table 4 shows the results of the *virtual* sequential portfolios (rows prefixed with “Virtual portfolio”), compared to the results that obtained the solvers from the competition with the default parameters, and with the best approach obtained using a tuner (“Loa (GGA, all-i)”). As in the MSE we run each solver with the same seed, except for NuWLS-c for which we also

■ **Table 4** Score of the *virtual* sequential portfolio compared with the single-execution approach (using  $VBS_b + MSE_b + LRUNS_b$  bounds).

	Score
Virtual sequential portfolio (N=2) - Loa (gga, all-i)	0.7642
Virtual sequential portfolio (N=3) - Loa (gga, all-i)	0.7642
NuWLS-c (max score on 50 seeds)	0.7560
NuWLS-c	0.7554
Loa (gga, all-i)	0.7513
DT-Hywalk	0.7432
TT-Open-WBO-inc (g)	0.7214
TT-Open-WBO-inc (i)	0.7180
TT-Open-WBO-inc (is)	0.7180
Loandra	0.6965

report on the best score value from 50 seeds. The  $N$  value shown in the *virtual* sequential portfolios rows indicates the length of the solvers' sequence. The portfolios are built on top of the configurations obtained after the selection phase with ("*Loa (GGA, all-i)*").

We notice that *virtual* sequential portfolios do perform better than NuWLS-c, and a selection of two configurations suffices to that end. Interestingly, if we build the *virtual* sequential portfolio on the test instances from the MSE 2022, then we get a better portfolio using three configurations that achieves a score of 0.7689, however, we cannot predict this portfolio based on the analysis we perform on the training instances from the MSE 2021.

Additionally, we conducted experiments to analyze the potential of combining solvers. In particular, we used the solver TT-Open-WBO<sup>3</sup>. We tuned this solver, and generated a *virtual* sequential portfolio combining the best configurations of TT-Open-WBO and Loandra. The *virtual* sequential portfolio (based on the analysis of the performance on the MSE 2021) obtained a score of 0.779 on the MSE 2022, which is the best score obtained for single-core evaluations. In comparison, the individual performance of Loandra and TT-Open-WBO after tuning is 0.747 and 0.751 respectively.

### 6.3 OptiLog Portfolio Generator

To facilitate the creation of the parallel and *virtual* sequential portfolios, we added support to compute them using the OptiLog framework.

```

1 from optilog.portfolio import get_parallel_portfolio
2
3 get_parallel_portfolio(
4     gga_scenario='./tuning-scenario',
5     n_solvers=10,
6     save_to='./parallel-portfolio'
7 )

```

■ **Listing 3** Computing a parallel portfolio with OptiLog.

<sup>3</sup> A version provided by the author of the solver with the parameters exposed.



## 7:14 Exploiting Configurations of MaxSAT Solvers

Listing 3 shows how we can generate and save a parallel portfolio with OptiLog. This portfolio is built by selecting  $N$  configurations as explained in Section 5, thus requiring a Tuning Scenario (generated with OptiLog as seen in Section 6.3). The function `get_parallel_portfolio` receives as parameters the Tuning Scenario that contains the results of the tuning process (`gga_scenario`), the number of solvers that will compose the parallel portfolio (`n_solvers`), and the directory where the scripts to launch each individual solver that composes the portfolio will be saved (`save_to`).

```
1 from optilog.portfolio import get_sequential_portfolio
2
3 get_sequential_portfolio(
4     path_scenario="./running-scenario",
5     n_solvers=2,
6     solution_regex=r"^o\s(\d+)",
7     save_to="./sequential-portfolio",
8     score_fn=maxsat_score_fn,
9     max_time_between_solutions=[5, 15, 25, 35]
10 )
```

■ **Listing 4** Computing a sequential portfolio with OptiLog.

Listing 4 shows how we are generating a sequential portfolio with the results of a Running Scenario. Note that to generate the *virtual* sequential portfolio we require the full trace of the solvers (in particular for the incomplete MaxSAT case, we need the evolution of the best bound over time), so we cannot build it from a Tuning Scenario directly. The parameters `gga_scenario`, `save_to`, and `n_solvers` mean the same as in the function to compute a parallel portfolio. Additionally, we have to specify the following parameters: `score_fn` is used to transform the lines matched by `solution_regex` to a score that the portfolio will try to maximize (in this example the score function is `score(s)` defined in Section 3), and `max_time_between_solutions` contains the possible values that the portfolio can choose from when selecting the parameter *MTBS*.

## 7 Conclusions

Given a target solver, we have presented an approach to easily generate a potentially much better solving approach. To this end, we exploit a set of alternative configurations of the same target solver coming from the residues of a tuning process. It is important to notice that we do not exploit any structure feature of the input problem or instance since in some domains these features are not easy to compute. In particular, we have shown how from a MaxSAT solver with a low ranking in one of the tracks of the MSE 2022 we can obtain a more competitive approach.

Our sequential portfolio generation approach can be seen as a first attempt to come up with effective restarting policies for MaxSAT solvers, something that has not been studied in depth in the literature.

Finally, the approach described has been integrated into the OptiLog framework avoiding the tedious process of setting up tuning environments and generating portfolios. Moreover, the API is general enough to be applied not only to MaxSAT solvers but to other solving approaches.




## References

- 1 Josep Alòs, Carlos Ansótegui, Josep M. Salvia, and Eduard Torres. OptiLog V2: Model, Solve, Tune and Run. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi: 10.4230/LIPIcs.SAT.2022.25.
- 2 Carlos Ansótegui, Josep Pon, and Meinolf Sellmann. Boosting evolutionary algorithm configuration. *Annals of Mathematics and Artificial Intelligence*, 2021. doi:10.1007/s10472-020-09726-y.
- 3 Carlos Ansótegui, Josep Pon, and Meinolf Sellmann. Boosting evolutionary algorithm configuration. *Annals of Mathematics and Artificial Intelligence*, 90(7-9):715–734, 2022. Publisher: Springer.
- 4 Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A Gender-based Genetic Algorithm for the Automatic Configuration of Algorithms. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, CP’09*, pages 142–157. Springer-Verlag, 2009. tex.acmid: 1789011 tex.numpages: 16 tex.year: 2009 event-place: Lisbon, Portugal. URL: <http://dl.acm.org/citation.cfm?id=1788994.1789011>.
- 5 Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, Ruben Martins, and Andreas Niskanen. MaxSAT Evaluation 2022 : Solver and Benchmark Descriptions. *Department of Computer Science Series of Publications B*, B-2022-2, 2022. Accepted: 2022-08-25T10:09:01Z Publisher: Department of Computer Science, University of Helsinki. URL: <https://helda.helsinki.fi/handle/10138/347396>.
- 6 Jeremias Berg, Emir Demirovic, and Peter Stuckey. Core-boosted linear search for incomplete maxsat. *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 16th International Conference, CPAIOR 2019*, 2019.
- 7 Yi Chu, Shaowei Cai, Zhendong Lei, and Xiang He. Nuwls-c: Solver description. *MaxSAT Evaluation 2022*, page 28, 2022.
- 8 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 1291–1299, 2018. URL: <https://www.ijcai.org/Proceedings/2018/180>.
- 9 Youssef Hamadi, Said Jabbour, and Lakhdar Sais. ManySAT: a parallel SAT solver. *JSAT*, 6:245–262, June 2009. doi:10.3233/SAT190070.
- 10 Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration (extended version). *International Conference on Learning and Intelligent Optimization*, 2011.
- 11 Jo Devriendt. Exact Solver Repository, April 2023. URL: <https://gitlab.com/JoD/exact>.
- 12 Saurabh Joshi, Prateek Kumar, Sukrut Rao, and Ruben Martins. Open-wbo-inc: Approximation strategies for incomplete weighted maxsat. *J. Satisf. Boolean Model. Comput.*, 11(1):73–97, 2019. doi:10.3233/SAT190118.
- 13 Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC – Instance-Specific Algorithm Configuration. In *ECAI 2010*, pages 751–756. IOS Press, 2010. doi: 10.3233/978-1-60750-606-5-751.
- 14 Marius Lindauer, Katharina Eggenberger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. In *ArXiv: 2109.09831*, 2021. URL: <https://arxiv.org/abs/2109.09831>.
- 15 Ole Lübke and Sibylle Schupp. nosat-maxsat. In *MaxSAT Evaluation 2022*, pages 29–30. Department of Computer Science, University of Helsinki, 2022.
- 16 Alexander Nadel. Polarity and Variable Selection Heuristics for SAT-Based Anytime MaxSAT: System Description. *Journal on Satisfiability, Boolean Modeling and Computation*, 12(1):17–22, September 2020. doi:10.3233/SAT-200126.

## 7:16 Exploiting Configurations of MaxSAT Solvers

- 17 Olivier Roussel. Description of ppfolio 2012. In *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, 2012.
- 18 Jiongzhi Zheng, Kun He, Zhuo Chen, Jianrong Zhou, and Chu-Min Li. Decision tree based hybrid walking strategies. *MaxSAT Evaluation 2022*, page 24, 2022.

# Symmetries for Cube-And-Conquer in Finite Model Finding

João Araújo   

Universidade Nova de Lisboa, Lisbon, Portugal

Choiwah Chow  

Universidade Aberta, Lisbon, Portugal

Mikoláš Janota   

Czech Technical University in Prague, Czech Republic

---

## Abstract

The cube-and-conquer paradigm enables massive parallelization of SAT solvers, which has proven to be crucial in solving highly combinatorial problems. In this paper, we apply the paradigm in the context of finite model finding, where we show that isomorphic cubes can be discarded since they lead to isomorphic models. However, we are faced with the complication that a well-known technique, the Least Number Heuristic (LNH), already exists in finite model finders to effectively prune (some) isomorphic models from the search. Therefore, it needs to be shown that isomorphic cubes still can be discarded when the LNH is used. The presented ideas are incorporated into the finite model finder Mace4, where we demonstrate significant improvements in model enumeration.

**2012 ACM Subject Classification** Computing methodologies; Theory of computation → Constraint and logic programming

**Keywords and phrases** finite model enumeration, cube-and-conquer, symmetry-breaking, parallel algorithm, least number heuristic

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.8

**Supplementary Material** *Software (Source Code)*: [https://github.com/ChoiwahChow/public/tree/main/CP\\_2023\\_Supplement\\_2.zip](https://github.com/ChoiwahChow/public/tree/main/CP_2023_Supplement_2.zip)

archived at `swh:1:cnt:69e6b145a05d29726512c0605cfb027c17c98dd3`

**Funding** *João Araújo*: Fundação para a Ciência e a Tecnologia, through the projects UIDB/00297-/2020 (CMA), PTDC/MAT-PUR/31174/2017, UIDB/04621/2020 and UIDP/04621/2020.

*Mikoláš Janota*: The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902. This article is part of the *RICAIP* project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306.

## 1 Introduction

An important tool that working algebraists need in their research is libraries of the algebras they are interested in. These libraries allow them to get intuitions, test or refute hypotheses and conjectures, and gain insights into the properties of the algebras (see examples on p. 2891 of [30]). Many libraries of algebraic models of small orders, such as the `smallsemi` package [14] for semigroups and the `loops` package [36] for quasigroups, are available in the GAP [16] system. A lot more such libraries are needed, but they often take an inordinate amount of time and computing resources to generate.

First-order logic (FOL) has been the most popular language to define algebras. There are two major resource-intensive steps in generating non-isomorphic models from FOL [27]. The first step is to generate models according to the laws specified by the FOL formula. This step often generates a huge number of isomorphic models. For example, given the first-order



© João Araújo, Choiwah Chow, and Mikoláš Janota;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 8; pp. 8:1–8:19

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

formula for semigroups, which is  $(x * y) * z = x * (y * z)$ , Mace4 [35] generates 1,021,120,198 models of order 7, out of which only 1,627,672 ( $\approx 0.16\%$ ) [44] are pairwise non-isomorphic. The second step is to eliminate the isomorphic models generated in the first step. In this paper, we propose a novel efficient and scalable parallel algorithm that not only speeds up the first step but also generates fewer isomorphic models. Suppressing the generation of isomorphic models in the first step reduces the workloads of both the first and the second steps. Not only does it make the whole process much faster, but the required computing resources (disk space, etc.) are also reduced.

While modern-day general-purpose computers are predominantly multi-core, harnessing parallelism for combinatorial search is surprisingly difficult. Consequently, there are few parallel algorithms in constraints programming in general, and in finite model enumeration in particular. Indeed, in satisfiability modulo theories (SMT), even negative results are concluded for cube-and-conquer [23]. A recent literature review concludes that “there is little overall guidance that can be given on how best to exploit multi-core computers to speed up constraint solving” [18]. We aim to help close this gap by devising new parallel algorithms for finite model enumeration.

This paper advances finite model enumerators toward the following two objectives:

1. Mathematicians can use the tool to quickly generate all models (up to isomorphism) of the classes of algebras of their interests on their multi-core computers.
2. The tool can also take advantage of massively parallel computing architectures to pre-generate models (up to isomorphism) of the classes of algebras of general interest.

We find inspiration in the well-established cube-and-conquer approach introduced for SAT [20]. In SAT this means splitting the search space by mutually exclusive conjunctions of propositional literals (cubes). In the context of finite model finding, the structure is richer – a decision of the solver corresponds to inserting a point into the graph of one of the considered functions, e.g.,  $f(0, 1) = 3$ . We comment on cube-and-conquer in more detail in Section 6.

We show that a cube can be excluded from the search if it is isomorphic to an existing one. Effectively, this is breaking symmetries in the search space. However, the task is nontrivial because finite model finders already contain a technique, called the least number heuristic (LNH), to exclude some isomorphic models. The LNH<sup>1</sup> enables the solver to consider only certain values from the co-domain for a given decision point. Therefore, we show that isomorphic cubes can be pruned in the presence of the LNH. Like so, we can take advantage of the two powerful and complementary techniques and ultimately suppress the generation of a large number of isomorphic models.

This paper’s contributions are the following:

1. Devise a low runtime overhead parallel algorithm based on the cube-and-conquer approach for finite model enumeration. This scalable algorithm divides finite model enumeration into many independent non-overlapping search jobs to make full use of the available resources.
2. We show that isomorphic cubes can be discarded without losing isomorphic models even in the presence of the well established symmetry breaking technique already present in finite model finders – the least number heuristic (LNH).
3. We extend the model finder Mace4 with the proposed techniques and evaluate it on a large number of problems, where significant speed-up is observed.

---

<sup>1</sup> Despite the technique being called a heuristic, it does not sacrifice the completeness of the solver.

## 2 Preliminaries

Familiarity with the general notions of abstract algebra such as groups, semigroups, and quasigroups is assumed, and so is general knowledge about functions and isomorphisms. A good reference is Chapters 2 and 5 of [9].

In this paper, the domain of the search space is denoted by the set  $D = \{0, \dots, n - 1\}$ , where  $n \geq 2$ . That is, we exclude the trivial case of searching on domains of size 1.

Let  $\pi$  denote an arbitrary permutation on  $D$ ,  $\pi_{id}$  denote the identity permutation, and  $\pi_{(a,b)}$  denote the permutation cycle  $(a, b)$ . For example,  $\pi_{(0,1)}$  is the permutation cycle  $(0, 1)$ .

### 2.1 Finite Model Enumeration

For a signature  $\Sigma$  and a FOL formula  $\mathcal{F}$  on  $\Sigma$ , a traditional finite model finder first expands the FOL formula to its ground representation by its domain elements in  $D$ , then searches for models by backtracking to exhaustively explore the search space [49]. The domain elements in  $D$  are seen as special constants not appearing in the original  $\mathcal{F}$ , c.f. [40].

Following the terminology of [49], a *value assignment (VA) clause* is a term  $f(a_1, \dots, a_k) = v$ , where  $f$  is a  $k$ -ary function symbol in  $\Sigma$  and  $a_j, v \in D$ . We refer to the term  $f(a_1, \dots, a_k)$  as the *cell term* (or simply *cell*) since conceptually the search fills the operation table of  $f$ .

To search for finite models in  $\mathcal{F}$ , the finite model finder employs a *cell selection* strategy to pick cell terms successively, without duplicates, to assign values from  $D$  to form VA clauses. If a newly formed VA clause causes any failure in the axioms in  $\mathcal{F}$ , then a new value will be tried for that cell term. If no value can be assigned to that cell term without failing the axioms in  $\mathcal{F}$ , then the model finder backtracks to the previous cell term to try to assign another value to it. When all cell terms in  $\mathcal{F}$  are assigned values without violating the axioms in  $\mathcal{F}$ , a model, as represented by its VA clauses, is found. After a model is found, the process can continue with backtracking to find more models.

A set of models can be partitioned into equivalence classes by isomorphisms. Intuitively, a model can be transformed into any other model in the same equivalence class by renaming its domain elements. Two models are said to be isomorphic to each other if an isomorphism exists from one model to the other.

The search space can be organized as a search tree in which nodes are VA clauses and edges join successive nodes with cell terms in the search order. The root node is an empty VA clause. The cell term in each node is selected by the cell selection strategy. A *search path* in a search tree is a path from the root to a node in the search tree. It can be represented by a sequence of VA clauses  $\langle t_0 = v_0; t_1 = v_1; \dots \rangle$ , where  $t_i$  is the cell term in the  $i^{\text{th}}$  position of the sequence and  $v_i \in D$ , and  $t_i \neq t_j$  when  $i \neq j$ . Furthermore, a search path will be terminated at the first VA clause that results in a violation of any axiom of  $\mathcal{F}$ .

If the length of a search path is the same as the total number of cell terms in  $\mathcal{F}$ , then it is a complete search path and its VA clauses represent a model. Otherwise, it is an incomplete search path representing *partial assignments* of cell values in  $\mathcal{F}$ .

The backtracking algorithm in its simplest form is to try every possible value assignment for every cell. For example, to search an FOL formula  $\mathcal{F}$  with just one binary operation, there are  $n^{n^2}$  possible combinations ( $n^2$  cells with  $n$  possible values each). Even the very small domain size of 4 gives over 4 billion combinations of cell values. However, in practice, the number of viable combinations to check is much smaller than the theoretically maximum number because of the constraints imposed by  $\mathcal{F}$ . Furthermore, a finite model finder may infer new VA clauses from existing ones by *propagation*.

► **Example 1.** Suppose the FOL formula contains only the equation  $f(x, y) = f(y, x)$ , that is, the operation  $f$  is commutative. After the assignment  $f(0, 1) = 0$ , the finite model finder can infer  $f(1, 0) = 0$ . This is referred to as positive propagation.

On the other hand, if the FOL formula contains the inequality  $f(x, y) \neq f(y, x)$ , then after the assignment  $f(0, 1) = 0$ , the finite model finder can exclude 0 from the list of possible values for the cell  $f(1, 0)$ . This is referred to as negative propagation. ◀

## 2.2 Least Number Heuristic

The least number heuristic (LNH) [4, 50, 51] is a very effective symmetry-breaking algorithm widely implemented in model finders/enumerators such as Mace4. The main idea of the LNH is that all domain elements that have not yet appeared in any VA clauses and the current cell term in the search are indistinguishable to each other and therefore only one of them, say, the smallest one, needs to be considered in a cell value assignment.

To ease discussions of the LNH, we introduce the notation  $\text{Vals}(P)$  to denote the set of all domain elements appearing in  $P$ , where  $P$  can be a search path, a VA clause, or a cell term.

► **Example 2.** For the cell term  $f(1, 1)$ :  $\text{Vals}(f(1, 1)) = \{1\}$ . For the VA clause  $f(1, 1) = 0$ :  $\text{Vals}(f(1, 1) = 0) = \{0, 1\}$ . For the partial search path  $S = \langle f(0, 0) = 0; f(1, 1) = 0 \rangle$ :  $\text{Vals}(S) = \{0, 1\}$ . ◀

The LNH can now be stated precisely: In adding a VA clause,  $t = v$ , to extend a search path  $S$ , the possible choices of  $v$  allowed under the LNH are  $\text{Vals}(S) \cup \text{Vals}(t) \cup \{s\}$  where  $s$  is the smallest domain element in  $D \setminus (\text{Vals}(S) \cup \text{Vals}(t))$ , and they are  $D$  if  $\text{Vals}(S) \cup \text{Vals}(t) = D$ . Strictly speaking, it is not necessary to set  $s$  to be the smallest domain element not seen so far, it could as well be the biggest one, for example. But the rule to set  $s$  must be unambiguous - only one value is consistently picked by the rule each time. In this paper, we always set  $s$  to be the the smallest domain element not seen so far.

Furthermore, we say a search path is *LNH-compliant* if it respects the LNH restrictions on the choices of values assigned to its VA clauses.

► **Example 3.** Suppose the domain size,  $|D|$ , is 4. Then the complete search path  $\langle f(1) = 0; f(0) = 3; f(3) = 1; f(2) = 1 \rangle$  is not LNH-compliant.

For the first VA clause in the search path,  $S = \emptyset$  and  $t = f(1)$ . So,  $\text{Vals}(S) \cup \text{Vals}(t) = \emptyset \cup \{1\} = \{1\}$ , and therefore  $D \setminus (\text{Vals}(S) \cup \text{Vals}(t)) = \{0, 2, 3\}$ . Thus,  $s = \min(\{0, 2, 3\}) = 0$ . The LNH limits the choices of the value for  $f(1)$  to  $\text{Vals}(S) \cup \text{Vals}(t) \cup \{s\} = \{0, 1\}$ . So the first VA clause  $f(1) = 0$  is LNH-compliant. However, for the second VA clause in the search path,  $S = \{f(1) = 0\}$  and  $t = f(0)$ . So,  $\text{Vals}(S) \cup \text{Vals}(t) = \{0, 1\} \cup \{0\} = \{0, 1\}$ , and therefore  $D \setminus (\text{Vals}(S) \cup \text{Vals}(t)) = \{2, 3\}$ . Thus,  $s = \min(\{2, 3\}) = 2$ . The LNH limits the choices of the value for  $f(0)$  to  $\text{Vals}(S) \cup \text{Vals}(t) \cup \{s\} = \{0, 1, 2\}$ , so  $f(0) = 3$  is not allowed under the LNH. Therefore, the whole search path is not LNH-compliant. ◀

The LNH does not impose any restrictions on the order of the cell terms in the search path<sup>2</sup>. It speeds up the search by limiting the choices of the values for the cell terms. Therefore, its effectiveness decreases with the increase in the length of the search path as more domain elements are used when more VA clauses are added to the search path.

<sup>2</sup> In practice, a number called the *maximal designated number* (*mdn*) is often used to partition the domain into 2 subsets so that  $\{0, \dots, \text{mdn}\}$  are domain elements already seen, and  $\{\text{mdn} + 1, \dots, n - 1\}$  are domain elements not seen so far [49]. In this case, cell selection strategies that keep the *mdn* small are preferred because the search tree will be kept narrower.

► **Example 4.** The *concentric cell selection strategy* is a simple cell selection strategy to minimize the growth of choices of values in the finite model search with the LNH. This strategy picks the cell  $f(a_0, \dots, a_{k-1})$  with the least  $r = \max(a_0, \dots, a_{k-1})$  from all available cells. Any fixed tie-breaker can be used in case of a tie. For example, one of the possible orders of the cells by this cell selection strategy for a binary operation is  $f(a_0, a_1) < f(b_0, b_1)$  if  $a_0 = a_1 \vee a_0 + a_1 < b_0 + b_1 \vee (a_0 + a_1 = b_0 + b_1 \wedge a_0 < b_0)$ . This gives the sequence  $f(0, 0)$ ,  $f(1, 1)$ ,  $f(0, 1)$ ,  $f(1, 0)$ ,  $f(2, 2)$ ,  $f(0, 2)$ ,  $f(2, 0)$ ,  $f(2, 1)$ ,  $f(1, 2)$ ,  $f(3, 3)$  . . . . ◀

## 2.3 Cube

A *cube* is a prefix of a search path, and as such, it can be specified by a sequence of VA clauses. Permutations and isomorphisms can be applied to a cube by applying them to its VA clauses. Specifically, if  $\pi$  is a permutation on  $D$  and  $B$  is a cube, then  $\pi(B) := \{f(\pi(a_1), \dots, \pi(a_k)) = \pi(v) \mid f(a_1, \dots, a_k) = v \text{ is a VA clause in } B\}$ . Observe that  $\pi_{id}(B)$  is the (unordered) set of all individual VA clauses in the cube  $B$ .

Note that predicates in an FOL formula can be implemented as functions with two values,  $T$  (true) and  $F$  (false), which do not affect the LNH because they are not domain elements. For convenience, we consider  $I(T) = T$  and  $I(F) = F$  for any isomorphism  $I$  so that the same terminology is used for both relations and functions.

Cubes are said to be isomorphic if their VA clauses are isomorphic. In particular, two cubes  $B_0$  and  $B_1$  are isomorphic if there is a permutation  $\pi$  on  $D$  such that  $\pi(B_0) = \pi_{id}(B_1)$ .

► **Example 5.** If  $B_0 = \langle f(0) = 0; g(0, 0) = 0; f(1) = 0; g(1, 1) = 0 \rangle$  and  $B_1 = \langle f(0) = 1; g(0, 0) = 1; f(1) = 1; g(1, 1) = 1 \rangle$ , then  $B_0$  and  $B_1$  are isomorphic because  $\pi_{(0,1)}(B_0) = \langle f(1) = 1, g(1, 1) = 1, f(0) = 1, g(0, 0) = 1 \rangle = \pi_{id}(B_1)$ . ◀

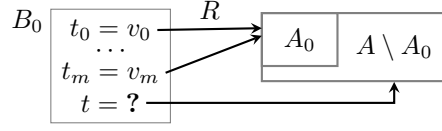
## 3 Isomorphic Cubes Redundancy

The main objective of this section is to show that isomorphic cubes can be removed from the search. More formally, if cubes  $B_0$  and  $B_1$  are isomorphic, then it is sufficient to explore assignments extending  $B_0$  and ignore *all* assignments extending  $B_1$ . We need to prove that any model lost by discarding  $B_1$  must necessarily be isomorphic to some model obtained from extending  $B_0$  under the LNH. This statement is intuitive, but the proof requires some care as effectively, we are dealing with a combination of two symmetry-breaking techniques: LNH and isomorphic cube pruning, under an arbitrary search strategy.

As a motivational example, consider the cube  $\langle f(0, 0) = 0 \rangle$ , which states that  $f$  is idempotent in 0. But because 0 does not appear in the original FOL formula, intuitively, the constant 0 cannot play a special role in the formula. Consequently, this cube searches *all* interpretations of  $f$  that have at least one idempotent. For instance, the cube  $\langle f(1, 1) = 1 \rangle$  will search the same interpretations, up to isomorphism. Now, we need to show this property formally and that it holds when the solver searches with the LNH restriction.

The key idea of the proof is that given a model  $B_1$  with VA clauses  $A$ , any cube that is isomorphic to a subset of  $A$  can be gradually extended to be a model isomorphic to  $B_1$ . Each extension step of the cube must uphold the following properties: (1) The cube is isomorphic to some subset of  $A$ . (2) The cube is LNH-compliant. The extension step is illustrated in Figure 1. We are given a cube  $B_0$  that is isomorphic to an  $A_0 \subseteq A$ . When the finite model finder decides on some empty cell  $t$ , we need to show that it is possible to find a value according to the LNH such that the extended cube is isomorphic to some subset of  $A$  containing  $A_0$ .





■ **Figure 1** Extension of a cube according to the VA clauses  $A$ .

► **Notation 1.** For a mapping  $R$  from  $D$  to  $D$  and a value  $d \in D$  we write  $\mathcal{E}_R^d$  for a mapping that maps  $d$  to  $R(d)$  if  $d \in \text{dom}(R)$  and otherwise maps  $d$  to  $\min(D \setminus \text{rng}(R))$ . We further write  $\mathcal{E}_R^{d_1, \dots, d_k}$  for successive extensions by  $d_1, \dots, d_k$ , i.e.  $\mathcal{E}_R^{d_1, d_2} = \mathcal{E}_{\mathcal{E}_R^{d_1}}^{d_2}$  etc. ◀

► **Example 6.** Suppose  $D = \{1, 2, 3\}$  and  $R : \{1\} \rightarrow \{2\}$  s.t.  $R(1) = 2$  and  $R^{-1}(2) = 1$  (so,  $R$  is a bijection). Then  $\mathcal{E}_R^2$  s.t.  $R(2) = \mathcal{E}_R^2(2) = 1$  is a valid extension of  $R$  because  $\min(D \setminus \{2\}) = 1$ . Furthermore,  $\mathcal{E}_R^{2,1}$  s.t.  $\mathcal{E}_R^{2,1}(1) = 2$  is a valid (but trivial) extension of  $\mathcal{E}_R^2$ . ◀

► **Lemma 7.** If  $R$  is a bijection between some  $D_0, D_1 \subseteq D$  and  $d \in D$  then  $\mathcal{E}_R^d$  is well-defined and also a bijection.

**Proof.** If  $d \in D_0$ , then  $\mathcal{E}_R^d = R$  and there is nothing to prove. If  $d \in D \setminus D_0$ , then by definition,  $\mathcal{E}_R^d = R \cup \{(d, p)\}$  for some  $p \in D \setminus D_1$ . Since  $R$  is a bijection from  $D_0$  to  $D_1$ ,  $d \notin \text{dom}(R)$ , and  $p \notin \text{rng}(R)$ , so  $\mathcal{E}_R^d$  is well-defined, one-one, and onto. That is, it is a bijection. ◀

► **Notation 2.**  $B \oplus \langle t = u \rangle$  is the new cube formed by extending the cube  $B$  with the VA clause  $t = u$ . ◀

The following lemma is the core of our proof. We have a cube  $B$  isomorphic to some partial assignment  $A_0$  and now we need to prove that for *any* model  $A$  completing  $A_0$  and *any* search strategy, we are able to extend  $B$  while observing the LNH. Then, the lemma is used to prove that isomorphic cubes can be discarded by induction on cube length (Theorem 9).

► **Lemma 8.** Let  $B$  be an LNH-compliant cube and  $A$  a model s.t.  $B$  is isomorphic to some  $A_0 \subseteq A$ . Then for any cell term  $t$  not appearing in  $B$ , there exists a value  $u$  and a VA clause  $t' = u' \in A \setminus A_0$ , s.t.  $B \oplus \langle t = u \rangle$  is LNH-compliant and isomorphic to  $A_0 \cup \{t' = u'\}$ .

**Proof.** Let  $R$  be an isomorphism mapping  $B$  to  $A_0$  and let  $t$  be a cell term  $f(a_1, \dots, a_k)$ . Define  $R_1$  as  $\mathcal{E}_R^{a_1, \dots, a_k}$ , and let  $t'$  denote the cell term  $f(R_1(a_1), \dots, R_1(a_k))$ , i.e., map the cell that the solver searches on into a cell in the prescribed model  $A$ .

Since  $A$  is a model, there must exist a value  $u' \in D$  with  $(t' = u') \in A$ , i.e.  $u'$  can be found by a lookup of  $t'$  in  $A$ . Since  $t$  is not a cell term in  $B$  and  $R_1$  is a bijection, so  $t'$  is not a cell term in  $A_0$  and must therefore be in  $A \setminus A_0$ . Thus,  $t' = u'$  is a VA clause in  $A \setminus A_0$ .

To obtain  $u$  (a value for cell  $t$ ), define  $R_2$  as  $\mathcal{E}_{R_1}^{u'}$ , i.e. map  $u'$  back into the search by extending the inverse. Then, set  $u = R_2(u')$ . By Lemma 7,  $R_2$  is bijection and it is therefore an isomorphism from  $A_0 \cup \{t' = u'\}$  to  $B \oplus \langle t = u \rangle$ . Finally, by definition of  $R_2$ ,  $u$  either already appears in  $B$  or otherwise is the smallest domain element not in  $B$ . Therefore, the extension of the cube  $B$  by the VA clause  $t = u$  is LNH-compliant. ◀

► **Theorem 9.** Suppose we are searching under the LNH with any cell selection strategy on a signature  $\Sigma$  and a FOL formula,  $\mathcal{F}$ , on  $\Sigma$ . If  $B_0$  and  $B_1$ , of length  $l \geq 0$ , are isomorphic cubes, and if  $M_1$  is a model obtained by completing (not necessarily under the LNH) the search path in  $B_1$ , then  $B_0$  can be extended by a search path  $S$  under the said LNH and cell selection strategy to a model  $M_0$  which is isomorphic to  $M_1$ .

**Proof.** We will use mathematical induction on the length of the extension,  $m$ , on  $S$  to prove the theorem. Let  $A$  denote the VA clauses of  $M_1$ , and  $A_0$  denote the VA clauses of  $B_1$ .

Base case is trivial as  $B_0$  and  $B_1$  are given as isomorphic when  $m = 0$ .

Induction step: Suppose the search path  $S$  is extended  $m$  times, where  $m > 1$ , so that  $S_m$  is LNH-compliant and isomorphic to a subset  $A_m \subseteq A$ . Then by Lemma 8,  $S_m$  can be extended by one VA clause with the cell term  $t_{m+1}$ , chosen by the said cell selection strategy, to  $S_{m+1}$  which is LNH-compliant and isomorphic to  $A_{m+1} \subseteq A$ .

Note that a model finder may do propagations after a cell value assignment. That is, some cell terms can be assigned values inferred from existing VA clauses. Propagations can be viewed as part of the cell selection strategy and be handled the same way as regular cell value assignments.

We can therefore conclude by mathematical induction that  $S$  can be extended to a complete search path when all cell terms in  $\mathcal{F}$  are filled with values such that  $S$  represents the model  $M_0$ , is LNH-compliant, and is isomorphic to  $A_s \subseteq A$ . Since  $M_0$  and  $M_1$  are of the same size, so  $A_s$  and  $A$  must necessarily be of the same size and hence must be equal. Therefore,  $M_0$  is isomorphic to  $M_1$ . ◀

Theorem 9 shows that isomorphic cubes always extend to isomorphic models. So, one of the isomorphic cubes may be discarded without losing any non-isomorphic model.

► **Corollary 10.** *On searching under the LNH with any cell selection strategy on a signature  $\Sigma$  and an FOL formula  $\mathcal{F}$  on  $\Sigma$ , if  $M_1$  is a model in  $\mathcal{F}$ , then there is a complete search path  $S$  under the said LNH that results in a model  $M_0$  which is isomorphic to  $M_1$ .*

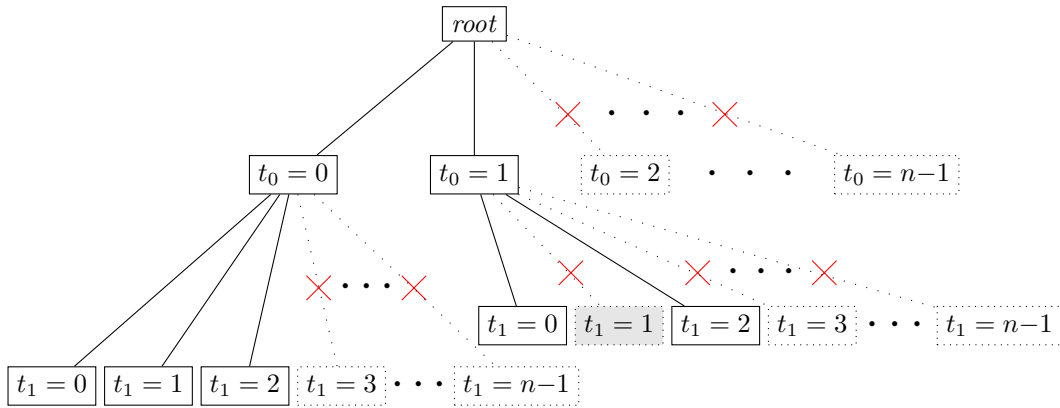
Corollary 10 proves the completeness of the LNH in that every model in any search is isomorphic to some model found by searching under the LNH. An alternative proof of the corollary is given in [50].

## 4 Searching with Cubes

Cubes can be constructed to partition the search space into non-overlapping subtrees that can be processed in parallel. It is not necessary to search all the subtrees that originate from the collection of cubes that span the entire search space because isomorphic cubes in the same collection can be eliminated without losing non-isomorphic models. For example, suppose we want to search for models of order 3 or more on a function  $f : D^2 \rightarrow D$  under the LNH with a cell selection strategy that selects  $f(0,0)$  then  $f(1,1)$  as the first 2 cell terms in the search process. There are at most 6 cubes of length 2 (listed below) under the said LNH and cell selection strategy, so together they must span the whole search space in the sense that every search path that starts with the cell terms  $f(0,0)$  then  $f(1,1)$  in the search tree must include one of the 6 cubes in it.

1.  $\langle f(0,0) = 0; f(1,1) = 0 \rangle$ .
2.  $\langle f(0,0) = 0; f(1,1) = 1 \rangle$ .
3.  $\langle f(0,0) = 0; f(1,1) = 2 \rangle$ .
4.  $\langle f(0,0) = 1; f(1,1) = 0 \rangle$ .
5.  $\langle f(0,0) = 1; f(1,1) = 1 \rangle$ .
6.  $\langle f(0,0) = 1; f(1,1) = 2 \rangle$ .

Since  $\pi_{(0,1)}(\text{Cube 1}) = \{f(1,1) = 1, f(0,0) = 1\} = \pi_{id}(\text{Cube 5})$ , so Cubes 1 and 5 are isomorphic and one of them can thus be discarded without losing non-isomorphic models per Theorem 9. This example demonstrates the importance of keeping the LNH in the search –



Note:  $t_0$  denotes  $f(0,0)$  and  $t_1$  denotes  $f(1,1)$ . A dotted line with a cross is a branch pruned by the LNH, except for the branch ending on the VA clause  $t_1 = 1$  (the shaded node), which is pruned by the isomorphic cubes removal algorithm.

■ **Figure 2** Partial Search Tree Showing Cubes of Length 2.

it cuts the search space from potentially  $n^2$  cubes down to 6. Theorem 9 allows us to further cut the number of cubes down to 5 (see Figure 2 for illustration). More isomorphic cubes can be removed with longer cubes (see Table 2).

The procedure of removing isomorphic cubes starts with generating a set of short cubes (typically of length 2 for a binary operation) that spans the entire search space. The model finder takes short cubes as inputs and runs with them as if they are generated by itself to generate longer cubes of predefined length  $l$ . Specifically, the model finder runs as usual, except that it emits the cubes of length  $l$  when the depth of the search tree reaches  $l$ . After outputting the cube, the model finder backtracks as if it has reached the bottom of the search tree, and runs on a new branch as usual until all cubes of length  $l$  are generated. Some models may be generated in this process due to propagation, and they are kept as part of the final outputs. Next, the cubes are compared for isomorphism and only one of any pair of isomorphic cubes is kept. This new set of non-isomorphic cubes of length  $l$  will be used as inputs to the model finder in the next round of generation of longer cubes. The process is repeated until the desired length of cubes is reached.

For searching models defined by one operation of arity  $k$ , we use the sequence of lengths  $l$ :  $k, 2^k, 3^k, 4^k, \dots$ . This is to match the concentric cell selection strategy (see Example 4 for its definition) of the finite model finder such as Mace4. We will discuss the best cube length to use in Section 5.3.

Finally, non-isomorphic cubes of the target length can then be processed independently in parallel and their output models collected separately.

## 4.1 Invariants

To speed up the isomorphic cubes removal process, the same invariant-based algorithm described in [2] to remove isomorphic models can be applied to cubes. Invariants, such as number of distinct domain elements, are properties that must be identical for cubes to be isomorphic. For example, the cubes  $A = \langle f(2,2) = 2; f(2,3) = 4 \rangle$  and  $B = \langle f(3,3) = 2; f(1,2) = 2 \rangle$  cannot be isomorphic because  $A$  contains an idempotent 2 but  $B$  does not. Powerful and inexpensive invariants for binary operations include:

1. Number of  $y$  such that  $x = (x * y) * x$ .
2. Number of  $y$  such that  $y = x * z$  for all  $z \in D$ .
3. Number of  $y$  such that  $y = z * y$  for all  $z \in D$ .
4. Number of idempotents  $x$  (i.e.  $x * x = x$ ) for all  $x \in D$ .
5. Number of  $y$  such that  $y * y = x$  for each  $x \in D$ .

First, invariant vectors (i.e. ordered lists of invariants) for cubes are calculated and used as hash keys to group cubes having the same invariant vectors into hash buckets. Then, cubes within the same bucket are tested for isomorphism. There is no need to test for isomorphism across buckets because isomorphic cubes must have the same invariant vectors. This saves tremendous amounts of testing time. Furthermore, buckets can be processed independently and in parallel to further speed up the process.

## 4.2 Work Stealing

In the basic form of this cube-based parallel algorithm, cubes are statically generated before the model enumeration process begins. It has the advantage of low runtime overheads as no synchronization among running finite model finders is needed. The preprocessing time for generating the cubes is also small for short to medium-length cubes. The disadvantage is that the workload may be uneven among the parallel processes. Some jobs may take a long time to finish when free workers sitting idle after finishing their jobs.

This problem can be solved with *work stealing* algorithms (also used in SAT [26]) in which a busy finite model searcher releases cubes that are not currently being worked on. For example, suppose a running model searcher is working on a cube  $B_0 = \langle f(0,0) = 0 \rangle$ , and its cell selection strategy picks the cell  $f(1,1)$  to assign value next. Under the LNH,  $f(1,1)$  may be assigned a value from  $\{0, 1, 2\}$ . If the model searcher is requested to spin out some work for other free workers, then it generates three cubes,  $B_0 = \langle f(0,0) = 1; f(1,1) = 0 \rangle$ ,  $B_1 = \langle f(0,0) = 1; f(1,1) = 1 \rangle$ , and  $B_2 = \langle f(0,0) = 1; f(1,1) = 2 \rangle$ . It continues to work on the cube  $B_0$  and releases  $B_1$  and  $B_2$  to other free workers.

## 5 Experimental Results

We integrate the cube-based algorithms into the finite model enumerator Mace4, which supports searching on FOL with the LNH and many cell selection strategies [35]. Parallelization is controlled outside Mace4. Only minor changes are made to Mace4 to

1. Accept cubes as inputs and continue searching for longer cubes or models from them.
2. Periodically check for signal for work stealing to spin off cubes for other workers.

The model searching logic in Mace4 remains intact. The concentric cell selection strategy (see Example 4 for its definition) is used in the experiments. A separate program removes isomorphic cubes by separating the cubes with equal invariants then check for isomorphisms (two cubes are isomorphic if one can be transformed to the other by a permutation).

We run the experiments on an Intel® Xeon®Silver 4110 CPU 2.0 GHz  $\times$ 32 computer, with 64 GB of random access memory (RAM), using 30 parallel processes unless otherwise stated. All times reported are wall clock times.

We pick many disparate and challenging problems from the MarcieX database [3], which contains a collection of 158 most popular algebras. We also draw an example of semigroup subvariety from [1]. The definitions of the algebras used in the experiments in this section are listed in Table 1, in which all clauses are implicitly universally quantified.

■ **Table 1** Definitions of Algebras Used in Experiments.

Algebra	FOL Definition
Semigroups	$x * (y * z) = (x * y) * z.$
Loops	$x * y = x * z \rightarrow y = z. \quad y * x = z * x \rightarrow y = z. \quad x * 0 = x. \quad 0 * x = x.$
$\text{var}\{N_2^1 \cap [x^2 = y^2]\}$	$x * (y * z) = (x * y) * z. \quad (x * x) * x = x * x. \quad x * y = y * x. \quad x * x = y * y.$
Tarski Algebras	$(x * y) * y = (y * x) * x. \quad x * (y * z) = y * (x * z). \quad (x * y) * x = x.$
Quasi-ordered Set	$x < y \wedge y < z \rightarrow x < z. \quad x < x.$
Involutive Lattices	$(x * y) * z = x * (y * z). \quad x * y = y * x. \quad (x + y) + z = x + (y + z). \\ x + y = y + x. \quad (x * y) + x = x. \quad (x + y) * x = x. \\ -(x + y) = -x * -y. \quad - - x = x.$

In the tables showing experimental results in this section, the rows with cube length 0 show the results of running Mace4 in a single thread without the cube-based algorithms.

Table 2 shows the results of applying Theorem 9 to remove isomorphic cubes for the binary operation of the semigroups of order 7. Observe that the percentage reduction of the number of cubes increases as the cube length increases. The isomorphic cubes removal algorithm is therefore complementary to the LNH because the LNH removes a lot of short cubes but loses its effectiveness as the length of the cubes grows.

■ **Table 2** #Cubes for Semigroups of Order 7.

Cube Length	# Cubes		Reduction (%)
	w/o Removal of Isomorphic Cubes	w/ Isomorphic Cubes Removed	
2	6	5	16.7
4	34	28	17.6
9	1,568	888	43.4
16	56,206	12,036	78.2
25	1,028,171	59,056	94.3

We run Mace4 to enumerate semigroups defined by a single binary operation. The results show a speedup of over 100 times when cubes of length 25 are used, with over 96% of the isomorphic models suppressed (see Table 3). The results on semigroups are indicative of the algorithm’s usefulness in general to the computational algebraists because algebraic structures related to semigroups are ubiquitous in algebra. Not only are there many well-known semigroup-related algebras, but also many semigroup varieties and subvarieties that are of high research interests [1].

Table 4 shows the results for loops (a quasigroup-related class of algebra) defined by a single non-associative binary operation. Here the reduction in the number of the output isomorphic models is not as pronounced. This is expected because the LNH works very well with the Latin square and removes a high percentage of the isomorphic models [48] before the isomorphic cubes removal takes place. For example, while only 0.16% of semigroups of order 7 generated by the LNH are non-isomorphic, 8.7% (106,228,849 out of 1,216,226,816) of the models generated for the loops of order 8 under the LNH are non-isomorphic. Nevertheless, the parallel algorithm provides 15 times improvement in speed for cube length of 16.

Table 5 shows the results of running the algorithms on the semigroup subvariety  $\text{var}\{N_2^1 \cap [x^2 = y^2]\}$  (see p. 40 of [1] for its definition and discussions). With longer cubes, the algorithms speed up the process by 26 times with 30 threads. The results confirm that the proposed algorithms work remarkably well with semigroup-related algebras.

The Tarski algebras are unlike both the semigroups and the quasigroups in that its multiplication table is not associative and is not a Latin square [3]. It shows the cube-based algorithms perform better and better as the length of the cube increases (see Table 6).

The quasi-ordered set is defined by one binary relation. The isomorphic cubes algorithms work well on relations just as it works well on functions. As shown in Table 7, when cubes of length 36 are used, over 99% of the isomorphic models are suppressed, and the search process is sped up by over 200 times.

As an example to demonstrate the effectiveness of the algorithms on more complex algebras, consider the Involutive Lattice [3], which is defined by two associative binary operations and one unary operation. For Involutive Lattices of order 13, the search tree has a maximum depth of 351. Using cubes of length of 105, we obtain a speedup of 300 times, with almost 98% of the isomorphic cubes suppressed (see Table 8). The results show that the isomorphic cubes algorithms are highly effective for both simple and complex algebras.

■ **Table 3** Running Cubes on Semigroups of Order 7.

Cube Length	#Cubes	#Models (Millions)	Time in min.	
			Gen. Cubes	Total
0		1,021.1		235.2
2	5	717.7	0.0	12.5
4	28	611.1	0.1	9.4
9	888	360.2	0.1	5.2
16	12,036	158.2	0.2	2.8
25	59,056	39.5	0.9	1.7

■ **Table 4** Running Cubes on Loops of Order 8.

Cube Length	#Cubes	#Models (Millions)	Time in min.	
			Gen. Cubes	Total
0		1,216		564.0
2	1	1,216	0.0	47.4
4	2	1,216	0.1	47.3
9	18	1,216	0.1	46.2
16	3,583	1,214	0.1	45.3

■ **Table 5** Running Cubes on  $\text{var}\{N_2^1 \cap [x^2 = y^2]\}$  of Order 9.

Cube Length	#Cubes	#Models (Millions)	Time in min.	
			Gen. Cubes	Total
0		313.0		72.0
2	1	156.5	0.0	2.9
4	1	156.5	0.1	2.8
9	2	156.5	0.1	2.8
16	5	120.9	0.1	2.3
25	16	55.5	0.2	1.3
36	70	13.0	0.3	0.8
49	331	1.5	1.0	1.1

■ **Table 6** Running Cubes on Tarski Algebras of Order 13.

Cube Length	#Cubes	#Models (Millions)	Time in min.	
			Gen. Cubes	Total
0		379.6		1,949.9
2	3	189.8	0.0	70.2
4	1	189.8	0.1	69.9
9	3	183.3	0.1	67.7
16	11	158.8	0.1	58.1
25	55	111.9	0.2	40.1
36	157	62.1	0.2	21.8
49	174	24.9	0.5	8.8
64	171	6.6	1.0	3.7

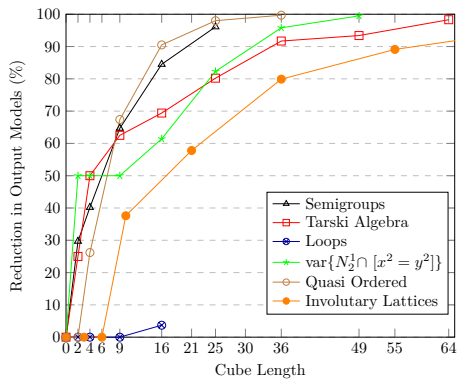
■ **Table 7** Running Cubes on Quasi-ordered Set of Order 8.

Cube Length	#Cubes	#Models (Millions)	Time in min.	
			Gen. Cubes	Total
0		642.8		59.9
2	1	642.8	0.0	4.2
4	3	474.6	0.1	3.2
9	9	209.5	0.1	1.7
16	33	61.3	0.1	0.8
25	139	12.6	0.2	0.3
36	713	2.0	0.3	0.3

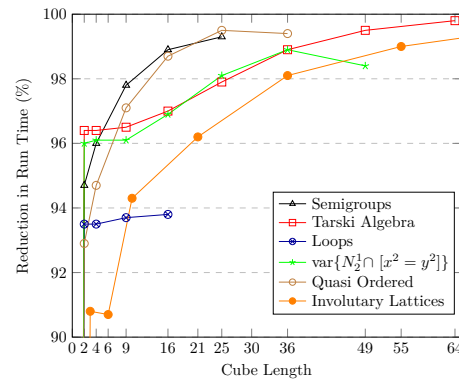
■ **Table 8** Running Cubes on Involutive Lattices of Order 13.

Cube Length	#Cubes	#Models (Millions)	Time in min.	
			Gen. Cubes	Total
0		423.0		4,719.7
3	2	423.0	0.0	432.5
6	3	423.0	0.1	432.8
10	6	263.9	0.1	270.0
21	23	178.6	0.1	180.9
36	108	84.9	0.2	88.3
55	555	46.0	0.3	46.2
78	1,710	19.8	0.5	20.6
105	5,048	8.7	4.9	14.3

The reductions in time and number of models (on top of the LNH) are summarized in Figures 3 and 4. Note that the reduction in total time is over 90% even for short cubes. However, the biggest gain in both reduction in time and in isomorphic models is when longer cubes are used. Reduction in isomorphic models also helps tremendously in the post processing step to extract non-isomorphic models.



■ **Figure 3** Reduction in Number of Output Models.



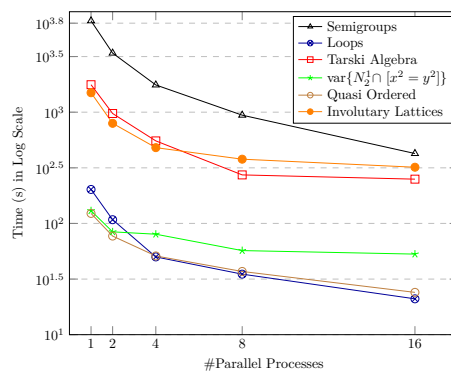
■ **Figure 4** Reduction in Total Time with 30 Parallel Processes.

### 5.1 Speedup of Finite Model Enumeration with Parallelization

As discussed, the cubes algorithms allow low-cost parallelization of the finite model enumeration process. Figure 5 and Table 9 show the performance of the parallel cubes algorithms with 1 to 16 parallel processes. Here, the reported times do not include isomorphic mode filtering; they are for Mace4 to generate models only. Note that when many processes compete for limited amount of RAM, swapping could slow down the processes substantially. This helps to explain why larger algebras, such as the Involutive Lattice of order 13, have their curves flattened out much faster than small algebras, such as the Semigroups of order 7. More processes also mean more work-stealing and higher overheads.

■ **Table 9** Performance w/ Multiprocessing.

Algebra	Order	Cube Length	Time in seconds				
			1 Process	2 Processes	4 Processes	8 Processes	16 Processes
Semigroups	7	25	6,626	3,397	1,757	940	425
Loops	7	16	202	108	50	35	21
Tarski algebras	13	64	1,766	973	552	273	250
$\text{var}\{N_2^1 \cap [x^2 = y^2]\}$	9	49	130	84	80	57	53
Quasi Ordered	8	36	123	77	51	37	25
Involutive Lattices	12	105	1,496	794	480	378	320



■ **Figure 5** Performance w/ Multiprocessing.

## 5.2 Isomorphic Cubes Removal Speeds up Isomorphic Models Filtering

As pointed out Section 1, reducing the number of Mace4 outputs also reduces the efforts needed to filter out isomorphic models. Table 10 shows, using involutive lattices as an example, the out-sized effect of the reduction of Mace4 outputs on the time to filter out the isomorphic models using the invariant-based isomorphic model filtering algorithm [2], with 30 parallel processes. With the reduction in number of Mace4 models, the isomorphic model filtering process is sped up by 2 orders of magnitude. The improvement in speed is observed to be better with models of higher orders. We would also point out that the isomorphic model filter generates the same non-isomorphic models with or without the cubes algorithms.

■ **Table 10** Running Invariant-based Isomorphic Models Filter on Involutive Lattices.

Order	w/o Cubes			w/ Cubes		
	#Non-iso Models	#Mace4 Output	Isomorphic Model Filter Time (s)	Cube Length	#Mace4 Output	Isomorphic Model Filter Time (s)
9	122	72,470	29	78	3,670	1
10	389	575,463	496	105	13,789	4
11	906	4,771,035	28,424	105	97,680	135
12	3,047	43,851,030	N/A	105	971,416	2,802



### 5.3 Optimal Cube Length

In general, the search process using longer cubes finishes earlier with fewer isomorphic models. However, we observe that there are three limiting factors on the lengths of the cubes.

First, as the length of the cubes gets longer, more and more models are generated as a result of propagations. This reduces the impact of removing isomorphic cubes because they represent a progressively smaller proportion of the isomorphic models. It is observed that when more than  $n - 2$  symbols out of the  $n$  domain elements are used in the cell terms, the number of (isomorphic) models will be substantial and extending the cube length does not bring enough reduction in isomorphic models to justify the increase in processing time.

Second, the isomorphic cubes removal time grows quite fast as the length of the cube grows. When the isomorphic cubes removal process takes more than a few minutes, further lengthening of the cubes will result in prohibitive overheads in the search process.

Lastly, when the final number of cubes is more than tens of thousands, the overheads in processing them becomes so high that the search becomes slower. This factor depends heavily on the number of processors available. More processors mean more parallel processes can be run without slowing down the whole search process.

One heuristic is to run cube generation until the number of cubes reaches some threshold or the runtime exceeds some threshold, then switch to model generation. The thresholds are system-dependent and can further be fine-tuned by experiments with algebras of interest.

## 6 Related Work

There is extensive research on *paralyzing SAT solving*, where the predominant approaches are search space partitioning and portfolios, c.f. [33]. We find inspiration in the *cube-and-conquer* approach proposed by Heule and colleagues [20–22], where the search space is partitioned by a lookahead solver into (many) cubes and then each subspace is solved by a CDCL SAT solver. In SAT, partitioning by a CDCL solver is nontrivial [32] and that is why the lookahead solver is useful for this task. Nevertheless, the use of the lookahead solver is not seen as an indispensable feature of the cube-and-conquer, as noted by Subercaseaux and Heule [46]. In our approach, we have a tight control over the decisions of the solver and we do not need a separate solver to perform the splitting. Additionally, we invest extra effort into search space splitting by identifying symmetries in the cubes.

The *adaptive prefix-assignment technique* [25] is a symmetry reduction algorithm used in SAT. The prefix is equivalent to a propositional cube, and the algorithm also tries to eliminate isomorphic cubes. In our case, we exploit symmetries specific to FOL – LNH and isomorphism at FOL level, which is absent in their algorithm (and in SAT in general).

*Parallel algorithms* can be characterized by how the search is done. There are two main search methods: embarrassingly parallel search (EPS) and work stealing search [7, 10, 26, 33, 41, 42]. In the former method, the task is decomposed into many sub-tasks that are queued up to be processed by free worker threads/processes. In the latter method, when a worker completes its task, it asks other workers for more work. The busy workers may split their tasks into smaller sub-tasks and pass some of them to the free workers. The main focus of this method is to keep all the CPUs running until all jobs are done, although for some cases, the work stealing scheme can affect efficiency [10]. The EPS method is a natural choice for the cube-based parallelization scheme because preprocessing can be performed to generate numerous non-isomorphic cubes by splitting the search space. However, a work-stealing procedure is essential in supplementing the EPS to balance uneven workloads [33].

Parallel algorithms can also help select the best strategy in solving a problem with the EPS method [39]. After a problem is decomposed into a large number of sub-tasks, a small number (e.g., 1%) of these sub-tasks are run in parallel using different strategies of the same solver or different solvers. The strategy that gives the best performance on the subset of sub-tasks will be used to run all sub-tasks. The same idea is used in the invariant-based isomorphic models removal algorithm [2]: it randomly generates a large number of invariants, then applies them to a small percentage of models to pick the best performing random invariants to apply to the whole set of models. This idea can be applied to the finite model finders that support multiple cell selection strategies to pick the best function order and cell selection strategy for any specific problem.

Finite model enumeration can be posed as a *constraint programming (CP)* task [27]. Some CP solvers, e.g., Minion [17] and Gecode [37], support parallelization [31]. In CP, the search space can be partitioned by adding constraints to rule in and/or out partitions. Each partition can be processed by a separate worker thread/process. Minion further implements a work stealing search scheme that also partitions the search space dynamically by splitting the existing constraint model after the search has started [15, 29]. However, to effectively add *symmetry-breaking* constraints such as lex-leaders to a CP solver often requires deep knowledge of the solver *and* the problem at hand (e.g., the semigroups in [15]) which may not be available when mathematicians first define and study a new algebraic structure.

Moreover, to use traditional CP solvers for finite model enumerations, mathematicians need to learn a new CP-specific language such as CHR [45] and Savile Row [38]. It is possible to use a translator to translate between languages, but that adds uncertainties to the fidelity and the optimality of the translated specifications. FOL remains one of the most popular languages among mathematicians due to its simple and intuitive syntax. Moreover, a popular automatic theorem prover, Prover9 [34], shares the same input language with Mace4. This adds more than just convenience to the process, as it also reduces the chances of discrepancies between Prover9 and Mace4 on the same problem.

A well-known issue with enumerating models defined with FOL are the isomorphic models included in the outputs. This is an inherent symmetry property of FOL [40]. There is extensive research on *symmetry-breaking* [4, 11–13, 28, 40, 43, 47]. Although complete symmetry-breaking is known to be computationally challenging [13, 47], many useful algorithms, such as the LNH and the XLNH [4, 5], have emerged in partial symmetry-breaking. The LNH can be considered a symmetry-breaking with interchangeable values in constraint satisfaction problems (CSP) [19]. The XLNH is more restrictive as it only works on unary operations. The LNH is implemented in many systems such as Falcon [50], SEM [51], FMSET [6], and Mace4. The isomorphic cubes algorithm, which removes more cubes as the cube length grows, complements the LNH.

Another important symmetry-breaking strategy is to steer the search engine away from the fruitless exploration of sub-search space by adding symmetry-breaking input clauses [13, 47]. The cube-based parallel algorithms are compatible with algorithms of this kind of strategy as long as they do not break the LNH.

Some finite model finders, such as SEMK [8] and SEMD [24], try to completely suppress isomorphic models in the search process. However, these isomorph-free algorithms are not easy to parallelize as global information is generated and consumed in many steps, requiring high-cost synchronizations between cooperating workers, especially when they run on different computers. The cube-based parallel algorithm, on the other hand, is an EPS method that requires no synchronizations between workers. The static removal of isomorphic cubes done in a preprocessing step is shown to be effective in suppressing isomorphic models even

before the actual search begins. The augmented work stealing algorithm is not high in synchronization costs because it does not involve communications between running jobs. The remaining isomorphic models from the cube-based algorithms can be efficiently removed by the invariant-based isomorphic model filtering algorithm as a postprocessing step.

Another algorithm, DSYM [4], exploits local symmetries by finding symmetries (synonym to isomorphisms in their terminology) under invariant partial interpretations (which are invariant cubes) and without parallelism. It also works with the LNH and XLNH. DSYM is a predictive algorithm that works at the *parent* level and predicts which of its immediate children will be isomorphic cubes. It can be seen as a special case of the isomorphic cube algorithm because it removes isomorphic cubes having the same immediate parents, while the isomorphic cube algorithm removes all isomorphic cubes, irrespective of their parents. Nevertheless, for the cases that DSYM covers, it does so right before the cubes are generated, while the isomorphic cube algorithm only detects the symmetries right after the cubes are generated. A disadvantage of DSYM is that it is not clear how it can be effectively parallelized. Furthermore, DSYM only detects symmetries under the same subtree. The isomorphic cubes removal algorithm, on the other hand, detects both global and local symmetries the same way, and hence detects and removes more symmetries than DSYM. Moreover, DSYM uses only two invariants in testing isomorphism between cubes, while we use many invariants that are proven successful in the invariant-based isomorphic model removal algorithm in our isomorphic cubes removal process. Nevertheless, DSYM can be applied to the cube generation process as well as the final model generation process. That is, the isomorphic cube removal algorithm is compatible with DSYM, as with any other symmetry-breaking algorithm that works with the LNH.

## 7 Conclusions and Future Work

In this paper, we introduce an efficient parallel algorithm together with a novel symmetry-removal mechanism for enumerating finite models. The approach is inspired by the cube-and-conquer paradigm, successfully used in SAT solving, which partitions the search space into cubes and then massively paralyzes. In contrast, our approach applies symmetries specific to finite model finding.

In conclusion, this paper fulfills an important unmet need for an efficient algorithm for enumerating finite algebraic models in computational algebra by enhancing the existing finite model enumeration process with the parallel cubes algorithm and the isomorphic cubes removal algorithm that reduce both the runtime and the number of output isomorphic models. These new algorithms are so scalable that they can be used on a laptop as well as on a cluster of powerful computers, and they require minimal efforts to safely integrate into existing finite model finders. Very importantly, these algorithms can be used as a black-box without requiring the users to have any knowledge about the way they work.

Future research will focus on improving isomorphic cube removal, on best cell selection strategy, and on predicting of optimal cube length.

---

## References

- 1 João Araújo, João Pedro Araújo, Peter J. Cameron, Edmond W. H. Lee, and Jorge Raminhos. A survey on varieties generated by small semigroups and a companion website, 2019. [arXiv:1911.05817](https://arxiv.org/abs/1911.05817).
- 2 João Araújo, Choiwah Chow, and Mikoláš Janota. Boosting isomorphic model filtering with invariants. *Constraints*, 27(3):360–379, July 2022. doi:10.1007/s10601-022-09336-x.

- 3 João Araújo, David Matos, and João Ramires. MarciEDB: a model and theory database. <https://marciedb.pythonanywhere.com>, 2022.
- 4 Gilles Audemard, Belaid Benhamou, and Laurent Henocque. Predicting and detecting symmetries in FOL finite model search. *J. Autom. Reason.*, 36(3):177–212, 2006. doi:10.1007/s10817-006-9040-3.
- 5 Gilles Audemard and Laurent Henocque. The eXtended least number heuristic. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR*, volume 2083 of *Lecture Notes in Computer Science*, pages 427–442, Berlin, Heidelberg, 2001. Springer. doi:10.1007/3-540-45744-5\_35.
- 6 Belaid Benhamou and Laurent Henocque. A hybrid method for finite model search in equational theories. *Fundam. Informaticae*, 39(1-2):21–38, 1999. doi:10.3233/FI-1999-391202.
- 7 R.D. Blumofe and C.E. Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 356–368, 1994. doi:10.1109/SFCS.1994.365680.
- 8 Thierry Boy de la Tour and Prakash Countcham. An isomorph-free SEM-like enumeration of models. *Electronic Notes in Theoretical Computer Science*, 125(2):91–113, 2005. Proceedings of the 5th International Workshop on Strategies in Automated Deduction (Strategies 2004). doi:10.1016/j.entcs.2005.01.003.
- 9 Stanley Burris and Hanamantagouda P. Sankappanavar. *A course in universal algebra*, volume 78 of *Graduate texts in mathematics*. Springer, New York, NY, 1981.
- 10 Geoffrey Chu, Christian Schulte, and Peter J. Stuckey. Confidence-based work stealing in parallel constraint programming. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP*, volume 5732, pages 226–241. Springer, 2009. doi:10.1007/978-3-642-04244-7\_20.
- 11 Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.
- 12 Michael Codish, Alice Miller, Patrick Prosser, and Peter James Stuckey. Breaking symmetries in graph representation. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 510–516. IJCAI/AAAI, 2013. URL: <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6480>.
- 13 James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 148–159. Morgan Kaufmann, 1996.
- 14 A. Distler and J. Mitchell. Smallsemi, a library of small semigroups in GAP, Version 0.6.12. <https://gap-packages.github.io/smallsemi/>, 2019. GAP package.
- 15 Andreas Distler, Christopher Jefferson, Tom Kelsey, and Lars Kotthoff. The semigroups of order 10. In Michela Milano, editor, *Principles and Practice of Constraint Programming - CP*, volume 7514, pages 883–899. Springer, 2012. doi:10.1007/978-3-642-33558-7\_63.
- 16 The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.11.1*, 2021. URL: <https://www.gap-system.org>.
- 17 Ian P. Gent, Christopher Jefferson, and Ian Miguel. Minion: A fast scalable constraint solver. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI, 17th European Conference on Artificial Intelligence, Including Prestigious Applications of Intelligent Systems (PAIS), Proceedings*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 98–102, Amsterdam, Netherlands, 2006. IOS Press. URL: <http://www.booksonline.iospress.nl/Content/View.aspx?piid=1654>.
- 18 Ian P. Gent, Ian Miguel, Peter Nightingale, Ciaran McCreesh, Patrick Prosser, Neil C. A. Moore, and Chris Unsworth. A review of literature on parallel constraint solving. *Theory Pract. Log. Program.*, 18(5-6):725–758, 2018. doi:10.1017/S1471068418000340.

- 19 Pascal Van Hentenryck, Pierre Flener, Justin Pearson, and Magnus Ågren. Tractable symmetry breaking for CSPs with interchangeable values. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 277–284. Morgan Kaufmann, 2003. URL: <http://ijcai.org/Proceedings/03/Papers/041.pdf>.
- 20 Marijn Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, HVC, Revised Selected Papers*, volume 7261, pages 50–65. Springer, 2011. doi:10.1007/978-3-642-34188-5\_8.
- 21 Marijn J. H. Heule, Oliver Kullmann, and Armin Biere. Cube-and-conquer for satisfiability. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 31–59. Springer, 2018. doi:10.1007/978-3-319-63516-3\_2.
- 22 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In *Theory and Applications of Satisfiability Testing (SAT)*, 2016. doi:10.1007/978-3-319-40970-2\_15.
- 23 Antti E. J. Hyvärinen, Matteo Marescotti, and Natasha Sharygina. Lookahead in partitioning SMT. In *Formal Methods in Computer Aided Design, FMCAD*, pages 271–279. IEEE, 2021. doi:10.34727/2021/isbn.978-3-85448-046-4\_37.
- 24 Xiangxue Jia and Jian Zhang. A powerful technique to eliminate isomorphism in finite model search. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 318–331, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- 25 Tommi Junttila, Matti Karpka, Petteri Kaski, and Jukka Kohonen. An adaptive prefix-assignment technique for symmetry reduction. *Journal of Symbolic Computation*, 99:21–49, 2020. doi:10.1016/j.jsc.2019.03.002.
- 26 Bernard Jurkowiak, Chu Min Li, and Gil Utard. A parallelization scheme based on work stealing for a class of SAT solvers. *J. Autom. Reason.*, 34(1):73–101, 2005. doi:10.1007/s10817-005-1970-7.
- 27 Majid Ali Khan. Efficient enumeration of higher order algebraic structures. *IEEE Access*, 8:41309–41324, 2020. doi:10.1109/ACCESS.2020.2976876.
- 28 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP*, volume 210 of *LIPICs*, pages 34:1–34:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CP.2021.34.
- 29 Lars Kotthoff and Neil C. A. Moore. Distributed solving through model splitting. *ArXiv*, abs/1008.4328, 2010.
- 30 Kenneth Kunen. The structure of conjugacy closed loops. *Transactions of the American Mathematical Society*, 352(6):2889–2911, 2000.
- 31 Arnaud Malapert, Jean-Charles Régin, and Mohamed Rezgui. Embarrassingly parallel search in constraint programming. *J. Artif. Intell. Res.*, 57:421–464, 2016. doi:10.1613/jair.5247.
- 32 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Improving search space splitting for parallel SAT solving. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI*. IEEE Computer Society, 2010. doi:10.1109/ICTAI.2010.56.
- 33 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. An overview of parallel SAT solving. *Constraints An Int. J.*, 17(3):304–347, 2012. doi:10.1007/s10601-012-9121-3.
- 34 W. McCune. Prover9 and mace4. <http://www.cs.unm.edu/~mccune/prover9/>, 2005–2010.
- 35 William McCune. Mace4 reference manual and guide, August 2003. URL: <https://www.cs.unm.edu/~mccune/prover9/mace4.pdf>.
- 36 Gábor Nagy and Petr Vojtěchovský. LOOPS, computing with quasigroups and loops in GAP, Version 3.4.1. <https://gap-packages.github.io/loops/>, November 2018. Refereed GAP package.
- 37 Morten Nielsen. Parallel search in gecode. *Technical Report, Gecode*, 2006.

- 38 Peter Nightingale, Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. Automatically improving constraint models in savile row. *Artificial Intelligence*, 251:35–61, 2017. doi:10.1016/j.artint.2017.07.001.
- 39 Anthony Palmieri, Jean-Charles Régin, and Pierre Schaus. Parallel strategies selection. *CoRR*, abs/1604.06484, 2016. arXiv:1604.06484.
- 40 Giles Reger, Martin Riener, and Martin Suda. Symmetry avoidance in MACE-style finite model finding. In Andreas Herzig and Andrei Popescu, editors, *Frontiers of Combining Systems FroCoS*, volume 11715, pages 3–21, Switzerland AG, 2019. Springer. doi:10.1007/978-3-030-29007-8\_1.
- 41 Jean-Charles Régin and Arnaud Malapert. Parallel constraint programming. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 337–379. Springer, 2018. doi:10.1007/978-3-319-63516-3\_9.
- 42 Jean-Charles Régin, Mohamed Rezgüi, and Arnaud Malapert. Embarrassingly parallel search. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 596–610, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40627-0\_45.
- 43 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. URL: <https://www.sciencedirect.com/science/bookseries/15746526/2>.
- 44 Neil J. A. Sloane and The OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2020. URL: <http://oeis.org/?language=english>.
- 45 Jon Sneyers, Peter van Weert, Tom Schrijvers, and Leslie de Koninck. As time goes by: Constraint handling rules: A survey of chr research from 1998 to 2007. *Theory and Practice of Logic Programming*, 10(1):1–47, 2010. doi:10.1017/S1471068409990123.
- 46 Bernardo Subercaseaux and Marijn Heule. Toward optimal radio colorings of hypercubes via SAT-solving. In Ruzica Piskac and Andrei Voronkov, editors, *Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 94 of *EPiC Series in Computing*, pages 386–404. EasyChair, 2023. doi:10.29007/qrmp.
- 47 Toby Walsh. Symmetry breaking constraints: Recent results. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012. URL: <http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4974>.
- 48 H. Zhang. Combinatorial designs by SAT solvers. *Handbook of Satisfiability*, pages 533–568, 2009. URL: <https://cir.nii.ac.jp/crid/1571980076163512448>.
- 49 Hantao Zhang and Jian Zhang. MACE4 and SEM: A comparison of finite model generators. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Computer Science*, pages 101–130. Springer, 2013. doi:10.1007/978-3-642-36675-8\_5.
- 50 Jian Zhang. Constructing finite algebras with FALCON. *Journal of Automated Reasoning*, 17:1–22, August 1996. doi:10.1007/BF00247667.
- 51 Jian Zhang and Hantao Zhang. SEM: a system for enumerating models. In *IJCAI*, pages 298–303, 1995. URL: <http://ijcai.org/Proceedings/95-1/Papers/039.pdf>.



# Guiding Backtrack Search by Tracking Variables During Constraint Propagation

Gilles Audemard  

CRIL, Univ. Artois & CNRS, France

Christophe Lecoutre 

CRIL, Univ. Artois & CNRS, France

Charles Prud'homme  

TASC, IMT-Atlantique, LS2N-CNRS, France

---

## Abstract

It is well-known that variable ordering heuristics play a central role in solving efficiently Constraint Satisfaction Problem (CSP) instances. From the early 80's, and during more than two decades, the dynamic variable ordering heuristic selecting the variable with the smallest domain was clearly prevailing. Then, from the mid 2000's, some adaptive heuristics have been introduced: their principle is to collect some useful information during the search process in order to take better informed decisions. Among those adaptive heuristics, *wdeg/dom* (and its variants) remains particularly robust. In this paper, we introduce an original heuristic based on the *midway* processing of failing executions of constraint propagation: this heuristic called *pick/dom* tracks the variables that are directly involved in the process of constraint propagation, when ending with a conflict. The robustness of this new heuristic is demonstrated from a large experimentation conducted with the constraint solver ACE. Interestingly enough, one can observe some complementary between the early, midway and late forms of processing of conflicts.

**2012 ACM Subject Classification** Computing methodologies → Discrete space search

**Keywords and phrases** Variable Ordering Heuristics, Variable Weighting

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.9

**Funding** This work has benefited from the support of the National Research Agency under France 2030, MAIA Project ANR-22-EXES-0009.

## 1 Introduction

Backtrack search remains a classical approach for solving instances of the Constraint Satisfaction Problem (CSP), and the related Constraint Optimization Problem (COP). It is based on depth-first exploration, which is conducted by instantiating variables in sequence and backtracking when dead-ends occur. For efficiently exploring the search space, a filtering process is performed at each step of the search so as to reduce the domains of the variables; typically most of the constraints guarantee the property known as (generalized) arc consistency [5, 1, 26, 14]. The order in which variables are chosen during the depth-first traversal of the search space is decided by a *variable ordering heuristic*  $H$ . At each internal node of the search tree built by the backtrack search algorithm, the next variable  $x$  is selected by  $H$ , and a value is assigned to  $x$  according to a *value ordering heuristic*. Choosing the right heuristics for solving a given constraint network is a key issue since different heuristics can lead to drastically different search trees.

In modern constraint solvers, three main principles are considered for guiding search (i.e., performing depth-first exploration):

- First, one should start by assigning variables that belong to the most difficult part(s) of the problem instance. This principle is derived from the recognition that there is no point in traversing the easy part(s) of an instance and then backtracking repeatedly when



© Gilles Audemard, Christophe Lecoutre, and Charles Prud'homme;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 9; pp. 9:1–9:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



it turns out that the first choices are incompatible with the remaining difficult part(s). Here the underlying *fail-first* principle is [13]: “To succeed, try first where you are most likely to fail”.

- Second, value selection should be based on the *succeed-first* or *promise* principle, which comes from the simple observation that to find a solution quickly, it is better to move at each step to the most promising subtree, primarily by selecting a value that is most likely to participate in a solution.
- Third, when starting to build the search tree, one should pay attention to the initial variable/value choices that are particularly important. Indeed, bad choices near the root of the search tree may turn out to be disastrous because they lead to exploration of very large fruitless subtrees. To make good initial choices, one strategy is to select the first branching decisions with special care, perhaps calling sophisticated and expensive procedures for this purpose. Another relevant strategy is to restart search several times, ideally learning some information each time in order to refine search guidance.

The current response (in solvers) to following these principles is as follows:

- Generic *adaptive* variable ordering heuristics that learn from conflicts during exploration are usually employed; a classical such heuristic being *wdeg/dom* [3], possibly combined with a mechanism, called last-conflict reasoning (lc) simulating a certain form of intelligent backjumps [18].
- Promising attempts to select values (or pairs variable-value) according to some elegant mechanisms such as Belief Propagation [23] have been introduced, but, unfortunately, we are not aware of any generic robust value ordering heuristic. One exception may be BIVS (Bound-Impact Value Selector) [8], but controlling its computation cost remains a difficult question. Consequently, it is rather frequent that the first (smallest) value be the default choice. Interestingly, for optimization, it is highly recommended to use in priority the value present in the last found solution, which is a technique known as solution(-based phase) saving [27, 7], clearly in concordance with the *promise* principle (as initially mentioned in [6, 10]).
- To address the issue of heavy-tailed runtime distributions [11], the search is restarted regularly, following a geometric progression (or the Luby sequence). Besides, by collecting nogoods [17] along the leftmost branch of the search tree at the end of each run (i.e., just before restarting), we have the guarantee of never exploring again the same parts of the search space (which is a nice feature when exhibiting all distinct solutions of a CSP instance).

This is the context of our contribution. More specifically, we focus our interest on the first crucial component: variable ordering heuristics. Indeed, in this paper, we introduce an original heuristic called *pick/dom* that learns from conflicts by identifying the variables that are directly involved in the process of constraint propagation (when failing). We show how to implement it in the variable-oriented propagation scheme. The robustness of this new heuristic is demonstrated by conducting a large experimentation with the well-known constraint solver ACE [15].

## 2 Preliminaries

A *Constraint Network* (CN) is composed of a finite set of  $n$  variables  $\mathcal{X}$ , and a finite set of  $e$  constraints  $\mathcal{C}$ . Each variable  $x$  must be assigned a value from its current domain, denoted by  $\text{dom}(x)$ . Each constraint  $c$  represents a mathematical relation over an ordered set of variables,

called the *scope* of  $c$ , and denoted by  $\text{scp}(c)$ . The *arity* of a constraint  $c$  is the size of its scope. The *degree* of a variable  $x$  is the number of constraints of  $\mathcal{C}$  involving  $x$ . A *solution* to a CN  $P = (\mathcal{X}, \mathcal{C})$  is the assignment of a value to each variable of  $\mathcal{X}$  such that all constraints of  $\mathcal{C}$  are satisfied. A constraint network is *satisfiable* iff it admits at least one solution. The *Constraint Satisfaction Problem (CSP)* is to determine whether a given constraint network is satisfiable, or not. A classical approach for solving this NP-complete problem is to perform a depth-first search with backtracking, while filtering domains after each taken decision. This procedure builds a binary search tree  $\mathcal{T}$ : for each internal node  $\nu$  of  $\mathcal{T}$ , a pair  $(x, v)$  is selected where  $x$  is a variable and  $v$  is a value in  $\text{dom}(x)$ . Then, two cases are considered: the assignment  $x = v$  (positive decision) and the refutation  $x \neq v$  (negative decision). The *future* variables of a constraint  $c$ , denoted by  $\text{fut}(c)$ , are the variables in  $\text{scp}(c)$  at a given node of the search tree that have not been explicitly assigned by the search algorithm.

When an objective function (integer or real-valued function defined on a subset of variables of  $\mathcal{X}$ ) is added to the constraint network, we obtain an instance of the *Constraint Optimization Problem (COP)*. Backtrack search for COP relies on an optimization strategy based on decreasingly updating the maximal bound (assuming minimization) whenever a solution is found; this is a kind of ramp-down strategy (related to Branch and Bound), whose principle is equivalent (still assuming a minimization problem) to adding a special objective constraint  $\text{obj} < \infty$  to the constraint network (although it is initially trivially satisfied), and to update the limit of this constraint whenever a new solution is found.

### 3 Variable Ordering Heuristics

We provide in this section a quick overview of popular general-purpose search heuristics. The simple variable ordering heuristic  $\text{dom}$  [13], which selects variables in sequence of increasing size of domain, has long been considered as the most robust backtrack search heuristic. However, twenty years ago, *adaptive* heuristics were introduced: they take into account information collected along the part of the search space (tree) already explored.

In this paper, we shall mainly focus our attention to the popular adaptive heuristics based on constraint weighting ( $\text{wdeg}$ ,  $\text{wdeg}/\text{dom}$ ,  $\text{cacd}$ ,  $\text{chs}$ ), and failure rating ( $\text{frba}/\text{dom}$ ). We will also refer to **impact**, **activity** and counting-based heuristics, which are defined as follows:

- **impact**, or **ibs** (Impact-Based Search), selects in priority the variable with the highest impact. The impact of a variable  $x$  gives a measure about the importance of  $x$  in reducing the search space [25]. The size of the search space of a CN  $P$  is the product of all current domain sizes:

$$\text{size}(P) = \prod_{x \in \mathcal{X}} |\text{dom}(x)|$$

The impact  $I$  of a variable assignment  $x = a$  on  $P$  is computed as follows:

$$I(x = a) = 1 - \frac{\text{size}(P')}{\text{size}(P)}$$

where  $P' = \phi(P|_{x=a})$  denotes the CN obtained after assigning  $x$  to  $a$  and running the filtering process  $\phi$  (e.g., enforcing arc consistency). Note that if  $P'$  leads to a failure, then  $I(x = a) = 1$ . It is easy to see that this heuristic can be used for value selection as well.

- **activity**, or **abs** (Activity-Based Search), selects in priority the variable with the highest activity. The activity of a variable  $x$  is roughly measured by the number of times the domain of  $x$  is reduced during search [22]. This heuristic is motivated by the key role of propagation in constraint programming and relies on a decaying sum to forget the oldest statistics progressively. More formally, the activity  $A(x)$  of a variable  $x$  is updated at

## 9:4 Guiding Backtrack Search by Tracking Variables During Constraint Propagation

each (new) node of the search tree (after a decision has been taken by the solver followed by constraint propagation) regardless of the outcome (success or failure) by the following two rules:

- $A(x) = A(x) * \gamma$ , where  $0 \leq \gamma \leq 1$  is an age decay parameter, if the domain of  $x$  has not been affected (i.e., has not been reduced)
- $A(x) = A(x) + 1$  otherwise

The activities are initialized by making random probing in the search space.

- Counting-based search relies on computing the solution density of each variable-value assignment for a constraint in order to build an integrated variable-selection and value-selection heuristic [24]. Depending on the constraints, computing such information can carry a high computational cost although some mechanisms have been proposed to accelerate it [9].

Now, to introduce `wdeg` and `wdeg/dom`, we need to describe the way constraint propagation is run each time a decision is taken by the backtrack search algorithm. Algorithm 1 describes the constraint-oriented propagation scheme, which uses of a set of constraints for piloting propagation. This simplifies the presentation here, whereas later, we will introduce the new heuristic `pick/dom` in the context of the variable-oriented scheme. Initially, the set  $Q$  contains the whole set of constraints of the constraint network. Then, each constraint  $c$  in  $Q$  is picked in turn and a filtering process is applied from  $c$ : typically, this is for enforcing arc-consistency (or a partial form) by calling `filter(c)` at Line 4. The call to this function returns a subset of variables involved in  $c$ , denoted by  $X$ , whose domains have been modified (i.e., such that at least one value has been removed from each of these domains). By means of  $X$ , we can update  $Q$  so as to ensure constraint propagation is run until a fixed point is reached. If ever the domain of one variable of  $X$  is empty, it simply means that a conflict occurred (a dead-end has been identified) and so, a backtrack is required. This is triggered by the returned Boolean value `false`, after having called the function `incrementWeight` with the culprit constraint (responsible for the domain wipeout) passed as a parameter. In the initial paper [3], the principle of constraint weighing is very simple: the weight of the culprit constraint  $c$  is incremented by 1.

■ **Algorithm 1** `propagate(( $\mathcal{X}$ ,  $\mathcal{C}$ ): CN): Boolean.`

---

```

1  $Q \leftarrow \mathcal{C}$ 
2 while  $Q \neq \emptyset$  do
3   pick and delete  $c$  from  $Q$ 
4    $X \leftarrow \text{filter}(c)$  //  $X$  are variables in  $\text{scp}(c)$  with reduced domains
5   if  $\exists x \in X \mid \text{dom}(x) = \emptyset$  then
6     incrementWeight( $c$ )
7     return false // detected inconsistency
8   foreach  $c' \in \mathcal{C} \mid c' \neq c \wedge X \cap \text{scp}(c') \neq \emptyset$  do
9      $Q \leftarrow Q \cup \{c'\}$ 
10 return true

```

---

To summarize, each constraint  $c$  admits a weight, initially set to 1, which is incremented whenever a domain wipeout occurs while filtering  $c$ . Importantly, it was observed experimentally that it was more effective to consider only the future variables involved in a culprit constraint. Technically, instead of associating a global weight `c.weight` with each constraint

$c$ , one can introduce a local weight  $c.\text{weight}[x]$  to be associated with each variable  $x$  in  $\text{scp}(c)$ . Hence, when a conflict occurs, instead of incrementing the weight  $c.\text{weight}$  of the culprit constraint, one can decide to increment the local weight  $c.\text{weight}[x]$  of each future variable involved in  $\text{scp}(c)$ .

The heuristics **wdeg** and **wdeg/dom** are defined as follows:

- **wdeg** selects in priority the future variable with the highest “weighted degree”. Each variable  $x$  is given a weighted degree, which is the sum of the weights over all constraints involving  $x$  and at least another future variable. For each future variable  $x$ , the score of  $x$  according to **wdeg** is:

$$\sum_{c \in \mathcal{C} : x \in \text{scp}(c) \wedge |\text{fut}(c)| > 1} c.\text{weight}[x]$$

- **wdeg/dom** selects in priority the future variable with the highest ratio “weighted degree to current domain size”. For each future variable  $x$ , the score of  $x$  according to **wdeg/dom** is:

$$\frac{\sum_{c \in \mathcal{C} : x \in \text{scp}(c) \wedge |\text{fut}(c)| > 1} c.\text{weight}[x]}{|\text{dom}(x)|}$$

In classical forms of **wdeg** and **wdeg/dom**, counters are incremented by 1., which remains very simplistic and does not differentiate between constraints. This is why constraint weighting, in the so-called variant **cacd** [28], has been refined by exploiting as information both the “current arity” of the culprit constraint (i.e., the number of future variables) and the size of the current domains of the future variables. The increment values computed for the classical and **cacd** variants are precisely shown in Algorithm 2.

■ **Algorithm 2** `incrementWeight(c: Constraint)`.

---

```

1 foreach  $x \in \text{fut}(c)$  do
2   if variant cacd then
3      $c.\text{weights}[x] \leftarrow c.\text{weights}[x] + \frac{1}{|\text{fut}(c)| \times (1 + |\text{dom}(x)|)}$ 
4   else
5      $c.\text{weights}[x] \leftarrow c.\text{weights}[x] + 1$ 

```

---

Note that to break ties, which correspond to sets of variables that are considered as equivalent by the heuristic, one can use a second criterion. However, for adaptive heuristics, it is usual that the first encountered variable with the best score is selected.

Finally, we introduce two recent heuristics: the former, **chs** [12], exploits the history of search failures, while the latter, **frba/dom** [20], computes the proportion of failing assignments for each variable.

- **chs** (Conflict-History Search), selects in priority variables appearing in recent failures. All failures are registered with a timestamp. More precisely, **chs** maintains for each constraint  $c$ , a score  $q(c)$  and updates it at every domain wipeout with an exponential recency weighted average:

$$q(c) = (1 - \alpha) \times q(c) + \alpha \times r(c)$$

where  $\alpha = 0.4$  (decreasing as time goes by) and  $r(c)$  is the reward given when a domain wipeout occurred. The reward is higher when the constraint frequently enters in conflict:

$$r(c) = \frac{1}{\#\text{Conflicts} - \text{Conflict}(c) + 1}$$

where  $\#Conflicts$  is the total number of conflicts and  $Conflict(c)$  stores the last value of  $\#Conflicts$  when  $c$  led to a failure. The conflict history score of a variable  $x$  which will be used in selecting the branching variable is given by:

$$\frac{\sum_{c \in \mathcal{C} : x \in \text{scp}(c) \wedge |\text{fut}(c)| > 1} q(c) + \delta}{|\text{dom}(x)|}$$

where  $\delta$  is a positive real number close to 0 that avoids random selection at the beginning of search.

- **frba/dom** (Failure Rate Based Search), selects in priority the variables that most often lead to a conflict when assigning them. For this purpose, two counters are associated with each variable  $x$ : the former,  $\#Conflicts(x)$ , records the number of times a failure has been observed just by propagating an assignment involving  $x$ , and the latter,  $\#Assigns(x)$ , the number of times the variable  $x$  was assigned. The failure rate of a variable  $x$  is then:

$$fr(x) = \frac{\#Conflicts(x)}{\#Assigns(x)}$$

In addition, similarly to the factor used for **chs**, we compute:

$$a(x) = \frac{1}{\#Conflicts - Conflict(x) + 1}$$

where  $\#Conflicts$  is the total number of conflicts and  $Conflict(x)$  stores the last value of  $\#Conflicts$  when  $x$  being assigned led to a failure.

The failure rate score of a variable  $x$  by **frba/dom** is then:

$$\frac{fr(x) + a(x)}{|\text{dom}(x)|}$$

#### 4 Variable Tracking in Conflicting Propagation

In this section, we introduce the principle of Variable Tracking in Conflicting Propagation (VTCP). More specifically, we introduce a new variable ordering heuristic called **pick/dom**, whose principle is to track the variables that are used to trigger filtering operations during constraint propagation. However, it is important to note that this tracking is only used to update counters (called “pick degrees”) associated with variables when constraint propagation ends with a conflict. We show how to implement such variable tracking within the variable-oriented propagation scheme.

To record information about tracked variables, we just need to associate a counter  $\text{pick}[x]$  with each variable  $x$  of the CN. Initially, this counter (“pick degree”) is set to 0. According to the selected mode (see below) used to update these counters, recorded values may be in real or integer forms. The heuristic **pick/dom** is simply defined as follows:

- **pick/dom** selects in priority the future variable with the highest ratio “pick degree to current domain size”. For each future variable  $x$ , the score of  $x$  according to **pick/dom** is:

$$\frac{\text{pick}[x]}{|\text{dom}(x)|}$$

In the rest of this section, we show how pick degrees are computed.

In ACE, constraint propagation follows the variable-oriented scheme (as initially introduced in [21]): the set  $Q$  contains variables. The principle is that whenever a value is removed

from the domain of a variable  $x$ , this variable is added to  $Q$ . In a first step, by ignoring any statement related to  $L$  and  $\Delta_x$ , we can rather easily recognize the variable-oriented propagation scheme in Algorithm 3: as long as there is a variable in  $Q$ , one of them,  $x$  is picked, and we execute the filtering algorithms (propagators) attached to all constraints involving  $x$ , while updating  $Q$  when necessary.

■ **Algorithm 3** `propagate(( $\mathcal{X}, \mathcal{C}$ ): CN): Boolean.`

---

```

1  $L = \langle \rangle$ 
2  $Q \leftarrow \mathcal{X}$ 
3 while  $Q \neq \emptyset$  do
4   pick and delete  $x$  from  $Q$ 
5    $\Delta_x \leftarrow 0$ 
6   for  $c \in \mathcal{C} \mid x \in \text{scp}(c)$  do
7      $X \leftarrow \text{filter}(c, \Delta_x)$  //  $\Delta_x$  is updated during call
8     if  $\exists y \in X \mid \text{dom}(y) = \emptyset$  then
9       append  $(x, \Delta_x)$  to  $L$ 
10      incrementPick( $L$ )
11      return false // detected inconsistency
12     $Q \leftarrow Q \cup X$ 
13  if  $\Delta_x > 0$  then
14    append  $(x, \Delta_x)$  to  $L$ 
15 return true

```

---

■ **Algorithm 4** `incrementPick( $L$ : Sequence).`

---

```

1 foreach  $i$  ranging from 1 to  $|L|$  do
2    $(x_i, \Delta_{x_i}) \leftarrow L[i]$ 
3   switch VARIANT do
4     case 0 do
5       increment  $\leftarrow 1$ 
6     case 1 do
7       increment  $\leftarrow \Delta_{x_i}$ 
8     case 2 do
9       increment  $\leftarrow 100 \times \frac{\Delta_{x_i}}{\sum_{j=1}^{|L|} \Delta_{x_j}}$ 
10    case 3 do
11      increment  $\leftarrow \frac{n - \text{depth}}{n} \times 100 \times \frac{\Delta_{x_i}}{\sum_{j=1}^{|L|} \Delta_{x_j}}$ 
12    pick[ $x_i$ ]  $\leftarrow \text{pick}[x_i] + \text{increment}$ 

```

---

Now, let us consider first the structure  $L$ , which is a list, initially empty, keeping track of any variable  $x$  that plays a role (i.e., triggers some effective filtering) during propagation, together with an information indicating the degree  $\Delta_x$  of this role. Notice that, since  $L$  is a list, the same variable may occur several times. Concerning the local variable  $\Delta_x$ , it is initialized to 0 in Line 5. When the loop starting at Line 6 ends,  $\Delta_x$  indicates how many

values have been deleted from the domain of  $x$  in the different calls to Function `filter` at Line 7. In practice, it is possible to handle  $\Delta_x$  in a non-intrusive way by introducing a global variable whose value is incremented whenever a value is deleted (whatever the domain is). If at Line 13, the value of  $\Delta_x$  is 0, it means that no filtering/reduction was performed at all since the time  $x$  was picked. This is then a useless “pick”, which is the reason why we do not update the structure  $L$  at Line 14. Importantly, the list  $L$  is only exploited if Algorithm 3 returns `false` (because a conflict is detected). Before returning `false`, the last picked variable is added to  $L$  (because we have the guarantee of some filtering) and the function `incrementPick` is called in order to update some picked degrees.

The way picked degrees are updated is shown in Algorithm 4. Four modes (denoted by values ranging from 0 to 3) are possible. In mode 0, the picked degree of any occurrence of a variable present in  $L$  is incremented by 1. In mode 1, the increment is given by  $\Delta_x$ , the impact of  $x$  after having been picked. In mode 2, each time constraint propagation is run, 100 points are shared according to the relative impacts of the variables present in  $L$ . In mode 3, a coefficient is applied to 100, depending on the current depth of the solver. As a first extreme case, the current depth is 0, which means that we are at the root node, and so, 100 points are spread. As a second extreme case, the current depth is  $n$ , meaning that we are a leaf, and 0 point is shared. The rationale is that we give more importance to nodes near the top of the search tree.

## 5 Experimental Results

In our experiments, we have compared general-purpose variable ordering heuristics based on constraint weighting, failure rating and variable tracking during conflicting propagation, with the constraint solver ACE [15]. More specifically, we have compared the four variants of `pick/dom` with `wdeg-cacd` [28], `wdeg/dom` [3] and `chs` [12], as well as the recently introduced `frba/dom`. From now on, for simplicity, `pick/dom`, `wdeg-cacd` and `frba/dom` will be referred as `pick` (while appending the mode in subscript text), `cacd` and `frba`, respectively. Note that `ibs` and `abs` are not retained in our experiments because they are usually outperformed when used in ACE. Concerning the value ordering heuristic, it systematically chooses the smallest value in domains.

We have considered two benchmarks, denoted by `xcsp-csp` and `xcsp-cop`, which are respectively composed of all CSP and COP instances selected for the main tracks of the XCSP<sup>3</sup> competitions (In 2019, instances were randomly selected from existing series, and there were no competitions held in 2020 and 2021) organized in 2017, 2018 [16] and 2022 [2] (most of them generated by the Python library PyCSP<sup>3</sup> [19]). They correspond to two full sets of 942 and 1,034 instances in format XCSP<sup>3</sup> [4], for exactly 77 and 50 problems, respectively. A time limit of 1,200 seconds was given per instance.

**Ranking.** Results will be partly analyzed from the scoring function used for the 2022 XCSP<sup>3</sup> competition. For self-containedness, we recall it now. The number of points won by a solver  $S$  is decided as follows:

- for CSP, this is the number of times  $S$  is able to solve an instance, i.e., to decide the satisfiability of an instance (either exhibiting a solution, or indicating that the instance is unsatisfiable)
- for COP, this is, roughly speaking, the number of times  $S$  gives the best known result, compared to its competitors. More specifically, for each instance  $I$ :

- if  $I$  is unsatisfiable, 1 point is won by  $S$  if  $S$  indicates that the instance  $I$  is unsatisfiable, 0 otherwise,
- if  $S$  provides a solution whose bound is less good than another one (found by another competing solver), 0 point is won by  $S$ ,
- if  $S$  provides an optimal solution, while indicating that it is indeed the optimality, 1 point is won by  $S$ ,
- if  $S$  provides (a solution with) the best found bound among all competitors, this being possibly shared by some other solver(s), while indicating no information about optimality: 1 point is won by  $S$  if no other solver proved that this bound was optimal, 0.5 otherwise.

## 5.1 Global Overview of Results

We start this experimental section with a global overview of the obtained results. The scores of tested heuristics on `xcsp-csp` and `xcsp-cop` are given in Table 1 and Table 2. First, let us make some comments on CSP. Here, the default version of ACE is the best one (`cacd`). The heuristics are relatively close, and the differences mainly come from the number of (solved) SAT instances. On our benchmark, the heuristic `frba` appears to be the worst one. Concerning the `pick` variants, they are quite close, even if `pick3` is the one that is most often the fastest heuristic.

■ **Table 1** Ranking on `xcsp-csp`. For each heuristic, we report the number of SAT/UNSAT instances, the total number of solved instances and the number of times a heuristic is the fastest for solving an instance. We also report the result for the virtual best solver/heuristic (VBS).

Heuristic	Solved	SAT	UNSAT	Fastest
VBS	676	481	195	676
<code>cacd</code>	<b>646</b>	<b>457</b>	<b>189</b>	411
<code>chs</code>	636	449	187	390
<code>pick<sub>3</sub></code>	632	441	<b>189</b>	<b>442</b>
<code>pick<sub>0</sub></code>	631	442	188	439
<code>pick<sub>1</sub></code>	630	441	<b>189</b>	427
<code>wdeg/dom</code>	626	442	183	377
<code>pick<sub>2</sub></code>	625	437	188	434
<code>frba</code>	618	431	187	391

For COP, all `pick` variants are the most efficient heuristics: indeed, the gap with the other heuristics is significant. The variant `pick3` is the best one in terms of the number of proved optima and the number of found best bounds. The heuristics based on constraint weighting got quite similar results, and even if the heuristic `frba` appears to be also outperformed, it is important to report that this heuristic is very good in proving optima.

In order to provide a deeper analysis of the results of our experiments, we propose, in the next section, to focus only on the following three heuristics:

- `cacd` as a robust representative of the heuristics based on constraint weighting. The results of `cacd`, `wdeg/dom` and `chs` are quite similar on COP but `cacd` appears to be the most efficient heuristic on CSP. In addition, this is the default search strategy of ACE.
- `frba` as a recent proposed heuristic based on failure rating.



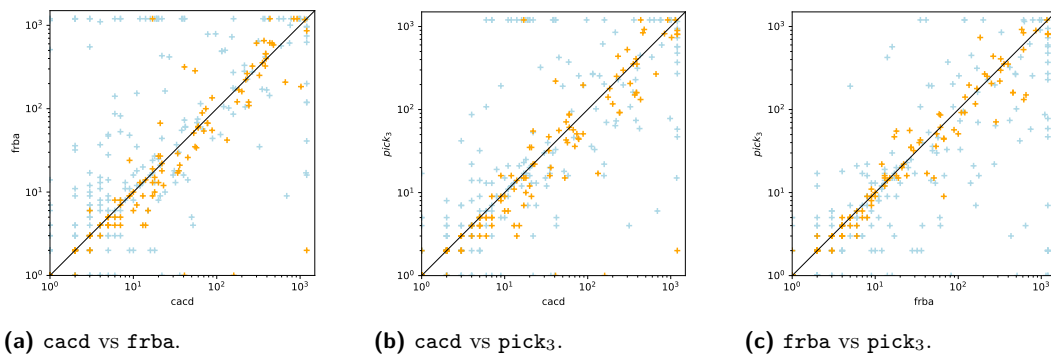
■ **Table 2** Ranking on `xcsp-cop`. For each heuristic, we report the number of proved optima, the number of times the best bound has been found and the score as computed for the 2022 XCSP<sup>3</sup> competition. We also report the results for the virtual best solver/heuristic (VBS).

Heuristic	Score	Optimum	Best Bound
VBS	959.00	372	956
<b>pick<sub>3</sub></b>	<b>624.50</b>	<b>347</b>	<b>627</b>
pick <sub>2</sub>	618.00	343	621
pick <sub>1</sub>	617.50	336	621
pick <sub>0</sub>	612.50	341	617
wdeg/dom	569.50	312	583
cacd	557.50	315	570
frba	557.00	341	560
chs	554.50	324	563

- pick<sub>3</sub> as the best variant of variable tracking in conflict propagation. Actually, it is the best heuristic on COP, able to find, most often, best bounds and also to prove them. On CSP, it is also the most efficient variant of VTCP and the fastest one.

## 5.2 Comparing Best Heuristics

In this section, we compare the three selected heuristics, namely, `cacd`, `frba` and `pick3`. We start this comparison on the `xcsp-csp` benchmark with Figure 1, which shows some classical scatter plots (permitting to compare two algorithms with rather good precision).



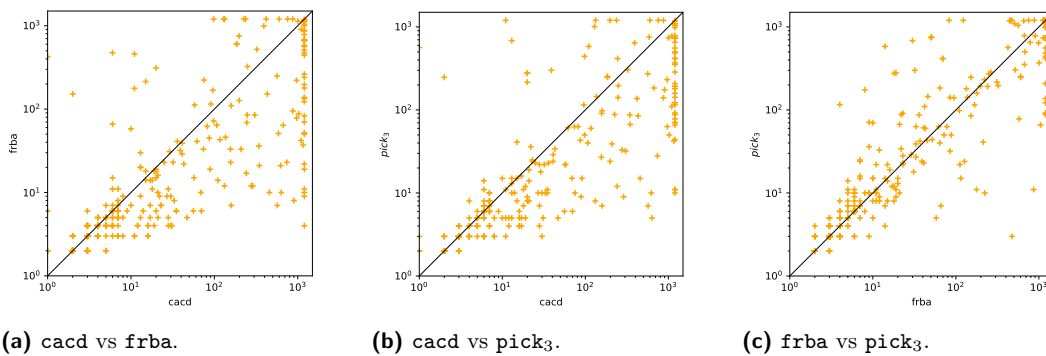
■ **Figure 1** Scatter plots for CSP instances. Each dot represents an instance: its value on the  $x$ -axis (resp.  $y$ -axis) represents the time needed by the heuristic labelling the  $x$ -axis (resp.  $y$ -axis) to solve it. Blue (resp. Orange) dots corresponds to SAT (resp. UNSAT) instances. The dots below the diagonal represent then the instances where the  $y$ -axis heuristic is faster than the  $x$ -axis one.

Here, it is clear that `frba` is less efficient than `cacd` and `pick3`. Even if `cacd` is the best heuristic on CSP (see Table 1), notably on the hardest instances, an instance-by-instance comparison between `cacd` and `pick3` looks less obvious, as the dots are uniformly distributed over both parts separated by the diagonal.

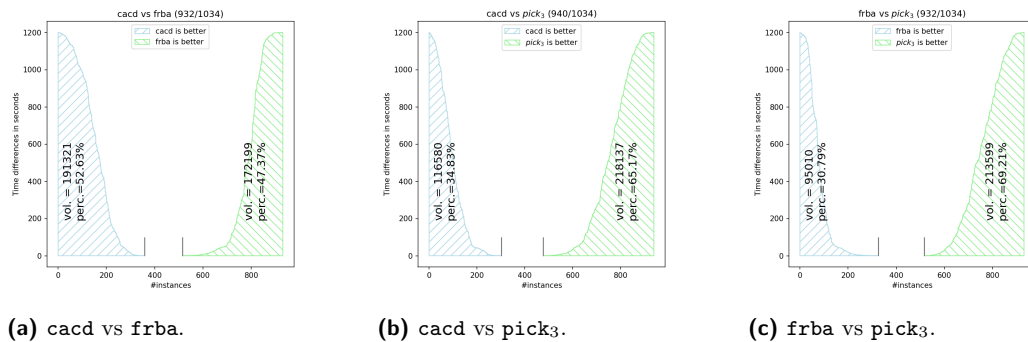
As an alternative for visualizing these results, a Venn diagram is depicted in Figure 4a. In such a diagram, each circle represents the instances solved by a heuristic. An overlapping region represents a set of instances solved in an *equivalent manner* by two heuristics, or

three in the case of the central region. For CSP, two heuristics are equivalent if they require the same amount of time (with a tolerance of one second) to find the same result (SAT or UNSAT). A region with no overlap emphasizes the instances that are better solved by a single heuristic. Here, the ranking is clear: `pick3` is the winner, followed by `cacd` and finally `frba`. To summarize the results for CSP: although `cacd` is the most robust heuristic when considering the number of solved instance (with a timeout set to 1,200 seconds), `pick3` is usually the fastest heuristic.

Next, we consider the `xcsp-cop` benchmark. A first analysis can be made from the scatter plots in Figure 2. Each plot is based only on the instances whose optimality has been proved by at least one of the two compared heuristics. On the one hand, one can see in Figure 2a and Figure 2b that `frba` and `pick3` are better at proving optimality than `cacd`. On the other hand, Figure 2c does not show any dominance between `frba` and `pick3`.



**Figure 2** Scatter plots for COP instances. Each dot represents an instance: its value on the  $x$ -axis (resp.  $y$ -axis) represents the time needed by the heuristic labelling the  $x$ -axis (resp.  $y$ -axis) to prove its optimum. The dots below the diagonal represent then the instances where the  $y$ -axis heuristic is faster than the  $x$ -axis one.

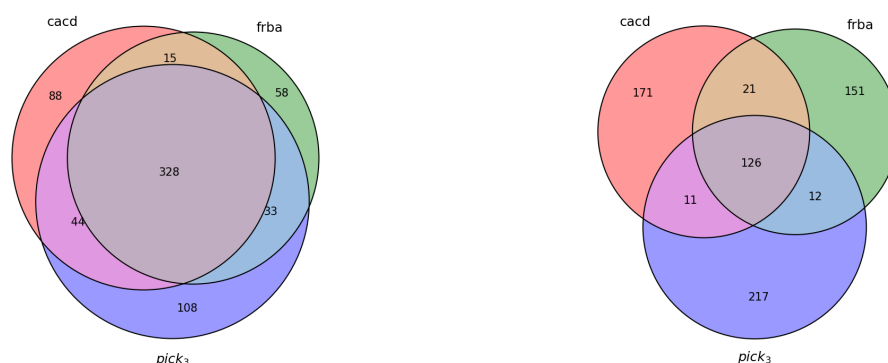


**Figure 3** Plots for COP Instances. Above each figure, the number of instances (partially) solved by at least one of the two heuristics as well as the total number of instances is indicated. When a heuristic is better than the other, this contributes to its area, representing the difference in resolution time. The *vol.* value indicates the volume of this surface, the *perc.* value indicates the ratio of the volume to the sum of the two volumes. The difference between two heuristics takes into account the best bounds found and possibly the proof of optimality.

Figure 3 allows a complementary analysis. Each graphics shows a pairwise comparison with two areas. When a heuristic is better than the other, the difference in solving time is

## 9:12 Guiding Backtrack Search by Tracking Variables During Constraint Propagation

computed. An heuristic  $a$  is better than a heuristic  $b$  if it proves optimality whereas  $b$  does not, or if it finds a better bound than  $b$ , or, finally, if they find the same bound and it is faster than  $b$ . In the two first cases, the resolution time of  $b$  is set to the time limit, namely 1,200. Each computed time difference contributes to the area of the best heuristic. Hence, a larger area means that the attached heuristic offers better performances than the other one. Each area is annotated with *vol.* and *perc.* which respectively denote the volume of the area and the ratio, in percentage, of the volume to the sum of the two volumes. Finally, strictly equivalent instances are indicated between the 2 vertical bars. In Figure 3a, one can observe that *cacd* is slightly more robust than *frba*. Interestingly, in Figure 3b and Figure 3c, the competitiveness of *pick<sub>3</sub>* is clearly visible: the area of *pick<sub>3</sub>* is almost twice as large as that of *cacd* or *frba*.



(a) Venn Diagram on CSP Instances.

(b) Venn Diagram on COP Instances.

■ **Figure 4** Venn diagrams for *cacd*, *frba* and *pick<sub>3</sub>* on CSP and COP instances. Each circle represents the instances solved by a heuristic. An overlapping region represents a set of instances solved in an equivalent manner by two heuristics, or three in the case of the central region. Two heuristics are considered as being equivalent if they found the same result in the same amount of time. For CSP, this is the time for proving (un)satisfiability. For COP, this is for getting the best bound or proving optimality. A region not overlapped emphasizes the instances that are better solved by a single heuristic.

These results are confirmed by the Venn diagram in Figure 4b. The circles, and especially the one for *pick<sub>3</sub>*, stand out from the centre. This Venn diagram also suggests that each heuristic would be more suitable for certain problems. This is confirmed in Table 3. First, *frba* performs particularly well on five problems, namely Cutstock, DC, ItemsetMining, OpenStacks and TravelingSalesman. Indeed, it is able to close more instances and provides better bounds than *cacd* and *pick<sub>3</sub>*. In turn, *cacd* behaves better on six problems: CyclicBandwith, Ramsey, StillLife and Taillard, and to a lesser extent PseudoBoolean and QueenAttacking. As for *pick<sub>3</sub>*, the set of problems where it dominates is clearly larger: Auction, BinPacking, ChessBoardColoration, CoinsGrid, EchelonStock2, FAPP, GraphColoring, MultiAgentPathFinding, NurseRostering, OPD, QuadraticAssignment, RCPSP, RLFAP, Spot, SteelMillSlab, SumColoring, TAL, TravelingTournament, Triangular and Warehouse. This is just under half of the problems. For some problems the gap is quite large, namely BinPacking, FAPP or RCPSP.

Finally, we show in Table 4 some details for some chosen (representative and singular)

instances of some problems. This may be helpful for testing and reproducing the results we have obtained.

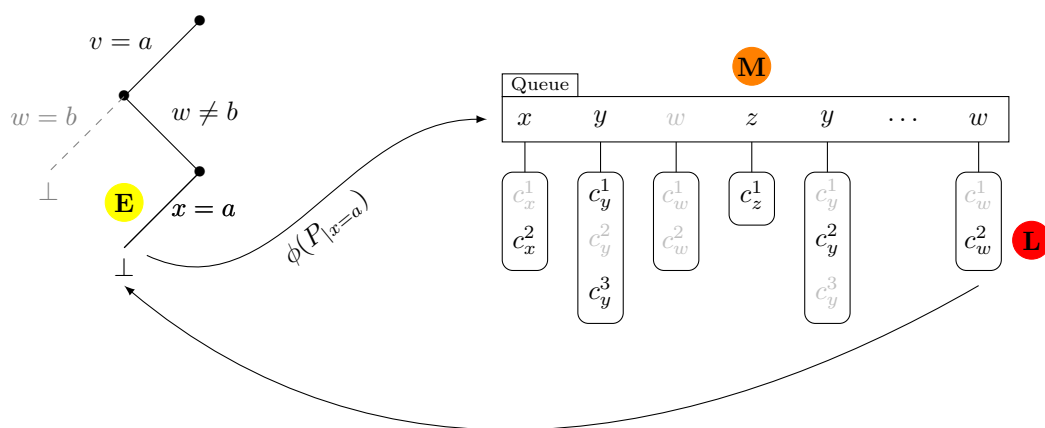
■ **Table 3** Details per problem (COP). For each problem, the number of instances is displayed, and for each heuristic, the couple 'number of proved optima : number of best bounds found' is provided. Best results are highlighted (when at least one heuristic is outperformed by the other(s)).

Problem	caed	frba	pick <sub>3</sub>	Problem	caed	frba	pick <sub>3</sub>
AirCraft (13)	<b>4:10</b>	5:6	<b>4:10</b>	NursingWork (12)	1:2	1:4	1:4
Auction (16)	2:6	2:4	<b>2:14</b>	OPD (17)	8:9	<b>9:14</b>	<b>9:15</b>
BACP (24)	4:24	4:24	4:24	OpenStacks (16)	12:16	<b>14:16</b>	12:16
BinPacking (51)	1:17	0:25	<b>2:33</b>	PeacableA (14)	4:9	<b>4:9</b>	4:8
BusScheduling (10)	1:5	1:5	1:8	PizzaVoucher (10)	4:7	<b>6:8</b>	<b>5:10</b>
CVRP (10)	0:2	0:2	0:8	PseudoB (30)	<b>13:23</b>	13:18	13:22
ChessBoard (17)	<b>3:13</b>	2:13	<b>3:14</b>	Quadratic (36)	6:26	6:14	<b>6:27</b>
ClockTriplet (10)	<b>2:9</b>	2:6	<b>2:9</b>	QueenAtt (17)	7:11	5:3	<b>8:6</b>
CoinsGrid (10)	2:7	2:4	<b>3:9</b>	Rack (4)	2:4	<b>3:4</b>	<b>3:4</b>
CrosswordDes (13)	<b>3:5</b>	3:3	<b>3:5</b>	Ramsey (17)	<b>5:17</b>	4:16	<b>5:14</b>
CutStock (17)	6:9	<b>8:9</b>	7:9	RCPSP (43)	31:34	34:35	<b>35:43</b>
CyclicBand (12)	<b>4:9</b>	2:3	2:6	RLFAP (50)	10:22	<b>12:32</b>	<b>12:48</b>
DC (26)	7:14	<b>10:25</b>	<b>12:18</b>	Spot (10)	2:7	2:6	<b>2:8</b>
EchelonStock2 (10)	0:9	0:7	<b>0:10</b>	SteelMill (17)	2:2	2:2	<b>2:7</b>
FAPP (18)	2:2	2:7	<b>3:13</b>	StillLife (47)	<b>15:46</b>	14:21	13:29
FastFood (17)	17:17	17:17	17:17	SumColoring (14)	4:8	4:7	<b>4:10</b>
Filters (8)	8:8	8:8	8:8	Taillard (51)	5:17	5:6	5:13
GolombRuler (47)	<b>19:31</b>	18:27	<b>18:35</b>	TAL (20)	9:16	<b>10:15</b>	<b>10:18</b>
GraphCol (28)	<b>14:24</b>	15:23	<b>17:24</b>	TemplateDes (15)	<b>12:13</b>	<b>12:13</b>	11:12
ItemsetMining (15)	3:10	<b>7:12</b>	6:10	TravelingTour (14)	2:7	2:2	<b>2:10</b>
Knapsack (31)	22:31	<b>26:31</b>	<b>26:31</b>	TravelingSale (29)	3:16	<b>3:17</b>	3:15
LowAuto (31)	12:21	<b>12:23</b>	<b>14:19</b>	Triangular (10)	0:3	1:5	<b>1:9</b>
Mario (10)	10:10	10:10	10:10	VRP (17)	0:9	0:1	<b>0:9</b>
MultiAgentP (20)	4:6	8:9	<b>9:17</b>	WarOrPeace (10)	6:10	6:10	6:10
NurseRost (41)	2:16	<b>3:3</b>	<b>3:29</b>	WareHouse (9)	0:3	0:0	<b>0:9</b>

## 6 Discussion

The three different ways of exploiting conflicts for guiding search, as experimented in the last section, show somewhat complementary behaviors. This can be explained by the fact that information is extracted at different moments: at the very beginning of the process conducting to a conflict (i.e., at the time of the decision), during constraint propagation, or at the time the last propagator (filtering algorithm) is solicited. One can then refer to such approaches as *early* (E), *midway* (M) and *late* (L) operational treatment of conflicts. This is illustrated in Figure 5 where a new decision  $x = a$  is taken, when solving a CN  $P$ , in the continuity of two previously taken decisions  $v = a$  and  $w \neq b$ . In our scenario, running constraint propagation  $\phi$  on (the current state of)  $P$  after having assigned the value  $a$  to  $x$ , i.e.,  $\phi(P|x = a)$ , reveals a new conflicting (dead-end) situation (denoted by  $\perp$ ). The early processing of this new conflict consists in considering the variable  $x$  involved in the decision as the main culprit. This is the principle behind the heuristic `frba/dom`. The midway processing of this conflict consists in considering all variables having played a role (i.e., having been picked) during propagation as having contributed to the failure. This is the principle underlying the heuristic `pick/dom`. The late processing of this conflict consists in considering the last constraint (here,  $c_w^2$ ) provoking a domain-wipeout (i.e., removing the last value of a domain) as the object of interest. This is the principle of constraint weighting, as in `wdeg/dom`.

One related heuristic, which is based on some form of midway strategy, is `abs`. However, `VTCP` and `abs` collect information according to different strategies. `abs` updates activity



■ **Figure 5** Illustration of pivotal moments for collecting information about conflicts: this correspond to early (E), midway (M) and late (L) processing of conflicts.

counters at each node, whereas VTCP only considers conflicting nodes. **abs** systematically updates the activity counters of each variable, whereas VTCP only increments the counters of variables at the origin of calls to effective propagators.

Importantly, we do believe that the experimental results we have obtained are significant, for several reasons. First, they can be reproduced in an open-source constraint solver (with a repository available in GitHub). Second, the number of models and instances used for our experiments is very large, involving more than 120 problems of various nature. Third, ACE is a competitive constraint solver and (although not officially engaged) showed good performances in (notably the COP main track) of the 2022 XCSP<sup>3</sup> competition.

Finally, it is true that the importance of variable tracking in conflicting propagation looks more limited when solving CSP instances. Actually, solving a (satisfiable) CSP instance involves one single phase: finding a solution, whereas solving a COP instance involves two subsequent phases: moving down towards an optimal solution, and proving optimality. We believe that **pick/dom** is rather efficient for the first phase of COP solving (for the second phase, a learning mechanism like in Picat [29] or OR-Tools becomes central).

## 7 Conclusion

In this paper, we have introduced a new way of exploiting conflicts during backtrack search so as to build a well-informed variable ordering heuristic. In contrast to existing heuristics relying on the early and late treatments of failing nodes, this new heuristic, **pick/dom**, consists in a midway processing of conflicts by tracking variables during constraint propagation. The robustness of **pick/dom** has been demonstrated from a vast experimentation campaign involving more than 120 problems (and around 2,000 instances). Interestingly, the three different forms of exploiting conflicts, based on different moments when to collect information, entail somewhat complementary behaviours of the solver. This opens some perspectives for building a still more robust solver by combining these conflict-based heuristics in a clever way. Identifying features or properties of problem instances (e.g., tree width, backbone size, presence of strong communities, structure of variable arrays, etc.) which are favorable for a certain form of conflict processing is an issue which would deserve to be addressed.

■ **Table 4** Details per instance. For each selected instance and each heuristic, the best bound, the time (in seconds) to find it (TB) and the time (in seconds) to prove the optimum (TO, if proven) are provided. Best results are highlighted. DC-A, DC-B, DC-C and DC-D stands for DC-midori-xor4-d1-t0-r05-v128, DC-midori-xor4-d1-t0-r08-v128, DC-skinny-xor0-d0-t0-r09-v64-z0, and DC-midori-xor4-d1-t0-r09-v128, respectively. Some other names have been shortened for more visibility.

Instance	cacd			frba			pick3		
	Bound	TB	TO	Bound	TB	TO	Bound	TB	TO
DC-A	<b>5</b>	733	739	<b>5</b>	9	<b>23</b>	5	21	28
DC-B	41	59	-	<b>8</b>	174	-	<b>8</b>	316	<b>484</b>
DC-C	<b>41</b>	5	-	<b>41</b>	9	-	<b>41</b>	3	<b>1,087</b>
DC-D	53	753	-	<b>9</b>	699	-	43	179	-
Fapp-m2s-21-0500	180,499K	1,195	-	<b>63,832K</b>	870	<b>881</b>	177,187K	1,008	-
Fapp-m2s-02-0250	221,762K	1,191	-	<b>221,520K</b>	1,158	-	221,591K	1,197	-
Fapp-m2s-test03-0400	453,213K	1,199	-	<b>449,289K</b>	1,192	-	450,098K	1,200	-
QueenAttacking-09	<b>0</b>	94	<b>98</b>	1	119	-	<b>0</b>	439	442
QueenAttacking-11	2	245	-	-	-	-	<b>1</b>	708	<b>736</b>
QueenAttacking-13	<b>11</b>	<b>158</b>	-	-	-	-	-	-	-
StillLife-11-14	<b>81</b>	264	-	79	652	-	<b>81</b>	625	-
StillLife-wastage-12	<b>76</b>	12	<b>872</b>	<b>76</b>	23	-	<b>76</b>	549	-
StillLife-wastage-37	<b>681</b>	1,189	-	619	1,182	-	585	398	-
Auction-cnt-d100	829K	200	-	825K	1,099	-	<b>849K</b>	1,197	-
Auction-sum-d100	840K	69	-	824K	77	-	<b>854K</b>	102	-
Auction-sum-d500	<b>3,368K</b>	186	-	3,264K	224	-	3,344K	89	-
CVRP-A-n32-k5	1,095	15	-	1,006	257	-	<b>835</b>	492	-
CVRP-A-n36-k5	1,050	77	-	1,039	1,112	-	<b>892</b>	74	-
CVRP-A-n34-k5	-	-	-	915	57	-	<b>813</b>	385	-
NurseRostering-01	<b>607</b>	1	-	<b>607</b>	1	11	<b>607</b>	1	<b>10</b>
NurseRostering-02	1,024	5	-	928	62	-	<b>833</b>	383	-
NurseRostering-19	68,621	1,200	-	-	-	-	<b>60,805</b>	1,197	-
Rlfap-graph-05-opt	2,882	883	-	<b>221</b>	2	<b>10</b>	<b>221</b>	62	<b>70</b>
Rlfap-graph-06-opt	46,647	1,198	-	34,004	1,198	-	<b>8,882</b>	769	-
Rlfap-scen-06-opt	11,211	1,166	-	10,102	491	-	<b>3,389</b>	124	-
Triangular-10	<b>20</b>	89	-	<b>20</b>	4	<b>138</b>	<b>20</b>	0	142
Triangular-22	50	46	-	50	161	-	<b>52</b>	167	-
Triangular-38	95	10	-	<b>97</b>	224	-	<b>97</b>	337	-
SteelMillSlab-m2s-3-0	68	1,059	-	-	-	-	<b>43</b>	995	-
SteelMillSlab-m2s-3-2	306	794	-	-	-	-	<b>171</b>	1,177	-
SteelMillSlab-m2s-4-0	161	1,109	-	-	-	-	<b>94</b>	1,186	-

## References

- 1 K.R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
- 2 G. Audemard, C. Lecoutre, and E. Lonca. Proceedings of the 2022 XCSP3 competition. *CoRR*, abs/2209.00917, 2022. doi:10.48550/arXiv.2209.00917.
- 3 F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.
- 4 F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3: an integrated format for benchmarking combinatorial constrained problems. *CoRR*, abs/1611.03398, 2016. URL: <http://arxiv.org/abs/1611.03398>, arXiv:1611.03398.
- 5 R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.
- 6 R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.
- 7 E. Demirovic, G. Chu, and P. Stuckey. Solution-based phase saving for CP: A value-selection heuristic to simulate local search behavior in complete solvers. In *Proceedings of CP'18*, pages 99–108, 2018.
- 8 J.-G. Fages and C. Prud'homme. Making the first solution good! In *Proceedings of ICTAI'17*, pages 1073–1077, 2017.
- 9 S. Gagnon and G. Pesant. Accelerating counting-based search. In *Proceedings of CPAIOR'18*, pages 245–253, 2018.
- 10 P.A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of ECAI'92*, pages 31–35, 1992.
- 11 C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.
- 12 D. Habet and C. Terrioux. Conflict history based heuristic for constraint satisfaction problem solving. *Journal of Heuristics*, 27(6):951–990, 2021.
- 13 R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
- 14 C. Lecoutre. *Constraint networks: techniques and algorithms*. ISTE/Wiley, 2009.
- 15 C. Lecoutre. ACE, a generic constraint solver. *CoRR*, abs/2302.05405, 2023. doi:10.48550/arXiv.2302.05405.
- 16 C. Lecoutre and O. Roussel. Proceedings of the 2018 XCSP3 competition. *CoRR*, abs/1901.01830, 2019. arXiv:1901.01830.
- 17 C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1:147–167, 2007.
- 18 C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18):1592–1614, 2009.
- 19 C. Lecoutre and N. Szczepanski. PyCSP3: modeling combinatorial constrained problems in Python. *CoRR*, abs/2009.00326, 2020. arXiv:2009.00326.
- 20 H. Li, M. Yin, and Z. Li. Failure based variable ordering heuristics for solving CSPs. In *Proceedings of CP'21*, 2021.
- 21 J.J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19:229–250, 1979.
- 22 L. Michel and P. Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Proceedings of CPAIOR'12*, pages 228–243, 2012.
- 23 G. Pesant. From support propagation to belief propagation in constraint programming. *Journal of Artificial Intelligence Research*, 66:123–150, 2019.
- 24 G. Pesant, C.-G. Quimper, and A. Zanarini. Counting-based search: Branching heuristics for constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 43:173–210, 2012.
- 25 P. Refalo. Impact-based search strategies for constraint programming. In *Proceedings of CP'04*, pages 557–571, 2004.
- 26 F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

- 27 J. Vion and S. Piechowiak. Une simple heuristique pour rapprocher DFS et LNS pour les COP. In *Proceedings of JFPC'17*, pages 39–45, 2017.
- 28 H. Watez, C. Lecoutre, A. Paparrizou, and S. Tabary. Refining constraint weighting. In *Proceedings of ICTAI'19*, pages 71–77, 2019.
- 29 N.-F. Zhou. An XCSP3 Solver in Picat. In *XCSP3 Competition 2022 Proceedings*, 2022.





# Incremental Constrained Clustering by Minimal Weighted Modification

Aymeric Beauchamp ✉ 

University of Orléans, INSA Centre Val de Loire, LIFO EA 4022, France

Thi-Bich-Hanh Dao ✉ 

University of Orléans, INSA Centre Val de Loire, LIFO EA 4022, France

Samir Loudni ✉ 

TASC (LS2N-CNRS), IMT Atlantique, France  
GREYC, University of Caen Normandy, France

Christel Vrain ✉ 

University of Orléans, INSA Centre Val de Loire, LIFO EA 4022, France

---

## Abstract

Clustering is a well-known task in Data Mining that aims at grouping data instances according to their similarity. It is an exploratory and unsupervised task whose results depend on many parameters, often requiring the expert to iterate several times before satisfaction. Constrained clustering has been introduced for better modeling the expectations of the expert. Nevertheless constrained clustering is not yet sufficient since it usually requires the constraints to be given before the clustering process. In this paper we address a more general problem that aims at modeling the exploratory clustering process, through a sequence of clustering modifications where expert constraints are added on the fly. We present an incremental constrained clustering framework integrating active query strategies and a Constraint Programming model to fit the expert expectations while preserving the stability of the partition, so that the expert can understand the process and apprehend its impact. Our model supports instance and group-level constraints, which can be relaxed. Experiments on reference datasets and a case study related to the analysis of satellite image time series show the relevance of our framework.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Computing methodologies → Semi-supervised learning settings

**Keywords and phrases** Incremental constrained clustering, Constrained optimization problem, User feedback

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.10

**Related Version** *Extended Version*: hal-04158825

**Supplementary Material** *Software (Source Code)*: <https://github.com/aymericb213/IAC>  
archived at `swh:1:dir:8f611496af5ad016912ca3d55379b50b36541f89`

**Funding** This work was supported by the French national research project HERELLES under grant agreement ANR-20-CE23-0022.

**Acknowledgements** The authors want to thank the anonymous reviewers for their comments and suggestions which helped to improve this paper.

## 1 Introduction

Clustering is a popular task in Data Mining in which data instances (i.e. points of a dataset) are grouped into distinct clusters according to their similarity. Over time, many strategies have been explored to compute data partitions, each of them having their own strengths and biases. Constrained clustering [16] aims to find relevant clusters by stating some desired properties in the form of constraints, thus alleviating the aforementioned biases. The most



© Aymeric Beauchamp, Thi-Bich-Hanh Dao, Samir Loudni, and Christel Vrain;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 10; pp. 10:1–10:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

commonly used constraints for clustering state that two points must be clustered together (*must-link*) or apart (*cannot-link*) [32]. Other constraints exist as for instance limiting the size or the diameter of the clusters [1]. They are usually given by a subject matter expert (SME) that possesses domain knowledge on the data; they model his/her knowledge and expectations about the result by expert constraints.

In practice, it may be difficult for an expert to express constraints solely on the basis of the data. It is usually easier to him/her to provide feedback on the current partition to refine it. This leads to a *human-in-the-loop* clustering process, where new constraints given or validated by the expert are incrementally integrated, modifying the result until user satisfaction. However, this raises several non trivial questions such as how to elicit the constraints, how to integrate them, or how to further exploit them. By incorporating human feedback and domain knowledge, the resulting clusters are more likely to align with the expert expectations while making them more intuitive and interpretable. This incremental setting mimics the natural step-by-step progression of an exploratory task such as clustering, where the user needs to iterate several times before satisfaction. In such a process, the result at each step should (1) be computed fast enough, (2) exploit the expert constraints efficiently (3) be similar to the result of the previous step, in order not to disturb the expert.

A naive way to integrate new expert constraints is to restart a constrained clustering algorithm. However, this presents at least two weaknesses: it starts from scratch without ever considering intermediate results and the new constraints can lead to a partition very different from the one previously shown to the expert. To cope with this problem, we propose a first generic framework that allows for truly interactive and iterative constrained clustering that fulfills the conditions mentioned above. Our main contributions are :

- an incremental constrained clustering framework designed for human interaction, combining active constraint selection and clustering modification;
- a new constraint programming model for minimal clustering modification, which ensures the stability of the partition.

The paper is organized as follows. We review in Section 2 related work on incremental constrained clustering and minimal clustering modification and present our method in Section 3. Experiments on reference and satellite image time series datasets are presented in Section 4 and perspectives are discussed in Section 5.

## 2 Related Work

The first works on using constraints in clustering are extensions of classic algorithms for handling *must-link/cannot-link* constraints [33, 10, 35]. They either search for a solution satisfying all the constraints [33] or a compromise between constraint satisfaction and clustering quality [6]. Thereafter methods allowing to integrate more general constraints, using declarative frameworks such as SAT [12], ILP [29] or CP [9] have been proposed. When new constraints are given by the user, all these methods require to restart from scratch, without any guarantee that the new partition will be similar to the previous one. Therefore they are not suited for an incremental setting.

There is a growing body of works related to incremental constrained clustering. In [7], the idea of gathering feedback from an existing result rather than expecting the user to provide insightful constraints by themselves is demonstrated. Later, the authors of [11] studied the problem of adding or removing a constraint from a constraint set satisfied by a partition. They described conditions under which the problem is easy to solve and an algorithm working under these conditions. In [26], a cluster refinement framework uses subclustering to find

representatives to present to the user before learning an embedding using the feedback. More recently, [21] proposed an incremental variant of a collaborative constrained clustering system that integrates constraint satisfaction in its objective function. Among declarative approaches, an ILP model for minimal clustering modification (MCM) is proposed in [20]. It constrains cluster diameters with the objective of removing undesirable properties from a partition. Another ILP model [28] computes a membership score for each point to each cluster and optimizes a criterion based on this score. It can modify the assignment of points to clusters, even if they are not involved in constraints. However those models are restrictive since they cannot handle conflicting constraints, which make the problem unsolvable.

Our approach seeks to preserve the general cluster structure of an existing partition while satisfying new constraints. To the best of our knowledge, this is the first approach explicitly tackling both quality and stability. It is based on CP, therefore it can integrate different kinds of constraints that can be either soft or hard, with some control over constraint relaxation and the ability to handle conflicting constraints. We use subclustering in a similar way to [26], albeit for a different purpose as it allows generalizing the changes decided by our CP model as well as finding cluster representatives used in our objective function.

### 3 Incremental and Active Clustering Framework

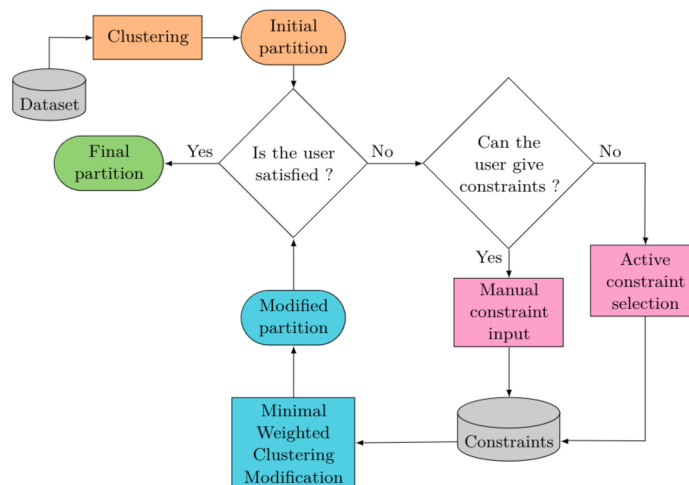


Figure 1 Schematic view of the incremental clustering cycle.

In this section, we describe our proposed incremental and active clustering (IAC) framework. Figure 1 gives a general overview of IAC. The incremental constrained clustering loop starts from an initial partition computed by any clustering algorithm. This partition is then shown to the user to collect his/her general feedback (satisfied or not). If he/she is not satisfied, he/she can modify this partition in two ways: by manually providing a set of constraints and/or by inferring it through an active constraint selection method. He/she can also set the proportion of constraints to satisfy as well as the scope of modifications. The clustering modification step updates the current partition according to these new constraints, optionally generalizing modifications to unconstrained data instances. The output is a new partition satisfying the constraints while preserving its stability, i.e. similar to the previous one. This process is repeated until the user is satisfied by the resulting partition. It is noteworthy that our framework is generic as any active constraint selection method and

any modification algorithm could be used as long as the constraints generated match the constraints handled in modification. We formulate the problem of clustering modification in a declarative way and present a CP model. This has the benefit of being able to integrate several types of constraints for the user feedback.

### 3.1 Minimal Weighted Clustering Modification

We consider the minimal weighted clustering modification (MWCM) problem: given a partition  $\mathcal{P}$  of  $N$  data instances (numbered from 1 to  $N$ ) into a number  $K$  of clusters, and a set of user constraints  $\mathcal{C}$ , the objective is to find a new partition  $\mathcal{P}'$  such that the constraints are satisfied while minimizing some function  $f$  modeling the difference between  $\mathcal{P}$  and  $\mathcal{P}'$ . For solving this problem, Algorithm 1 shows the different steps that are detailed below.

■ **Algorithm 1** Minimal Weighted Clustering Modification.

---

**Input:** Dataset  $\mathcal{X}$ , partition  $\mathcal{P}$ , constraints  $\mathcal{C}$ , anchor generation rate  $\alpha$ , super-instance rate  $\beta$ , constraint satisfaction rate  $\delta$

**Output:** modified partition  $\mathcal{P}'$

- 1:  $anchors \leftarrow \text{COMPUTEREPRESENTATIVES}(\mathcal{X}, \mathcal{P}, \alpha)$  ▷ See section 3.1.1
  - 2:  $X \leftarrow \text{COMPUTECOPINSTANCES}(\mathcal{X}, \mathcal{P}, \mathcal{C}, \beta)$  ▷ Instances used in COP (Section 3.1.2)
  - 3:  $\mathcal{D} \leftarrow \text{DISTANCEMATRIX}(X, anchors)$  ▷ See section 3.1.1
  - 4:  $p \leftarrow \text{GETCONSTRAINEDPARTITION}(X, \mathcal{P})$  ▷ Cluster membership of the constrained instances
  - 5:  $mods \leftarrow \text{SOLVEMODEL}(\mathcal{D}, p, \mathcal{C}, \delta)$  ▷ Solves the COP in Section 3.2
  - 6: **return**  $\text{APPLYMODIFICATIONS}(mods, \mathcal{P})$  ▷ Updates  $\mathcal{P}$  and generalizes modifications
- 

#### 3.1.1 Objective function and anchors

A straightforward candidate for  $f$  is to count the number of instances that have changed their cluster membership between  $\mathcal{P}$  and  $\mathcal{P}'$  [20]:

$$\arg \min \sum_{i=1}^N \mathbb{I}(\mathcal{P}[i] \neq \mathcal{P}'[i]) \quad (1)$$

where  $\mathcal{P}[i]$  denotes the number  $c \in [1, K]^1$  of the cluster containing instance  $i \in [1, N]$  and  $\mathbb{I}$  the indicator function that returns 1 if the expression given as argument is true, and 0 otherwise. The main drawback of Equation (1) is that it does not take into account the structure of the clusters. For example, it is reasonable to consider that putting two instances in a nearby cluster is more akin to the idea of minimal modification than putting one instance into a faraway cluster. Therefore we propose an alternate objective function that integrates a distance-based weighting of the modifications:

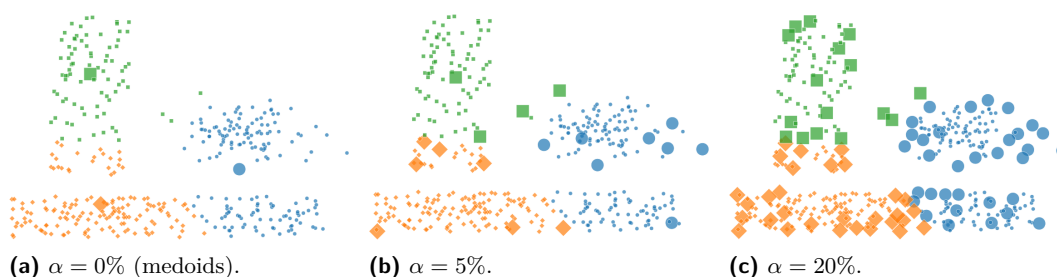
$$\arg \min \sum_{i=1}^N \mathbb{I}(\mathcal{P}[i] \neq \mathcal{P}'[i]) \mathcal{D}[i, \mathcal{P}'[i]] \quad (2)$$

where  $\mathcal{D}$  is a distance matrix of dimensions  $N \times K$  such that  $\mathcal{D}[i, c]$  is the distance of instance  $i$  to cluster  $c$ . This objective function measures the changes between  $\mathcal{P}$  and  $\mathcal{P}'$  and keeps the cumulative distances resulting from these changes small. This objective function is integrated into the model for MWCM that will be presented in Section 3.2.

---

<sup>1</sup> We use the notation  $[1, K]$  for the set  $\{1, \dots, K\}$ .

A simple way to compute  $\mathcal{D}[i, c]$  is to set  $\mathcal{D}[i, c] = d(i, \mu_c)$ , where  $\mu_c$  is the representative (medoid or centroid) of  $c$  and  $d$  is a distance measure, typically the Euclidean distance. This has however a known limitation: the modifications made by the model implicitly treat all clusters as spherical. It can be counterproductive if the user seeks more complex shaped clusters. To overcome this, we use anchors [17] such that each cluster will be represented by a subset of its instances, in order to better represent its structure. Anchors are computed by dividing each cluster  $c$  of the input partition into smaller sub-clusters using Single-Link hierarchical clustering. For each of these sub-clusters, an anchor is defined as the instance minimizing the sum of its distances with the other instances of the sub-cluster. Using anchors,  $\mathcal{D}[i, c]$  represents the distance of instance  $i$  to its closest anchor belonging to cluster  $c$ . A parameter  $\alpha$  defining the proportion of instances per cluster (in percentage) that will become anchors is used. For example, a rate of 0% means that we only use medoids, while at a rate of 100%, all instances are anchors (cf. Figure 2).



■ **Figure 2** Anchor positions for different values of  $\alpha$  computed from the partition generated by Kmeans on `1sun` dataset. The anchors are represented bigger than normal instances.

### 3.1.2 Generalizing constraints with super-instances

In a real use case, we assume that the expert will only react on a small number of instances per iteration. As a result, the clustering modification could become unnoticeable when the dataset size is large compared to the number of constraints, which is a fairly common case. Hence, exploiting expert feedback to *generalize* the modifications to relevant unconstrained instances is an important issue w.r.t. the challenge of asking a reasonable number of queries. This issue has been highlighted in [34], and is especially relevant in our incremental setting. Furthermore, this generalization must be controlled to ensure the expert can grasp the scope of potential modifications. We assume that, in most cases, the expert will want to modify a zone around the selected instances and not only the instances themselves. Bearing this intuition in mind, making use of nearest neighbors or a proximity radius seem adequate, but these methods lack predictability: if an instance is within the radius of two constrained points who were reassigned to different clusters, determining how to resolve the generalization is not obvious.

We propose to use super-instances, i.e. virtual instances grouping several real data points, to generalize the modifications. Note that *generalization* does not mean that new constraints are generated, rather that the super-instances are passed directly to the model instead of the data instances (see Appendix C). Thus any modification in the cluster membership of a super-instance amounts to changing the membership of every real data instance that compose it. The generalization scope is controlled as follows: *the less a cluster is divided, the stronger the impact of a modification*. Thus the scope depends on the number of super-instances, determined by a rate  $\beta$  proportional to the cluster size. As such, setting  $\beta$  to e.g. 10%

means that each cluster will be split into a number of super-instances equal to 10% of its size, with 1 being equivalent to not generalizing at all. Super-instances are determined by sub-clustering each cluster of the current partition into small groups, each group representing a super-instance. We empirically found that *complete-link* hierarchical clustering is adequate, despite its memory usage which makes it unsuitable on large datasets. Other alternatives include the density-based OPTICS as in [26], or the Furthest Point First (FPF) algorithm [15]. However, user constraints defined on instances need to be transferred to super-instances. This may raise potential conflicts. To avoid this pitfall, we ensure that every super-instance contains no more than one constrained data point. If this is not the case, we split the super-instance using the constrained instances as centers of the new split super-instances. An illustrative case is given in Appendix A.

### 3.2 Constraint Optimization Problem Formulation

Taking advantage of declarative approaches, we formulate the problem of finding a similar partition satisfying user constraints as a Constraint Optimization Problem (COP). In the following, we use the term *instance* to denote a data instance or a super-instance if it is used.

**Variables and Objective Function.** Only instances that are subject to the constraints will be concerned by the COP. Function `GETCONSTRAINEDPARTITION` in Algorithm 1 produces the subset  $X$  containing the constrained instances. For each instance in  $i \in X$ , we define a variable  $G_i$  with the domain  $[1, K]$ , where  $G_i = c$  means instance  $i$  is assigned to cluster  $c$  in the new partition  $\mathcal{P}'$ . Using Eq. (2), the objective function is:

$$\arg \min \sum_{i \in X} \mathbb{I}(G_i \neq \mathcal{P}[i]) \mathcal{D}[i, G_i] \quad (3)$$

**User constraints.** Several instance-level and group-level constraints can be expressed in our model, as below. Must-link (ML)/cannot-link (CL) constraints on two instances  $i, j$  stating that the instances must/cannot be in the same cluster, can be expressed by  $G_i = G_j$  for ML and  $G_i \neq G_j$  for CL. We also compute the transitive closure on ML/CL constraints [25], which derives supplementary constraints according to three rules : (i) if  $ML(a, b)$  and  $ML(b, c)$ , then  $ML(a, c)$ ; (ii) similarly  $ML(a, b)$  and  $CL(b, c)$  imply  $CL(a, c)$  ; (iii) in a binary clustering case ( $K = 2$ ),  $CL(a, b)$  and  $CL(b, c)$  imply  $ML(a, c)$ .

Triplet constraint  $(a, p, n)$  [23] states that a reference instance  $a$  is more similar to instance  $p$  than to instance  $n$ . Instance  $p$  is therefore called positive instance and  $n$  negative. This constraint can be expressed using an implication constraint as follows:

$$G_a = G_n \implies G_a = G_p \quad (4)$$

Span-limited constraints [27] restricting the span of a set of instances  $S \subseteq X$ . A *specific* span-limited constraint states that the instances of  $S$  must be assigned only to clusters from a given subset  $C \subseteq [1, K]$ . It can be expressed using the *count* global constraint<sup>2</sup>:

$$\text{count}(c, [G_i \mid i \in S], =, 0) \quad \forall c \notin C \quad (5)$$

A *generic* span-limited constraint specifies that the instances of  $S$  must be assigned to at most a number  $\gamma$  of clusters. It can be expressed using the constraint *atmost\_nvalue*<sup>3</sup> [5]:

$$\text{atmost\_nvalue}(\gamma, [G_i \mid i \in S]) \quad (6)$$

<sup>2</sup> <https://sofdem.github.io/gccat/gccat/Ccount.html>

<sup>3</sup> [https://sofdem.github.io/gccat/gccat/Catmost\\_nvalue.html](https://sofdem.github.io/gccat/gccat/Catmost_nvalue.html)

More thematic expert feedback can also be integrated as implication constraints. In our model, they are of the form  $P \implies Q$ , where  $P$  and  $Q$  are conjunctions of simpler constraints such as ML/CL.

**Cluster creation.** Our model allows  $\mathcal{P}'$  to have more clusters than  $\mathcal{P}$ , by defining the domain of  $G_i$  as  $[1, K']$ , with  $K' > K$ . This enables assigning an instance to a new cluster if some constraints are unsatisfied. For example, if  $K = 2$  and the expert states three constraints  $G_i \neq G_j, G_j \neq G_l$  and  $G_l \neq G_i$ , then it is necessary to create a new cluster. Given its conditions of apparition, it will typically be very small. In order to prevent the model from creating clusters because it would be optimal to do so w.r.t. the objective function, we set the distance  $\mathcal{D}[i, k']$  to a value greater than all other distances for  $K < k' \leq K'$ .

**Relaxing constraints.** During the incremental process, the expert could make mistakes when trying to improve the partition, which would result in adding conflicting constraints, thus leading to an over-constrained CP model. We reify the user constraints to gain control over constraint satisfaction. Each constraint  $c \in \mathcal{C}$  is associated with a Boolean variable  $S_c$  such that  $S_c = 1$  iff  $c$  is satisfied. The satisfaction rate  $\delta$  sets a lower/upper bound or the exact value of the number of constraints the model must satisfy:

$$\sum_{c \in \mathcal{C}} S_c \stackrel{\cong}{=} \delta \cdot |\mathcal{C}| \quad (7)$$

Constraint relaxation can both solve problems with conflicting constraints and ignore - or warn the user about - constraints that would modify instances far from their new cluster. Relaxation however increases runtime due to the additional Boolean variables. As is, our framework automatically detects ML/CL conflicts and reduces the satisfaction rate accordingly.

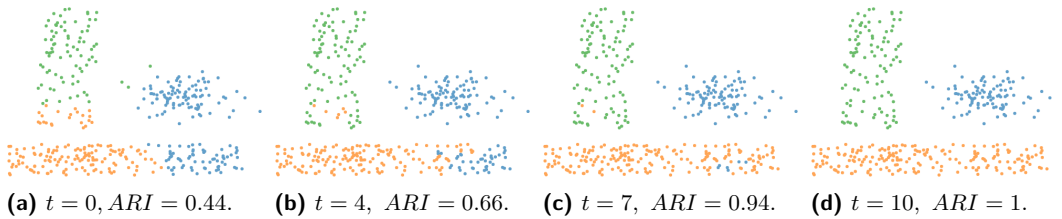
**Managing the constraint store.** Algorithm 1 solves this COP with the user constraints collected at each iteration. The incremental setting raises the issue of managing the constraints between iterations. It is possible to store every constraint received since the beginning of the process to ensure that all expert feedbacks are respected. In the experiments, we choose instead to treat the constraint set given at each iteration independently. In this way, if the expert adds a constraint in conflict with another one given previously, we consider that the user is simply rescinding some of his/her feedback. The expert can also mark some constraints as mandatory so that they are kept satisfied throughout the process. Appropriately managing the constraint store is a potential future research lead.

### 3.3 Active Constraint Selection

Obtaining constraints manually can be costly. This motivates active constraint selection methods [2, 25, 37], which select the most informative constraints to query. To evaluate the interest of exploiting an active constraint selection approach within our framework, we use NPU [37], a neighborhood-based sampling strategy. Neighborhoods  $\mathcal{N}$  are groups of instances whose cluster assignment is certain, they represent the underlying clusters. NPU iteratively builds the neighborhoods by selecting the most informative instance  $x^*$  and querying its relation with respect to existing neighborhoods. The informativeness of an instance  $x$  is defined by the ratio  $H(\mathcal{N}|x)/\mathbb{E}[q(x)]$ , where  $H(\mathcal{N}|x)$  is the entropy measure of the uncertainty to assign  $x$  to a neighborhood in  $\mathcal{N}$ , and  $\mathbb{E}[q(x)]$  the expected number of queries needed to discover its neighborhood.



Exploiting this informativeness, we adapt the NPU framework to the incremental setting. The neighborhoods  $\mathcal{N}$ , which are initially empty, are constructed and kept throughout the iterations. For each informative instance  $x^*$ , queries are put on the membership relation between  $x^*$  and each neighborhood  $N \in \mathcal{N}$ . Once the user answers favorably, a ML constraint is created with the queried neighborhood, otherwise a CL is created. If no ML is achieved, a new neighborhood is created for  $x^*$  (see the detailed algorithm in Appendix B). In relation to constraint management between iterations, it must be pointed out that the preservation of the neighborhoods between iterations prevents selecting constraints conflicting with those selected in earlier iterations. Indeed, an instance stored in a neighborhood is never picked again by NPU, and is only used to generate constraints with an instance that has not been presented to the user before. The only factor that could cause previous constraints to be involuntarily relaxed is a high generalization scope. We found no such occurrences in our experiments with the values we tested for  $\beta$  (see Section 4.2.1). Fig. 3 illustrates the framework walkthrough on a toy example, with noticeable separation between non-spherical clusters that KMEANS is unable to recover.



■ **Figure 3** Illustration of IAC with ( $\alpha = 20\%$ ,  $\beta = 30\%$ ) over 10 iterations on `1sun` dataset, starting from a KMEANS partition (Fig. 3a). Subsequent figures show the evolution of the partition after  $t$  iterations of IAC with NPU. Adjusted Rand Index with ground truth is reported on each figure.

## 4 Experiments

In this section, the experiments aim to answer the following research questions (RQ):

1. What effect do IAC parameters ( $\alpha$  and  $\beta$ ) have on clustering results ? (Section 4.2.1)
2. How does our CP model scale with the number and type of constraints ? (Section 4.2.2)
3. How does the constraint relaxation of IAC compare with other methods that use soft constraints ? (Section 4.2.3)
4. How effective is IAC in an active constraint selection context ? (Section 4.2.4)
5. What is the performance of our framework in terms of clustering quality, partition similarity and runtime when compared to state of the art methods ? (Section 4.2.4)
6. How effective is our framework on a real use case with human feedback ? (Section 4.3)

### 4.1 Experimental Methodology

**Evaluation measures.** For all experiments we use datasets for which a *ground-truth* labeling is known. The produced partition is then compared to the known partition using an external measure. A high value of the measure indicates a good partition, meaning that the clustering algorithm has successfully identified the already known structure. We consider three measures: Adjusted Rand Index (ARI) [19], Adjusted Mutual Information (AMI) [31] and Folkes-Mallows Index (FMI) [13]. ARI measure is defined by:

$$ARI(\mathcal{P}, \mathcal{P}') = \frac{2(ab - cd)}{(a + d)(d + b) + (a + c)(c + b)} \quad (8)$$

where  $a$  (resp.  $b$ ) is the number of instances pairs clustered together (resp. apart) in  $\mathcal{P}$  and  $\mathcal{P}'$ , and  $c$  (resp.  $d$ ) is the number of pairs clustered together (resp. apart) in  $\mathcal{P}$  and apart (resp. together) in  $\mathcal{P}'$ . AMI is a variation of Normalized Mutual Information corrected for chance:

$$AMI(\mathcal{P}, \mathcal{P}') = \frac{MI(\mathcal{P}, \mathcal{P}') - E(MI(\mathcal{P}, \mathcal{P}'))}{\max(H(\mathcal{P}), H(\mathcal{P}')) - E(MI(\mathcal{P}, \mathcal{P}'))} \quad (9)$$

where MI is the measure of mutual information, H the entropy of a partition, and E the expected mutual information if the partitions are random. Finally FMI is the geometric mean of precision and recall, using one partition as a reference for the other:

$$FMI(\mathcal{P}, \mathcal{P}') = \sqrt{\frac{TP}{TP + FP} \cdot \frac{TP}{TP + FN}} \quad (10)$$

We also measure the runtime of the modification step (reclustering or MWCM) since it is crucial to take it into account in the incremental setting. We set a timeout after 1 hour of modification time.

**Experimental protocol.** We have implemented our CP model in Python 3.11 using the CPMpy library [18], interfacing with CP-SAT solver from `or-tools`<sup>4</sup>. We chose this library because it allows to easily use landmark ML libraries such as `scikit-learn` together with CP. Code for reproducing the experiments is available at the repository given at the summary of this paper. The implementation of NPU and all clustering algorithms we considered for comparison (COPKMeans, PCKMeans and MPCKMeans) are from the `active-semi-supervised-clustering`<sup>5</sup> library. All experiments were run on a computer with two 48-core Intel Xeon processors at 4 GHz and 64 GB of RAM running Ubuntu 20.04.

For each dataset, we first generate an initial partition with KMEANS [24] with  $K$  set to the true number of clusters. Queries correspond to pairwise constraints. We emulate user feedback using the ground truth labeling of the data as an oracle i.e. a must-link constraint is added if the selected pair of instances belong to the same class, and a cannot-link otherwise. For **RQs 1, 4** and **5**, we perform 10 iterations of selection-modification loop. At each iteration, we use the current partition to select a batch of 10 queries with NPU, get feedback from the oracle, and apply either a constrained clustering algorithm to the full dataset or our CP model for cluster modification. In order to smooth out the random effects occurring in the partition initialisation and in constraint selection, we repeat each experiment 90 times.

To evaluate the overall performance over the 11 successive partitions (including the initial partition) obtained for each run, we compute for each metric the area under the budget curve (AUBC) [38]) for different fixed budgets of queries to ask the user. Given the budget curve, the AUBC is calculated by the trapezoid method, and the higher value reflects better performance of the evaluated method under varying budgets. For each metric, we compute two types of AUBC:  $AUBC_{quality}$  when comparing the successive partitions to the ground truth partition, and  $AUBC_{similarity}$  when comparing two consecutive intermediate partitions. Since  $AUBC_{similarity}$  values are defined over the interval  $[0, 0.9]$ , we perform a min-max normalization so that all metrics are defined over the  $[0, 1]$  range. To statistically compare the performance of different algorithms and/or configurations of the same algorithm for different parameter settings on multiple data sets, we resort to Bayesian pairwise comparison [4] using

<sup>4</sup> <https://developers.google.com/optimization>

<sup>5</sup> <https://github.com/datamole-ai/active-semi-supervised-clustering>

■ **Table 1** Dataset Characteristics, with  $N$  the number of instances,  $A$  the number of features and  $K$  the number of clusters or classes.

UCI				FCPS			
Name	$(N, A, K)$	Name	$(N, A, K)$	Name	$(N, A, K)$	Name	$(N, A, K)$
iris	(150, 4, 3)	ionosphere	(351, 34, 2)	lsun	(400, 2, 3)	chainlink	(1000, 3, 2)
wine	(178, 13, 3)	yeast	(1484, 8, 10)	target	(770, 2, 6)	wingnut	(1016, 2, 2)
sonar	(208, 60, 2)	statlog	(2310, 19, 7)	atom	(800, 3, 2)	engytime	(4096, 2, 2)
glass	(214, 9, 6)	Letters	(20000, 16, 26)				
ecoli	(336, 7, 8)	MNIST	(70000, 784, 10)				

baycomp<sup>6</sup> library. The principle is to use Bayes’ rule to update a prior statistical distribution representing the null hypothesis (both compared algorithms have the same performance), with a likelihood function modeling the experimental observations. We then get a posterior distribution, reflecting how the prior belief has changed, taking the observations into account. Using the Markov chain Monte Carlo method, the posterior is sampled 50,000 times to estimate the probability of one algorithm being better than the other as well as the probability of being in the region of practical equivalence (or *rope*). In practice, querying the posterior distribution allows to simulate repeating the whole experimental process and to quantify the likelihood of our results. We choose to fix to 1% the difference of performance between the methods as the rope.

## 4.2 UCI and FCPS Datasets

In this section, we report experimental results on ten real-world datasets from the UCI repository<sup>7</sup> and on six synthetic datasets from the FCPS [30] suite designed to address specific challenges to the clustering algorithms such as lack of linear separability, classes defined by data density rather than data spacing, no cluster structure at all, etc. A summary of the basic characteristics is given in Table 1. We used the versions of datasets available under the library `clustering-benchmarks`<sup>8</sup> [14].

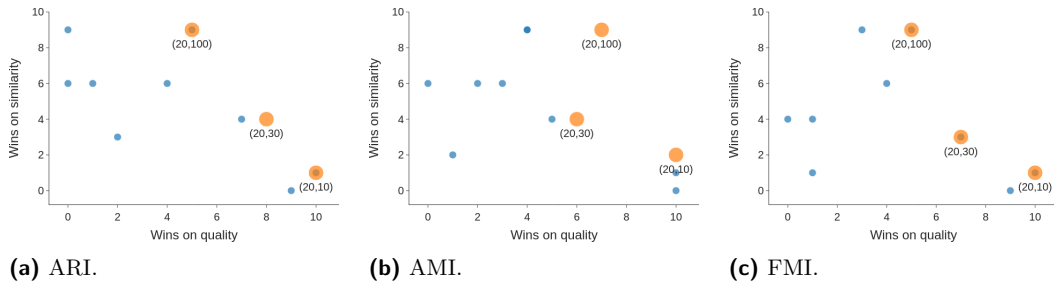
### 4.2.1 Parameter Settings of IAC

To answer **RQ1**, we evaluate the effects of different parameter settings of the clustering modification step of our IAC framework: the anchor generation rate  $\alpha \in \{0\%, 5\%, 20\%\}$  and the super-instance generation rate  $\beta \in \{10\%, 30\%, 50\%, 100\%\}$ . This makes a total of 12 configurations of parameter combinations to evaluate. Recall that  $\alpha = 0\%$  means that we only compute the cluster medoids, while  $\beta = 100\%$  means no generalization by super-instances is performed. For each configuration and each metric, we perform Bayesian pairwise comparison according to  $AUBC_{quality}$  and  $AUBC_{similarity}$  values for each of the three metrics ARI, AMI and FMI over all the datasets and count the number of wins. More precisely, given a pairwise comparison between two configurations  $conf_1$  and  $conf_2$ , using a Bayesian hierarchical model [8], we get three probabilities: the probability that  $conf_1$  has higher scores than  $conf_2$ , the probability that differences are within the region of practical equivalence (rope), or that  $conf_2$  has higher scores. If  $(p_{conf_1} > p_{conf_2} + p_{rope})$ , then we count this comparison as winning for  $conf_1$ .

<sup>6</sup> <https://baycomp.readthedocs.io/en/latest/index.html>

<sup>7</sup> <https://archive.ics.uci.edu/ml/index.php>

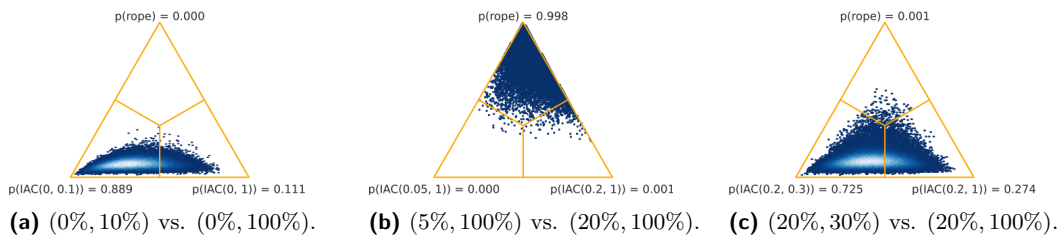
<sup>8</sup> <https://clustering-benchmarks.gagolewski.com/weave/suite-v1.html>



■ **Figure 4** Pairwise plot comparing the number of wins for each configuration  $(\alpha, \beta)$  using a Bayesian hierarchical model, according to  $AUBC_{quality}$  (horizontal) or  $AUBC_{similarity}$  (vertical). Best values are close to the top left, with configurations on the Pareto front in orange.

The table of wins is available in Appendix D. Configurations using anchors (i.e.  $\alpha \neq 0$ ) ensure the highest  $AUBC_{quality}$  values in almost all configurations. Additionally, the best values are obtained with  $\alpha = 20\%$ . For a fixed value of  $\alpha$ ,  $AUBC_{quality}$  values increase significantly with the decrease of  $\beta$ , 10% being the best value. This result suggests that a large scope of generalization does not strongly impact the modification of the clustering and thus its quality.  $AUBC_{similarity}$  values have the opposite behavior:  $\beta = 10\%$  is the worst setting for similarity. Note that the impact of  $\alpha$  seems negligible compared to  $\beta$ . It slightly improves similarity when  $\beta$  is low, while its contribution seems negligible in the absence of generalization. Generalizing modifications understandably degrades similarity, albeit not dramatically. A Pareto front made of three configurations on every metric (see Fig. 4) emerges from the results: one best in quality (20%, 10%), another best in similarity (20%, 100%), and a compromise setup (20%, 30%).

Figure 5 plots the posterior distribution of pairwise comparison of configurations in a simplex. The distribution is shown as a triangle with regions corresponding to different samples of the distribution, e.g. Figure 5a compares the posterior distribution obtained with  $\alpha = 0\%$  (medoids only) and  $\beta = 10\%$  with the one obtained without generalization. This indicates that there is a 88.9% probability that using medoids combined with SI for this value of  $\beta$  is better than using medoids without SI.

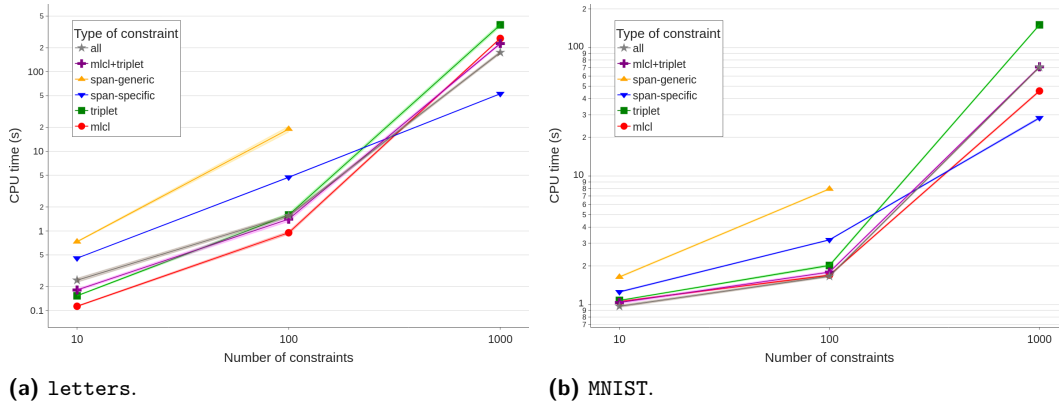


■ **Figure 5** Simplex view of Bayesian comparison of two configurations w.r.t  $AUBC_{quality}$  of ARI. Each sample is plotted according to probabilities  $p_{conf_1}$  (left),  $p_{rope}$  (top) and  $p_{conf_2}$  (right).

## 4.2.2 Impact of Number and Types of User Constraints on Runtime

In this section, we answer **RQ2** by studying two points: the computational efficiency of our CP model and the expressiveness of our approach (see Sect. 3.2). For these experiments, we consider 3 sizes of constraint set (10, 100, 1000). For each test case, we randomly generate sets of four types of constraints (pairwise, triplet, span-limited specific and generic). We compute an initial KMeans partition and run our CP model only once for each set of constraints and

## 10:12 Incremental Constrained Clustering by Minimal Weighted Modification



■ **Figure 6** Evolution of running time of our CP model for the two largest datasets when varying the number and type of constraints, with 95% confidence interval. CPU times are in log-scale.

we report the average CPU times over 90 runs. For pairwise constraints, we disable the computation of transitive closure to keep the number of constraints unchanged. Span-limited constraints are created by randomly choosing 10 instances and finding the ground truth set of clusters - or number of clusters, in the generic case - to which the group belongs.

**Runtime analysis.** Figure 6 shows the results we obtained for `letters` and `MNIST` datasets with  $\alpha = 0\%$  and  $\beta = 100\%$ . Our CP model can process 10 ML/CL constraints in less than 0.05 seconds, while for triplet and specific span-limited constraints the runtime reaches 0.035 and 0.15 seconds respectively. This seems very reasonable in an incremental context. For the `yeast` dataset, with 100 pairwise constraints, it takes 0.35 seconds, whereas for triplet and span-limited constraints the runtime increases up to 1.36 seconds. With 1000 constraints, the runtime is more than 44s for specific span-limited, 67s for ML/CL, and over 180s for triplet; generic span-limited constraints took up too much memory to finish. However, in practice, the number of constraints expected from the user is in the tens rather than the hundreds or thousands. Triplet and span-limited constraints show a substantial increase in runtime compared to ML/CL constraints.

**Mixed constraint types.** One of the advantages of our approach is its ability to easily combine different types of constraints without the need to create a specialized algorithm. To demonstrate this ability, we compared two composite settings:

- **mlcl+triplet:** generate pairwise and triplet constraints in equal proportions, e.g. 50 ML/CL and 50 triplet constraints for the 100 constraints case.
- **all:** similar to **mlcl+triplet**, except a pairwise (resp. triplet) constraint is replaced by a specific (resp. generic) span-limited constraint.

As far as we are aware, such problems cannot be solved by any existing techniques. Problems with ML/CL and triplet constraints take much less time to solve than those involving the three types of constraints, the latter being much more time-consuming. Surprisingly, with 1000 constraints, the ML/CL+Triplet combination takes less time compared to the case where only triplet constraints are involved. All these results underline the relevance of carefully selecting a small number of constraints to guarantee a good compromise between efficiency and quality of the final clustering.

■ **Table 2** Comparative study for clustering with pairwise constraints relaxation for the `mk2` dataset. Metrics are ARI with ground truth (Quality), ARI with unconstrained KMeans (Similarity), runtime and number of constraints relaxed in the solution ( $n_r$ ).

	Test case with conflicts				Test case with $\delta = 94\%$			
	Quality	Similarity	Time	$n_r$	Quality	Similarity	Time	$n_r$
IAC+Anchors	0.576	<b>0.075</b>	5.262	49.7	0.309	<b>0.177</b>	3.051	60.1
IAC+Anchors+SI	<b>0.760</b>	0.024	4.868	49.83	0.393	0.108	2.972	60.1
PCK-Means	0.081	0.051	<b>3.639</b>	149.3	0.375	0.045	<b>2.834</b>	66.5
MPCK-Means	0.078	0.017	29.27	159.9	<b>0.406</b>	0.019	26.98	61.2

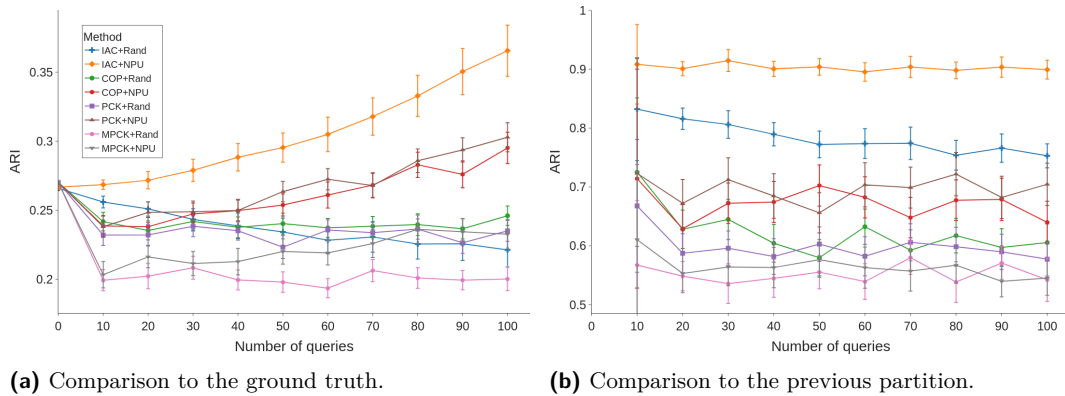
### 4.2.3 Relaxing Constraints

We address the research question **RQ3** by comparing IAC with existing constrained clustering methods for constraint relaxation. We selected two methods: *Pairwise Constrained K-Means* (PCK-Means) [3] allows the violation of ML and CL constraints, and *Metric PCK-Means* (MPCK-Means) [6] combines PCK-Means with distance-metric learning [36]. As the alternatives only support pairwise constraints, we compare performance for problems that involve only ML/CL constraints. To that end, we consider two settings: in the first one, we generate 950 constraints at random based on the ground truth and add 50 conflicting constraints so that some constraints must be relaxed to solve the problem; in the second, we generate 1000 constraints at random without any explicit conflict constraint and set the satisfaction rate  $\delta$  of IAC to the mean satisfaction rate of PCK-Means and MPCK-Means. We run two variants of IAC: IAC+Anchors ( $\alpha = 20\%$ ,  $\beta = 100\%$ ) and IAC+Anchors+SI ( $\alpha = 20\%$ ,  $\beta = 30\%$ ). Table 2 shows the performance of the different approaches for the `mk2` dataset, measured by ARI, runtime and number of constraints relaxed. Each value in the table is the average of 90 runs with different sets of constraints. As previously, for each run, we compute an initial KMeans partition and run our CP model only once for each set of constraints. Results show that in the presence of conflicts, our method violates fewer constraints than the other methods. Furthermore, the runtimes of IAC are comparable to those of PCK-Means. In contrast, MPCK-Means is significantly more expensive. In terms of clustering accuracy (measured by ARI), our approach clearly outperforms the compared to the alternatives. When imposing a satisfaction rate to the model, PCK-Means and MPCK-Means achieve comparable or better quality than IAC, but similarity stays low. However, IAC achieves again better performance in number of constraints relaxed.

### 4.2.4 Comparing IAC with alternatives in the incremental setting

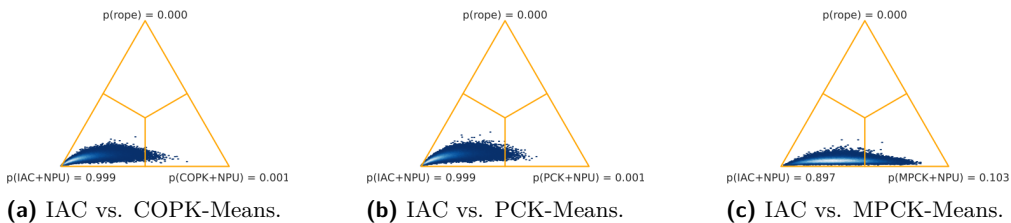
In this section, we address research questions **RQ4** and **RQ5**, and conduct experiments to evaluate the interest and performance of our IAC framework for active constraint section context. NPU can be used in combination with any semi-supervised clustering algorithm, we use the same ones as in the previous section, including COPK-Means algorithm [33]. This leads to several combinations, and for each combination we perform 10 iterations of selection-modification loop. For each iteration, we use the current partition to select a batch of 10 queries with NPU and at random, get feedback from the user, and perform either a reclustering or MCM. In this experiment, queries correspond to pairwise constraints. In light of the results of Section 4.2.1, we choose the compromise configuration between quality and similarity ( $\alpha = 20\%$ ,  $\beta = 30\%$ ) for this experiment.

## 10:14 Incremental Constrained Clustering by Minimal Weighted Modification

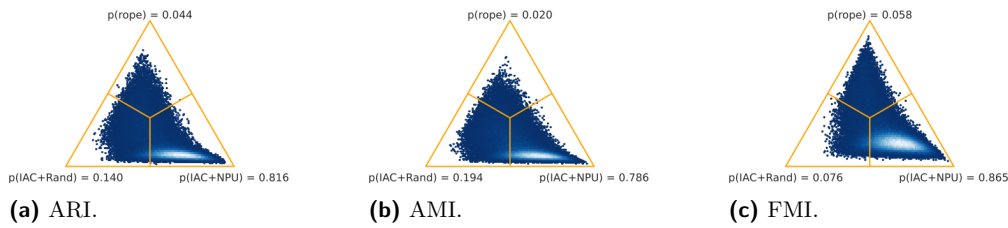


■ **Figure 7** ARI scores of the partition at each iteration of selection-modification, compared to ground truth (left) or to the previous partition (right), for the `glass` dataset. All ARI scores are the mean and 95% confidence interval over 90 runs. Higher is better.

**Clustering quality comparison.** Figure 7 shows the evolution of the ARI scores over 10 iterations of incremental and active clustering modification for the `glass` dataset. IAC+NPU produces increasingly better clusterings as more iterations are given (cf. Fig. 7a), while keeping a high similarity throughout the iterations (cf. Fig. 7b). None of the competitors produces a clustering with a high ARI. Interestingly, both PCK-Means and COPK-Means with NPU are able to find good clusterings while MPCK-Means is not, even after a relatively large number of iterations. However, the behaviour of the competitors are more chaotic in terms of similarity. We observe similar results for the other datasets (due to lack of space, all other results are available via our link in the summary). These results also show that methods with a random selection of constraints produce typically worse results. We validate these observations by Bayesian comparison w.r.t.  $AUBC_{quality}$  and  $AUBC_{similarity}$  values for each metric. Fig. 8 show that IAC has a high probability to perform better than the alternatives on ARI ; we have similar results for AMI and FMI. Interestingly, using NPU leads to better similarity (see Fig. 9). This suggests that the use of an active constraint selection strategy brings another advantage besides improving the quality of the clustering. However, in an online context, runtime is particularly important as it requires user interaction and selecting the next query can be very costly, superseding the time taken for modification. For the biggest datasets (`Letters` and `MNIST`), only methods using random selection finish before timeout. We can conclude from these results that our model for minimal clustering modification is effectively a better way for active incremental constrained clustering compared to the naive approach to incrementality.

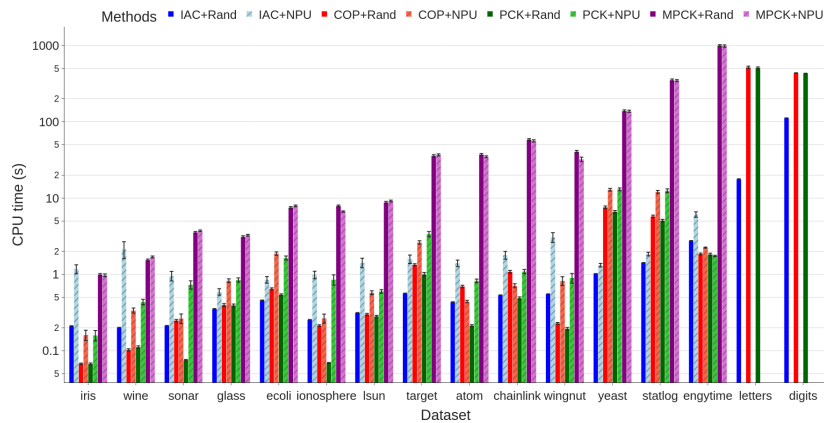


■ **Figure 8** Bayesian comparisons with IAC using NPU w.r.t.  $AUBC_{quality}$  values for ARI.



■ **Figure 9** Bayesian comparison of IAC with or without NPU w.r.t  $AUBC_{similarity}$  for all metrics.

**Runtime comparison.** In Fig. 10, the runtime of modification (reclustering or MWCM) is shown for each dataset. The runtime of our model is comparable to those of the competitors, although it seems to have better scaling on the largest datasets. Empirically, IAC is an order of magnitude faster on *yeast*, *statlog* and *letters* datasets that have a large number of clusters. It is also noteworthy that MPCK-Means is much slower than other methods due to metric learning, yet this increase does not translate into better quality or similarity than IAC. The limiting factor for the modification step of IAC is the computation of anchors and super-instances, whose scaling is worse than solving the COP in itself.



■ **Figure 10** Evolution of the runtime of tested methods on our benchmark datasets, in seconds (log scale). Methods using NPU are hatched.

### 4.3 Tree Cut Data

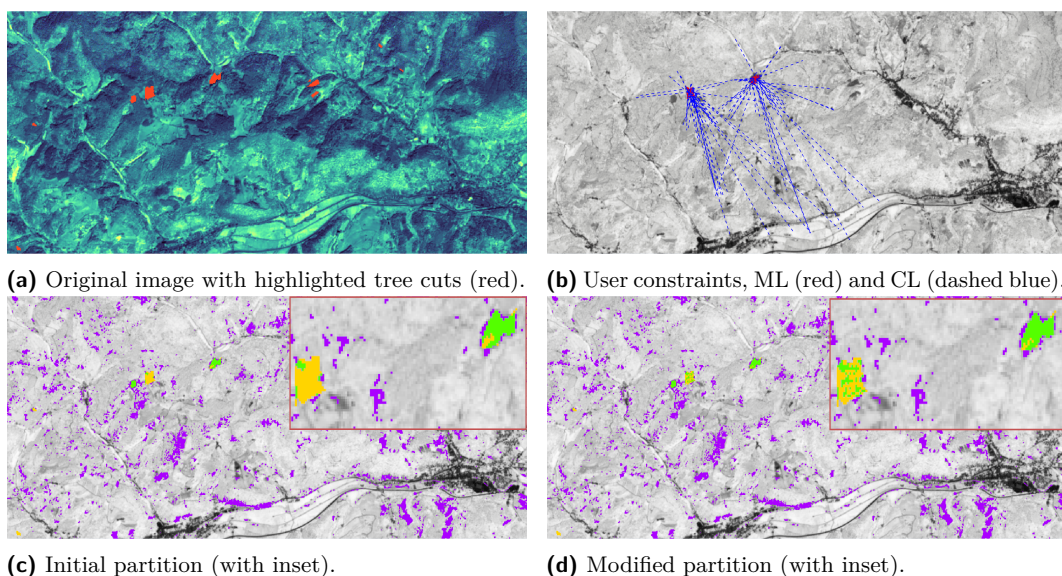
**Introducing the data.** Our case study for research question **RQ6** concerns the analysis of satellite image time series (SITS) composed of 11 images of dimensions  $724 \times 337$  of a zone of the Vosges mountains in eastern France, taken irregularly on the span of 3 years from 2016 to 2018. Each pixel is associated to a series of NDVI (*Normalized Difference Vegetation Index*) values denoting the level of vegetation at each timestamp. At our disposition are labels that separate the SITS into three classes: vegetation, artificial structures, and tree cut zones. This last class has several properties: it was precisely labeled by domain experts, whereas the two other classes are more approximately defined. It is also a very small class as shown in Fig. 11a, containing only 639 instances (less than 0.3% of the data), which makes it hard to detect with unsupervised learning. Image-wise, 10 zones have been identified as places where trees have been cut within the time interval. Lastly, the evolution of these



## 10:16 Incremental Constrained Clustering by Minimal Weighted Modification

zones (a sharp decrease of NDVI value followed by a slow return to normal) is similar to that of field harvesting or grassland mowing, which complicates the problem further. In these conditions, expert intervention is paramount.

**Problem definition.** A set of 179 ML/CL constraints has been collected in [21] from domain experts, focused on the two largest tree cut zones (204 and 147 instances, i.e. more than half of tree cuts) as shown in Fig.11b. We define the problem as recovering these areas with binary constrained clustering. Following [22], we clustered the dataset with K-Means and  $K = 15$ , only retaining the cluster covering the areas the most as the “positive” cluster of our problem. In Fig. 11c, this cluster is displayed in colors, while the “negative” cluster is composed of all pixels not colored. We then used this binary partition as input of IAC, and selected the unsatisfied user constraints in the partition to improve it. In our experiments, 79 constraints were unsatisfied. IAC was set to iterate until all constraints are satisfied. We set  $\beta$  to 100% as the large dataset size means that super-instance computation takes hours to complete, which is not compatible with a real life setting.



■ **Figure 11** Some views of the use case ; pictures (c) and (d) show the “positive” cluster, before and after modification. Highlighted therein are the true positives (green), false negatives (yellow), and false positives (purple). Best viewed in colors.

**Results.** The modified clear cut cluster is displayed in Fig. 11d. The recovering of tree cut can be observed as the green zone of true positives enlarges in this figure : the left area progressed from 37 to 92 true positives, covering almost half the area. The right area gained 10 true positives. The modification was made within 22 seconds.

## 5 Conclusion

We have developed IAC, a framework for clustering modification that can be used in an incremental setting where an expert iteratively adds constraints, either manually or using an active method. A CP model for minimal weighted clustering modification ensures that the general cluster structure is preserved to maintain some continuity between iterations, as

shown in experiments on reference datasets and on a real use case. It can also efficiently exploit an active query strategy to converge faster, and handle contradictory constraints that the user may give as input. The runtime of IAC is dependent of the constraint selection step, which requires further experiments with more active methods and/or to develop a new one suited for the incremental setting. It would also be interesting to explore the use of multiple CP models for modification, such as [20] for minimal modification with cluster-level constraints. Lastly, there remain open questions about the potential reuse of relaxed constraints at a later iteration : What constraints to choose ? When to propose them to the user ? The conception of a strategy answering these interrogations is worth considering.

---

## References

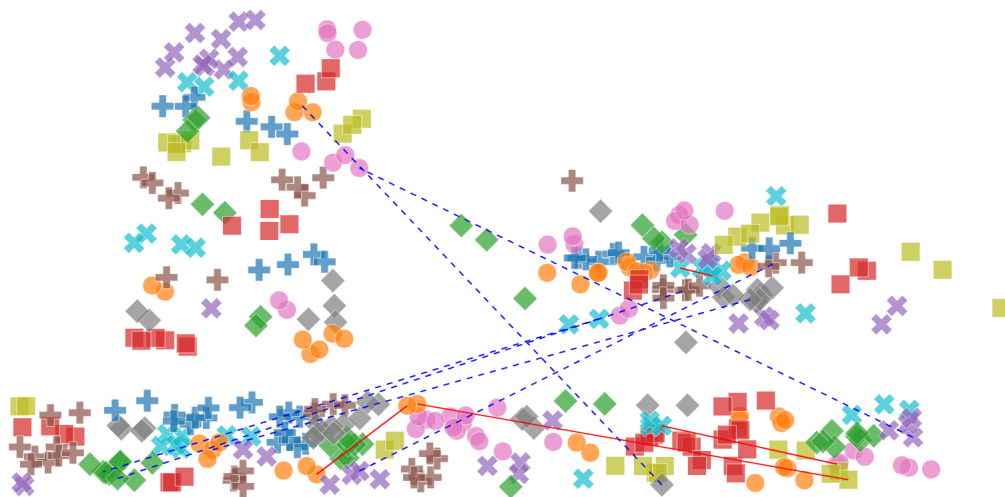
- 1 Arindam Banerjee and Joydeep Ghosh. Scalable clustering algorithms with balancing constraints. *Data Min. Knowl. Discov.*, 13(3):365–395, 2006.
- 2 Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. Active semi-supervision for pairwise constrained clustering. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, pages 333–344. SIAM, 2004. doi:10.1137/1.9781611972740.31.
- 3 Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. Active Semi-Supervision for Pairwise Constrained Clustering. In *ICDM*, pages 333–344, 2004.
- 4 Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: A tutorial for comparing multiple classifiers through Bayesian analysis, 2017.
- 5 Christian Bessière, Emmanuel Hébrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Filtering Algorithms for the NValue Constraint. *Constraints*, 11(4):271–293, 2006.
- 6 Mikhail Bilenko, Sugato Basu, and Raymond J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the 21st International Conference on Machine Learning*, pages 11–18, 2004.
- 7 David Cohn, Rich Caruana, and Andrew McCallum. Semi-Supervised Clustering with User Feedback. Technical report, Cornell University, 2001. doi:10.1201/9781584889977.ch2.
- 8 Giorgio Corani, Alessio Benavoli, Janez Demšar, Francesca Mangili, and Marco Zaffalon. Statistical comparison of classifiers through Bayesian hierarchical modelling. *Machine Learning*, 106(11):1817–1837, November 2017. doi:10.1007/s10994-017-5641-9.
- 9 Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, 244:70–94, 2017.
- 10 Ian Davidson and S. S. Ravi. Agglomerative Hierarchical Clustering with Constraints: Theoretical and Empirical Results. In *Knowledge Discovery in Databases: PKDD 2005*, Lecture Notes in Computer Science, pages 59–70, 2005.
- 11 Ian Davidson, S. S. Ravi, and Martin Ester. Efficient incremental constrained clustering. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 240–249, 2007.
- 12 Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based Framework for Efficient Constrained Clustering. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 94–105, 2010.
- 13 E. B. Fowlkes and C. L. Mallows. A Method for Comparing Two Hierarchical Clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983.
- 14 Marek Gagolewski. A Framework for Benchmarking Clustering Algorithms, 2022.
- 15 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- 16 Germán González-Almagro, Daniel Peralta, Eli De Poorter, José-Ramón Cano, and Salvador García. Semi-Supervised Constrained Clustering: An In-Depth Overview, Ranked Taxonomy and Future Research Directions, 2023.

- 17 Mathieu Guilbert, Christel Vrain, Thi-Bich-Hanh Dao, and Marcilio C. P. de Souto. Anchored Constrained Clustering Ensemble. In *International Joint Conference on Neural Networks, IJCNN*, 2022.
- 18 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Modref*, volume 19, 2019.
- 19 Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.
- 20 Chia-Tung Kuo, S. S. Ravi, Thi-Bich-Hanh Dao, Christel Vrain, and Ian Davidson. A framework for minimal clustering modification via constraint programming. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI'17*, pages 1389–1395, 2017.
- 21 Baptiste Lafabregue, Pierre Gancarski, Jonathan Weber, and Germain Forestier. Incremental constrained clustering with application to remote sensing images time series. In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2022.
- 22 Thomas Lampert, Baptiste Lafabregue, Thi-Bich-Hanh Dao, Nicolas Serrette, Christel Vrain, and Pierre Gancarski. Constrained Distance-Based Clustering for Satellite Image Time-Series. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(11):4606–4621, 2019.
- 23 Eric Yi Liu, Zhaojun Zhang, and Wei Wang. Clustering with relative constraints. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '11*, pages 947–955, New York, NY, USA, August 2011. Association for Computing Machinery. doi:10.1145/2020408.2020564.
- 24 James MacQueen. Some Methods For Classification And Analysis Of Multivariate Observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297, 1967.
- 25 Pavan Kumar Mallapragada, Rong Jin, and Anil K. Jain. Active query selection for semi-supervised clustering. In *19th International Conference on Pattern Recognition*, pages 1–4, 2008.
- 26 Logan Adam Mitchell. *INCREMENT - Interactive Cluster Refinement*. PhD thesis, Brigham Young University, 2016.
- 27 Nguyen-Viet-Dung Nghiem, Christel Vrain, and Thi-Bich-Hanh Dao. Knowledge integration in deep clustering. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2022, Proceedings, Part I*, volume 13713 of *Lecture Notes in Computer Science*, pages 174–190. Springer, 2022. doi:10.1007/978-3-031-26387-3\_11.
- 28 Nguyen-Viet-Dung Nghiem, Christel Vrain, Thi-Bich-Hanh Dao, and Ian Davidson. Constrained Clustering via Post-processing. In *Discovery Science*, *Lecture Notes in Computer Science*, pages 53–67, 2020.
- 29 Abdelkader Ouali, Samir Loudni, Yahia Lebbah, Patrice Boizumault, Albrecht Zimmermann, and Lakhdar Loukil. Efficiently finding conceptual clustering models with integer linear programming. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI'16*, pages 647–654, 2016.
- 30 Alfred Ultsch and Jörn Lötsch. The Fundamental Clustering and Projection Suite (FCPS): A Dataset Collection to Test the Performance of Clustering and Data Projection Algorithms. *Data*, 5(1):13, 2020.
- 31 Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.
- 32 Kiri Wagstaff and Claire Cardie. Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning, ICML '00*, pages 1103–1110, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- 33 Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 577–584, 2001.

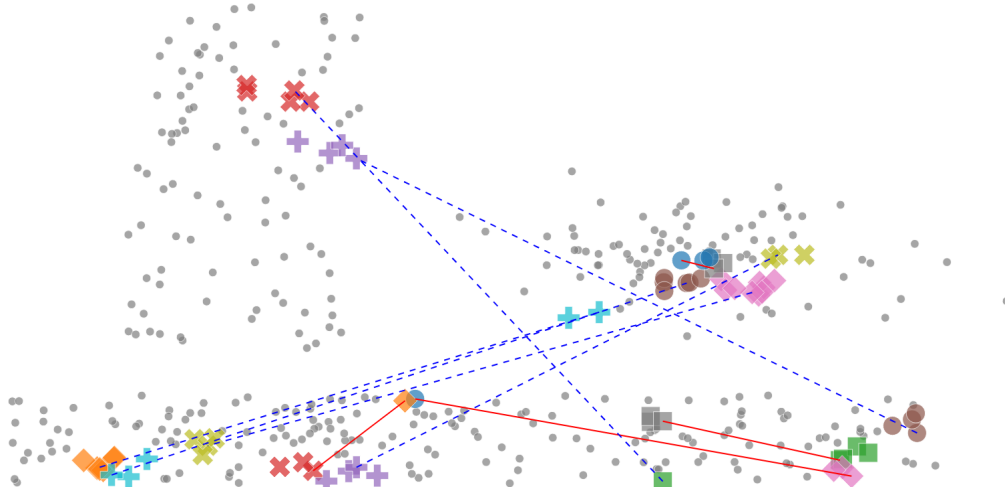
- 34 Kiri L. Wagstaff. Value, Cost, and Sharing: Open Issues in Constrained Clustering. In *Knowledge Discovery in Inductive Databases*, pages 1–10, 2007.
- 35 Xiang Wang and Ian Davidson. Flexible constrained spectral clustering. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 563–572, 2010.
- 36 Eric Xing, Michael Jordan, Stuart J Russell, and Andrew Ng. Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002.
- 37 Sicheng Xiong, Javad Azimi, and Xiaoli Z. Fern. Active Learning of Constraints for Semi-Supervised Clustering. *IEEE Trans. on Knowledge and Data Engineering*, 26(1):43–54, 2014.
- 38 Xueying Zhan, Huan Liu, Qing Li, and Antoni B. Chan. A Comparative Survey: Benchmarking for Pool-based Active Learning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, volume 5, pages 4679–4686, August 2021.

## A Super-instances generation

This is an example case of generating super-instances and splitting to prevent emergent conflicts. In Fig. 12a, the clusters of the partition in Fig. 2 are divided into super-instances. However, some super-instances contain multiple constrained instances, e.g. the green one in the bottom left, which could lead to conflicts. In Fig. 12b, these super-instances have been splitted.



(a) Result of generating super-instances through *complete-link* hierarchical clustering on every cluster.



(b) Final super-instances after splitting. Greyed instances are unconstrained and not used in the CSP.

■ **Figure 12** Exemple preprocessing for super-instance generation on *1sun* dataset. Instances sharing a color are represented by the same super-instance. ML constraints are in red, CL constraints in dashed blue.

## B Incremental NPU

This is the variant of NPU we use in the selection step of IAC. In Algorithm 2, the main modifications are the removal of the reclustering step of the original NPU, and the output of the set of constraints obtained from the queries for the modification step. Algorithm 3 is unchanged and is shown to give a complete view of the algorithm.

### Algorithm 2 Incremental NPU.

---

**Input** : Dataset  $\mathcal{D}$ , partition  $\mathcal{P}$ , oracle  
**Output** : constraint set  $\mathcal{C}$

- 1:  $\mathcal{C} \leftarrow \emptyset$  ;  $l \leftarrow 1$  ;  $\mathcal{N} \leftarrow N_1 \mid N_1 = \{\text{random}(\mathcal{D})\}$
- 2:  $x^* \leftarrow \text{MostInformative}(\mathcal{D}, \mathcal{P}, \mathcal{N})$
- 3: **for** each  $N_i \in \mathcal{N}$  in decreasing order of  $P(x^* \in N_i)$  **do**
- 4:   Query  $x^*$  against any  $x_i \in N_i$  to the oracle
- 5:   **if**  $(x^*, x_i, ML)$  **then**
- 6:      $\mathcal{C} \leftarrow (x^*, x_i, ML)$
- 7:      $N_i = N_i \cup x^*$
- 8:     **break**
- 9:   **else**
- 10:     $\mathcal{C} \leftarrow (x^*, x_i, CL)$
- 11: **if** no ML is returned **then**
- 12:    $l++$  ;  $N_l = x^*$  ;  $\mathcal{N} \leftarrow \mathcal{N} \cup N_l$
- return**  $\mathcal{C}$

---

### Algorithm 3 MostInformative.

---

**Input** : Dataset  $\mathcal{D}$ , partition  $\mathcal{P}$ , set of neighborhoods  $\mathcal{N}$   
**Output** : most informative data point  $x^*$

- 1: Learn a random forest classifier using  $\mathcal{P}$  as labels
- 2: Compute the similarity matrix  $M$  s.t.  $M[i, j]$  is the number of leaves where  $i$  and  $j$  are together normalized by the number of trees of the RF
- 3: **for** each  $x \in \mathcal{U} = \mathcal{D} \setminus \mathcal{N}$  **do**
- 4:   **for**  $i = 1$  to  $l$  **do**
- 5:     
$$p(x \in N_i) = \frac{\frac{1}{|N_i|} \sum_{x_j \in N_i} M(x, x_j)}{\sum_{p=1}^l \frac{1}{|N_p|} \sum_{x_j \in N_p} M(x, x_j)}$$
- 6:      $H(\mathcal{N}|x) = -\sum_{i=1}^l p(x \in N_i) \log_2 p(x \in N_i)$
- 7:      $E(x) = \sum_{i=1}^l i * p(x \in N_i)$
- return**  $\arg \max_{x \in \mathcal{U}} \frac{H(\mathcal{N}|x)}{E(x)}$

---

### C Modifications and generalization

For each modified instance (or super-instance), we store its initial cluster membership and its new cluster membership. This allows the framework to keep a history of modifications and to easily retrieve the partition at any given iteration. Considering generalization, we also keep track of the composition of each constrained super-instance. When the COP produces a solution, Algorithm 4 transmits the modifications from the super-instance to the real data.

■ **Algorithm 4** APPLYMODIFICATIONS.

---

**Input** : dataset  $\mathcal{X}$ , super-instances  $S$ , modifications  $\mathcal{M}$ , partition  $\mathcal{P}$   
**Output** : modified partition  $\mathcal{P}'$

---

- 1:  $\mathcal{P}' \leftarrow \mathcal{P}$
- 2: **for** each  $sp \in S$  **do**
- 3:      $points \leftarrow \{x \in \mathcal{X} \mid x \in sp\}$
- 4:     **for** each  $p \in points$  **do**
- 5:         Update the membership of  $p$  in  $\mathcal{P}'$  with the corresponding value in  $\mathcal{M}$

**return**  $\mathcal{P}'$

---

### D Bayesian pairwise comparison of IAC configurations

■ **Table 3** Number of wins for each configuration  $(\alpha, \beta)$  using a Bayesian hierarchical model. Values in parentheses indicate the number of cases where the probability that a configuration has a higher score is greater than 95%. Values of  $\alpha$  and  $\beta$  are in percentage (%).

$(\alpha, \beta)$	$AUBC_{quality}$			$AUBC_{similarity}$		
	ARI	AMI	FMI	ARI	AMI	FMI
(0,10)	9 (1)	10 (1)	9 (1)	0	0	0
(0,30)	2 (0)	1 (0)	1 (0)	3 (1)	2 (0)	1 (1)
(0,50)	0	0	0	6 (3)	6 (2)	4 (2)
(0,100)	0	4 (1)	3	9 (9)	9 (9)	9 (9)
(5,10)	10 (3)	10 (3)	10 (3)	1 (0)	1 (0)	1 (0)
(5,30)	7 (3)	5 (3)	7 (3)	4 (2)	4 (1)	3 (1)
(5,50)	1 (0)	2 (0)	1 (0)	6 (4)	6 (3)	4 (2)
(5,100)	5 (2)	4 (0)	5 (1)	9 (9)	9 (9)	9 (9)
(20,10)	10 (7)	10 (6)	10 (6)	1 (1)	2 (1)	1 (0)
(20,30)	8 (3)	6 (3)	7 (3)	4 (2)	4 (2)	3 (1)
(20,50)	4 (1)	3 (1)	4 (1)	6 (5)	6 (4)	6 (4)
(20,100)	5 (2)	7 (3)	5 (1)	9 (9)	9 (9)	9 (9)

# Simplifying Step-Wise Explanation Sequences

**Ignace Bleukx** ✉ 

DTAI, KU Leuven, Belgium

**Jo Devriendt** ✉ 

DTAI, KU Leuven, Belgium

**Emilio Gamba** ✉ 

Data Analytics Lab, VUB, Brussels, Belgium

DTAI, KU Leuven, Belgium

**Bart Bogaerts** ✉ 

Artificial Intelligence Lab, VUB, Brussels, Belgium

**Tias Guns** ✉ 

DTAI, KU Leuven, Belgium

Data Analytics Lab, VUB, Brussels, Belgium

---

## Abstract

Explaining constraint programs is useful for debugging an unsatisfiable program, to understand why a given solution is optimal, or to understand how to find a unique solution. A recently proposed framework for explaining constraint programs works well to explain the unique solution to a problem step by step. It can also be used to step-wise explain why a model is unsatisfiable, but this may create redundant steps and introduce superfluous information into the explanation sequence. This paper proposes methods to simplify a (step-wise) explanation sequence, to generate simple steps that together form a short, interpretable sequence. We propose an algorithm to greedily construct an initial sequence and two filtering algorithms that eliminate redundant steps and unnecessarily complex parts of explanation sequences. Experiments on diverse benchmark instances show that our techniques can significantly simplify step-wise explanation sequences.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming

**Keywords and phrases** explanation, deduction, constraint programming, propagation

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.11

**Supplementary Material** *Software (Source code)*: <https://github.com/ML-KULeuven/SimplifySeq> archived at `swh:1:dir:a0de0a65a6c2c3a69d84a1de4047a360d8854195`

**Funding** This research was partly funded by the Flemish Government (AI Research Program), the Research Foundation - Flanders (FWO) project G070521N; the European Research Council (ERC) under the EU Horizon 2020 research and innovation programme (Grant No 101002802, CHAT-Opt) and the European Union's Horizon 2020 research and innovation programme under grant agreement No 101070149, project Tuples.

## 1 Introduction

As AI agents become more expressive and powerful, a growing need arises for *explainable AI*: methods that can explain the decisions of an automated agent. Such explanations can be needed for legal reasons,<sup>1</sup> but are also essential to provide trust to users. Considerable research has gone into developing explanation techniques for black-box machine learning methods [17, 27], including techniques based on formal models [33].

---

<sup>1</sup> E.g., the GDPR – <https://gdpr.eu>





## 11:2 Simplifying Step-Wise Explanation Sequences

But also explaining formal models themselves, such as constraint programs and satisfiability problems [37], is of high importance. Although individual constraints typically have a clear meaning, their interaction can be highly non-trivial, which led to the development of explanation methods for constraint programs [21]. A prominent branch of *explainable constraint solving* is occupied with explaining why a set of constraints is unsatisfiable. Most of these methods [30, 32, 26, 28, 31, 25, 29] extract a *minimal unsatisfiable subset* (MUS): a (minimal) subset of the constraints that renders the problem unsatisfiable. Such a MUS provides a user with a (potentially large) subset of constraints that yield inconsistency. Recently, there has been work on guiding users with respect to *what* can be done to restore feasibility [22, 39], but there is a lack of tools to better explain *why* a problem is inconsistent.

In a sense, traces or proof-logs of a solver (as common in SAT [24] and recently also finding its way into richer formalisms [2, 18, 3, 8, 4]) provide an explanation of why a model is unsatisfiable. Still, they would quickly overwhelm a user, as it involves constraint reformulations, auxiliary variables, and branching decisions. Other solver-generated explanations can be extracted from highly effective *Lazy Clause Generation* solvers [34] combining constraint propagation and SAT solving. In this setting, every propagation is *explained* by adding a clause to the SAT solver. However, these kinds of explanations are by nature restricted to explaining the propagation of a single constraint. Instead, we start from *step-wise explanations* [7, 15] where each explanation step in a sequence refines a partial assignment, using a minimal set of *constraints* as well as *facts* derived in previous steps.

These step-wise approaches were developed in the context of explaining the unique solution of satisfiable logic puzzles. Each step explains why a certain variable in the unique solution took that specific value. An example is explaining how to solve a Sudoku-puzzle [13, 20] where each step derives the value of a cell in the solution. In this context, the number of explanation steps is at most the number of variables in the problem.

But this explanation framework can be applied more broadly to constraint satisfaction problems (CSPs), in particular also to those that are unsatisfiable. A step-wise explanation of an over-constraint model allows to *debug* it, by listing steps similar to a debugger for programming languages. Each step shows a (preferably small) subset of constraints causing the removal of allowed values in variables domains, up to where a conflict is derived.

In contrast to explanation sequences for satisfiable problems, in the unsatisfiable case, only a subset of the variables and derived values contribute to the conflict and should therefore be explained. Therefore, directly applying the step-wise explanations framework to this new setting results in overly complex explanations.

Furthermore, finding *the shortest* sequence of explanations is a hard problem. A recent paper [6] touches upon the complexity of finding the shortest sequence of arc-consistency propagations steps. In that setting, the goal is to explain the full result of the arc-consistency algorithm. In the general case, where other/stronger propagation algorithms are used to find propagation steps, the problem may become even harder.

In this paper, we do not consider finding *the best* explanation sequence. Instead, we investigate how to find good – interpreted here as short and with small steps – step-wise explanations in the context of explaining why a CSP is unsatisfiable. The proposed techniques also apply to explaining the objective value of an optimization problem, explaining the solution(s) of constraint problems, and can be of use in interactive configuration problems [23].

For this, we contribute the following:

1. For the first time, we consider the quality of explanation *sequences* as a whole;
2. We formalize the properties that good explanation sequences, and explanation steps from which they are built, should adhere to;

3. We propose a new normal form for explanation sequences;
4. In sections 5 and 6 we propose new algorithms to simplify sequences in this normal form;
5. We show our methods significantly simplify explanation steps *and* sequences compared to current approaches.

## 2 Preliminaries

We now formalize constraint satisfaction problems and step-wise explanation sequences as used throughout this paper.

### 2.1 Constraint Satisfaction Problems (CSPs)

► **Definition 1** (CSP). A CSP is a triple  $(\mathcal{X}, \mathcal{D}, \mathcal{C})$  [37] with

- $\mathcal{X}$  a set of variables;
- $\mathcal{D}$  a set of domains  $D_x$  of allowed values for each variable  $x$  of  $\mathcal{X}$ , i.e.,  $\mathcal{D} = \{D_x \mid x \in \mathcal{X}\}$ ;
- $\mathcal{C}$  a set of constraints, each over a subset of the variables.

A *full assignment* to a CSP  $(\mathcal{X}, \mathcal{D}, \mathcal{C})$  is a mapping such that each variable takes a value from its domain. A *constraint* is a function mapping full assignments to true or false, typically described by a formula (e.g.,  $x + y \geq 1$ ). A constraint is *satisfied* by a full assignment if it maps the constraint to true. A *solution* to a CSP is a full assignment satisfying all constraints in  $\mathcal{C}$ . A CSP is *unsatisfiable* if it has no solution. A set of constraints  $\mathcal{S}$  is a *logical consequence* of another set  $\mathcal{S}'$  if all solutions to  $\mathcal{S}'$  are solutions to  $\mathcal{S}$  – written as  $\mathcal{S}' \models \mathcal{S}$ . Constraints can be arranged in a *constraint graph* where nodes represent variables that are connected by an edge if they co-occur in a constraint.

For the remainder of this paper, we assume the set of variables and their domains are known. A *positive literal* is an equality  $x = v$  and a *negative literal* is an inequality  $x \neq v$  for variable  $x$  and value  $v$ . Negative literals represent the constraint that a variable cannot be assigned to some value from its domain. We will often employ sets of negative literals, where we use  $x \in \mathcal{R}$  as a shorthand for  $\{x \neq v \mid v \in D_x \setminus \mathcal{R}\}$ . For example, with  $D_x = \{0, 1, 2, 3\}$ ,  $x \in \{1, 2\}$  means the negative literals  $\{x \neq 0, x \neq 3\}$ . In other words,  $x \in \mathcal{R}$  denotes a set of negative literals enforcing that  $x$  can *only* take values remaining in  $\mathcal{R}$ . Note that a positive literal  $x = v$  is equivalent to the set of negative literals  $x \in \{v\}$ . For the remainder of this paper, literals are assumed to be **negative** unless explicitly mentioned otherwise. With  $\perp$  we denote the *trivial inconsistency*, i.e., the singleton set containing the literal *false*.

For simplicity, this paper only uses *integer domains* for variables. An integer domain is represented as either a finite range with a lower and upper bound, or an enumerated set. E.g., the range  $[0..3]$  and the set  $\{0, 1, 2, 3\}$  are identical. All ideas and algorithms in this paper apply to non-integer finite domains as well.

Given a set of constraints partitioned in *soft* constraints and a set of *hard* constraints, a *minimal unsatisfiable subset* (MUS) is a subset of the soft constraints that is unsatisfiable in conjunction with the hard constraints, and for which all strict subsets are satisfiable in conjunction with these hard constraints. MUS-calculation techniques are a well-studied research field and several algorithms for this exist [25, 32]

### 2.2 Step-wise explanations

The *step-wise explanation framework* was introduced in the context of first-order logic and Boolean satisfiability [10, 9, 7, 15]. We here reinterpret it from a finite-domain CSP perspective.

## 11:4 Simplifying Step-Wise Explanation Sequences

Given a CSP  $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ , an *explanation step* is a triple  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$ , where the *input*  $\mathcal{E}$  and the *output*  $\mathcal{N}$  are disjoint sets of literals, and  $\mathcal{S} \subseteq \mathcal{C}$  is the *constraint subset*, such that  $\mathcal{E} \cup \mathcal{S} \models \mathcal{N}$ . Informally, an explanation step consists of a subset of constraints which, together with some input literals, imply “new” output literals.

Given a CSP  $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ , a *target* set of literals  $\mathcal{T}$  and a *given* set of literals  $\mathcal{G}$ , an *explanation sequence* of length  $n$  from  $\mathcal{G}$  to  $\mathcal{T}$  is a sequence  $\langle (\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i) \rangle_{1 \leq i \leq n}$  of explanation steps where  $\mathcal{E}_i \subseteq \mathcal{G} \cup \bigcup_{1 \leq j < i} \mathcal{N}_j$  and  $\mathcal{T} \subseteq \mathcal{G} \cup \bigcup_{1 \leq j \leq n} \mathcal{N}_j$ . Informally, an explanation sequence is a sequence of explanation steps where each step derives some new literals and can do so using  $\mathcal{G}$  as well as previously derived outputs. Eventually, the sequence should derive all literals in  $\mathcal{T}$ . As a whole, an explanation sequence explains how a target set is entailed by the union of a given set and the constraints of a CSP.

For satisfiable CSPs with a unique solution, we can let  $\mathcal{T}$  be a unique variables assignment. In general, we can set  $\mathcal{T}$  to the intersection of all solutions. An explanation sequence where  $\mathcal{G} = \emptyset$  and  $\mathcal{T}$  is inconsistent explains a CSP’s unsatisfiability.

We say an explanation step *derives* or *explains* a literal  $l$  if  $l \in \mathcal{N}$ , and a sequence derives or explains a literal if one of its steps does.

The *maximal output* for an input  $\mathcal{E}$  and constraint subset  $\mathcal{S}$  is the set of all literals implied by  $\mathcal{E} \cup \mathcal{S}$ , minus  $\mathcal{E}$ . The maximal output can be calculated using a FULLPROPAGATE algorithm which finds the set of literals that are true in *all* solutions to the constraints. Such a FULLPROPAGATE function is implemented in multiple systems such as the Answer Set Programming system Clasp [16], the IDP system [11], and more recently in the pseudo-Boolean solver Exact [12, 14].

► **Definition 2** (Maximal sequence). *An explanation step  $(\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i)$  in a sequence is maximal if (1)  $\mathcal{E}_i$  is the union of all previously derived literals and the given set, i.e.,  $\mathcal{E}_i = \mathcal{G} \cup \bigcup_{j < i} \mathcal{N}_j$ , and (2)  $\mathcal{N}_i$  is the maximal output of  $\mathcal{E}_i$  and  $\mathcal{S}_i$ . An explanation sequence where all steps are maximal is a maximal sequence.*

Given a sequence of constraint sets  $\langle \mathcal{S}_i \rangle_{1 \leq i \leq n}$  and a given set  $\mathcal{G}$ , the *maximal* step-wise explanation sequence is the unique sequence  $\langle (\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i) \rangle_{1 \leq i \leq n}$  where all explanation steps are maximal. Clearly, maximal sequences contain much more input/output literals than a user would care about. Moreover, calculating the maximal output of a step using FULLPROPAGATE is an expensive operation. In the general case, any sound propagation algorithm can be used to calculate the output of a step, but using a maximal one will provide us with a useful normal form.

► **Example 3.** Consider the following unsatisfiable CSP and explanation sequences which we will use as a running example:

- $\mathcal{X} = \{x, y, z, v, w\}$
- $\mathcal{D} = \{D_x = D_y = [1..3], D_z = D_v = D_w = [0..3]\}$
- $\mathcal{C} = \{x + y + z \geq 7, x + y + w \leq 4, x \leq v, v + z \leq 3\}$

### 3 Greedy initial sequence construction

Naively, generating explanation sequences involves constructing sequences of explanation steps that neatly match each other’s input and output. In existing work [7, 15], explanation sequences are constructed using an iterative loop that greedily searches for the best next explanation step to add. Each individual step in the explanation sequence is optimal with respect to an assumed cost function or heuristic. This cost function for example takes into account the number of constraints and the number of input literals used for each step.

■ **Table 1** Two explanation sequences for the same unsatisfiability for  $\mathcal{G} = \emptyset$  and  $\mathcal{T} = \perp$ .

(a)				(b)			
$i$	$\mathcal{E}$	$\mathcal{S}$	$\mathcal{N}$	$i$	$\mathcal{E}$	$\mathcal{S}$	$\mathcal{N}$
1	$\emptyset$	$x + y + z \geq 7$	$z \neq 0$	1	$\emptyset$	$x + y + z \geq 7$ $x + y + w \leq 4$	$z \in [3..3]$
2	$\emptyset$	$x + y + z \geq 7$ $x + y + w \leq 4$	$z \in [3..3]$ $w \neq 3$	2	$\emptyset$	$x \leq v$	$v \in [1..3]$
3	$z \in [3..3]$	$x \leq v$ $v + z \leq 3$	$\perp$	3	$v \in [1..3]$ $z \in [3..3]$	$v + z \leq 3$	$\perp$

To find a set of constraints and literals that explains a new literal  $n$ , given a sequence of  $i - 1$  previous steps, we can extract an unsatisfiable subset from  $\mathcal{C} \cup \mathcal{G} \cup \bigcup_{j < i} \mathcal{N}_j \cup \{\neg n\}$ . This is precisely what is done in the algorithms presented in [7], by enumerating subsets  $\mathcal{S}$  of increasing size and finding a small MUS in this way for each literal to explain.

Such a construction method works well for explaining unique solutions, where the target contains *all* consequences of the CSP. However, for arbitrary target set, a literal might be derivable with a simple step, but deriving that literal doesn't bring us any closer to explaining the target. Nevertheless, literals that do not appear in the target may well be useful to build other intermediate steps with. Balancing which literals (and their associated explanation steps) to include in the sequence, or to exclude, will be a crucial theme in the next sections.

We propose a greedy sequence construction algorithm inspired by [7]. Given a CSP  $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ , a given set  $\mathcal{G}$  and a target set  $\mathcal{T}$ , Algorithm 1 CONSTRUCT-GREEDY describes our construction algorithm. It keeps track of an input set  $\mathcal{E}$  that contains the union of  $\mathcal{G}$  and all literals explained so far. Using this set of input literals, it iteratively tests if constraint subsets  $\mathcal{S} \subseteq \mathcal{C}$  of growing size  $|\mathcal{S}| = i$  can produce an explanation step. For this, the algorithm calculates the maximal output  $\mathcal{N}$  for  $\mathcal{E} \cup \mathcal{S}$  for which  $\mathcal{E} \cup \mathcal{S} \models \mathcal{N}$  and  $\mathcal{N} \cap \mathcal{E} = \emptyset$  using a FULLPROPAGATE routine. If this output set  $\mathcal{N}$  is not empty, i.e., a new literal could be derived, it adds the step  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$  to the sequence, extends  $\mathcal{E}$  with  $\mathcal{N}$  and resets the constraint subset size  $i$ . This process is repeated until the target has been explained ( $\mathcal{T} \subseteq \mathcal{N}$ ) and the sequence is finished.

Notice the algorithm is guaranteed to terminate as  $\mathcal{S}$  grows in every iteration. Eventually,  $\mathcal{S}$  will be equal to  $\mathcal{C}$  if no smaller subset was able to derive a new literal. Of course, we here assume  $\mathcal{G} \cup \mathcal{C} \models \mathcal{T}$ .

► **Example 4.** The table below shows a maximal explanation sequence with given set  $\mathcal{G} = \emptyset$  and target set  $\mathcal{T} = \perp$  for the following unsatisfiable CSP:

- $\mathcal{X} = \{p, q, r, s\}$
- $\mathcal{D} = \{D_p = D_q = D_r = D_s = [0..3]\}$
- $\mathcal{C} = \{p + q \leq 1, q + 2r \leq 4, 3s + p + r \leq 1, \text{alldiff}(p, q, r, s)\}$

### 3.1 Properties

► **Lemma 5.** For each step  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$  in a sequence generated by CONSTRUCT-GREEDY, and for each  $\mathcal{S}' \subsetneq \mathcal{S}$ , the maximal output for  $\mathcal{S}'$  and  $\mathcal{E}$  is empty.

Otherwise CONSTRUCT-GREEDY would have picked  $\mathcal{S}'$  to form a step with  $\mathcal{E}$ . A direct consequence of this is that the sequences constructed by CONSTRUCT-GREEDY are *atomic*:

► **Property 1 (Atomic).** An explanation step  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$  is atomic if no explanation sequence from  $\mathcal{E}$  to  $\mathcal{N}$  exists in which each step uses a strict subset of  $\mathcal{S}$ . An explanation sequence is atomic if all its steps are.

■ **Algorithm 1** CONSTRUCT-GREEDY .

---

**Input:** CSP  $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ , given  $\mathcal{G}$ , target  $\mathcal{T}$

```

1  $Seq \leftarrow$  empty sequence
2  $\mathcal{E} \leftarrow \mathcal{G}, i \leftarrow 0$ 
3 while true do
4   forall  $\mathcal{S} \subseteq \mathcal{C}$  where  $|\mathcal{S}| = i$  and  $CONNECTED(\mathcal{S})$  do
5      $\mathcal{N} \leftarrow FULLPROPAGATE(\mathcal{E} \cup \mathcal{S}) \setminus \mathcal{E}$ 
6     if  $\mathcal{N} \neq \emptyset$  then
7       extend  $Seq$  with  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$ 
8        $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{N}, i \leftarrow 0$ 
9       if  $\mathcal{T} \subseteq \mathcal{E}$  then
10        | return  $Seq$ 
11        | break
12    $i \leftarrow i + 1$ 

```

---

■ **Table 2** A maximal sequence that could have been constructed by CONSTRUCT-GREEDY.

$i$	$\mathcal{E}$	$\mathcal{S}$	$\mathcal{N}$
1	$\emptyset$	$p + q \leq 1$	$p \in [0..1]$ $q \in [0..1]$
2	$p \in [0..1]$ $q \in [0..1]$	$q + 2r \leq 4$	$r \neq 3$
3	$p \in [0..1]$ $q \in [0..1] r \neq 3$	$3s + p + r \leq 1$	$r \neq 2$ $s \in [0..0]$
4	$p \in [0..1] q \in [0..1]$ $r \in [0..1] s \in [0..0]$	$\text{alldiff}(p, q, r, s)$	$\perp$

The third step of the first sequence of Example 3 is not atomic: it can be split into two steps over  $x \leq v$  and  $v + z \leq 3$  each, as is done in the second sequence of the same example.

We believe this property is desirable for easy-to-understand explanation sequences as it requires each explanation step to be as small as possible. Note that explanation sequences generated by the literature [7, 15] also exhibit atomicity.

### 3.2 Efficiency optimizations

The two main bottlenecks in this algorithm are *enumerating large subsets of constraints* and *calculating the maximal output  $\mathcal{N}$*  for a given  $\mathcal{E}$  and  $\mathcal{S}$ . To improve runtime, we employ two efficiency optimizations. First, we cache calls to the FULLPROPAGATE routine. When a maximal output call for some  $\mathcal{E}$  and  $\mathcal{S}$  is repeated, the output is just taken from the cache. This is useful as the domains of many variables are unchanged between calls and the propagation only considers the literals from  $\mathcal{E}$  with regard to the variables occurring in  $\mathcal{S}$ .

A second optimization inspects the constraint graph of each enumerated subset  $\mathcal{S}$  using the CONNECTED function call on line 4 in the algorithm. This function checks if the constraint graph is connected. If not, the graph can be split up into two or more components  $\mathcal{S}_c$  and the maximal output  $\mathcal{N}$  for  $\mathcal{S}$  and  $\mathcal{E}$  is exactly the union of the maximal outputs  $\mathcal{N}_i$  for each component  $\mathcal{S}_c$  and input  $\mathcal{E}$ . By Lemma 5, all smaller  $\mathcal{S}_c$  did not imply any new literal, so  $\mathcal{S}$  cannot imply any either. Naturally, this means CONSTRUCT-GREEDY can skip subset  $\mathcal{S}$ .

### 3.3 Comparison to literature

The idea of CONSTRUCT-GREEDY is similar to the first literature approach [7]: iterate over increasingly larger subsets  $\mathcal{S} \subseteq \mathcal{C}$  and check whether an explanation step  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$  exists.

In contrast to [7], CONSTRUCT-GREEDY always greedily picks the first explanation step that can derive at least one unexplained literal to add to the explanation sequence, instead of generating multiple candidates and picking the optimal one under some quality function. The reasons for this difference are twofold: a simpler algorithm and less computational effort per step. Compared to logic puzzles, which are aimed to be solved by humans and contain at most hundreds of literals, for arbitrary CSPs the literals can easily grow into the millions (e.g., for large integer domains), so efficiency is more of a concern here. This is also the reason we do not employ the second literature approach [15], whose algorithm computes the optimal next explanation step according to a linear cost function directly, without explicit enumeration, by exploiting the hitting set duality between MUS's and correction subsets.

Still, by iterating over small constraint subsets first, CONSTRUCT-GREEDY adds steps that employ a small set of constraints, which is preferred to easily understand the explanation [7].

Note that CONSTRUCT-GREEDY constructs *maximal* sequences. This contrasts with the literature approaches which minimize the input literals for each step during construction. The motivation for this difference is that we first want to minimize the number of steps and the number of constraints for each step. The more literals are derived by a step, the more literals future steps can use as input. Thereby potentially needing fewer constraints to derive new literals. Moreover, by deriving more literals in a single step, the quicker we might reach the target set.

One can also minimize the number of input literals for each step in the sequence during post-processing of the sequence, thereby simulating the sequences produced by literature approaches in terms of input literals [7, 15]. In Section 7, this is precisely what is denoted as **Filter simple**.

## 4 Simple post-processing

In this section, we consider two straightforward ways to simplify explanation sequences requiring little to no computational effort.

### 4.1 Looseness

For an explanation sequence to be interpretable, we deem the length of the sequence to be an important metric. Each step in an explanation sequence requires effort by a user to process and understand. Naturally, this means no *loose* steps should be part of the sequence:

► **Property 2 (Loose).** *Given an explanation sequence  $\langle (\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i) \rangle_{1 \leq i \leq n}$  of  $\mathcal{T}$  from  $\mathcal{G}$ , the step  $(\mathcal{E}_j, \mathcal{S}_j, \mathcal{N}_j)$  is called loose if  $\langle (\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i) \rangle_{i \neq j}$  still forms an explanation sequence of  $\mathcal{T}$  from  $\mathcal{G}$ . An explanation sequence is loose if one of its steps is loose.*

A step is loose if it can be left out of the sequence. This is the case for instance when none of a step's output literals is used as input for later steps or is part of the target set.

The first step in the sequence 1a of Example 3 is loose, as it can just be removed without impacting the sequence. In sequence 1b of that same example, no loose steps are present as at least one output of each step is used later on.

Existing approaches [7, 15] focused mostly on explaining a unique solution of a CSP, where every step derives at least one yet unexplained literal of that unique solution. This means every step is guaranteed to derive a literal in the target set. However, [7] also describes

## 11:8 Simplifying Step-Wise Explanation Sequences

*nested explanations.* These clarify a single explanation step  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$  by creating a new explanation sequence from  $\mathcal{S}$  with  $\mathcal{E}$  and the negation of  $\mathcal{N}$  as the given set, and  $\perp$  as target set. The construction of nested explanations in [7] may lead to *loose* steps.

Loose steps are easy to detect in a sequence and can simply be removed from the explanation sequence until none remain.

### 4.2 Pertinence

After constructing a sequence and filtering out loose steps, the output of a step may still contain irrelevant literals as not *all* output literals have to be used later in the sequence. This requires a user to “waste” effort in understanding why useless literals are implied by the step. Clearly, this is undesired, motivating the following definition.

► **Property 3 (Pertinent).** *An explanation sequence is pertinent if (i)  $\mathcal{G}$  and the  $\mathcal{N}_i$  are pairwise disjoint, and (ii) all output literals are part of the target or the input of some following step.*

The first condition states that each literal should only be derived once (and obviously, given literals should not be rederived). The second condition states that everything we derived should be used. To transform an explanation sequence into a pertinent one, we can simply loop over the sequence and remove any outputs not satisfying any of the conditions in the definition. Removing loose steps from a pertinent sequence corresponds to removing steps with an empty output set. In Example 3, the first step of sequence 1a is not pertinent, as the literal  $w \neq 3$  in its output is not part of any input or of the target. For satisfiable CSPs, the algorithms described in the literature [7] are guaranteed to produce a pertinent explanation sequence, but not so for unsatisfiable CSPs.

## 5 Relaxation-based filtering

Informally, a sequence is pertinent if it derives only those literals that are (indirectly) needed for the target. However, a step in a sequence may also contain input literals that are not needed to derive its output. Naturally, this is an undesired property of explanation steps as it requires the user to *process* more literals than needed.

► **Property 4 (Sparse).** *An explanation step  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$  is sparse if no step  $(\mathcal{E}', \mathcal{S}, \mathcal{N})$  with  $\mathcal{E}' \subsetneq \mathcal{E}$  exists. An explanation sequence is sparse if all its steps are.*

The sequence of Example 4 contains non-sparse steps, e.g., the second step does not need the literals  $p \in [0..1]$  as input to derive  $z \neq 3$ . The sequences in Example 7 are both sparse.

Previous approaches [7, 15] construct explanation steps by minimizing the input  $\mathcal{E}$ , making the resulting steps and sequences sparse. In contrast, CONSTRUCT-GREEDY constructs a maximal sequence which typically has non-sparse steps due to non-useful output literals of previous steps. We address this by a *relaxation-based filtering* algorithm, which calculates a so-called *relaxation* for each step.

► **Definition 6 (Relaxation).** *A relaxation of a step  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$  is a step  $(\mathcal{E}', \mathcal{S}, \mathcal{N})$  with  $\mathcal{E}' \subseteq \mathcal{E}$ .*

An explanation step is sparse if and only if it allows no strict relaxations. Given a set of implied literals  $\mathcal{N}$ , let  $\neg\mathcal{N}$  be the constraint denoting that at least one of the literals in  $\mathcal{N}$  is false, i.e.,  $\neg\mathcal{N} = \bigvee_{x \neq v \in \mathcal{N}} x = v$ . Then,  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$  is sparse if for every subset  $\mathcal{E}' \subsetneq \mathcal{E}$  it holds that  $\mathcal{E}' \cup \mathcal{S} \cup \{\neg\mathcal{N}\}$  is satisfiable, as in that case,  $\mathcal{E}' \cup \mathcal{S} \not\models \mathcal{N}$  and  $\mathcal{E}'$  cannot form a relaxation for  $(\mathcal{E}, \mathcal{S}, \mathcal{N})$ .

■ **Algorithm 2** RELAXATION-BASED FILTERING.

---

**Input:** Explanation sequence  $Seq$ , given set  $\mathcal{G}$ , target set  $\mathcal{T}$

```

1  $Req \leftarrow \mathcal{T} \setminus \mathcal{G}$ 
2 for  $(\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i) \in reverse(Seq)$  do
3    $\mathcal{N}_i \leftarrow Req \cap \mathcal{N}_i$ 
4   if  $\mathcal{N}_i$  is empty then
5     drop  $(\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i)$  from  $Seq$ 
6   else
7      $\mathcal{E}_i \leftarrow MUS(soft: \mathcal{E}_i, hard: \mathcal{S}_i \cup \{\neg \mathcal{N}_i\})$ 
8      $\mathcal{N}_i \leftarrow FULLPROPAGATE(\mathcal{E}_i \cup \mathcal{S}_i)$ 
9      $Req \leftarrow ((Req \setminus \mathcal{N}_i) \cup \mathcal{E}_i) \setminus \mathcal{G}$ 

```

---

To calculate sparse steps, we need to find a relaxation for each step. For this, we can calculate a MUS with  $\mathcal{E}$  as soft constraints and  $\mathcal{S} \cup \{\neg \mathcal{N}\}$  as hard constraints. This MUS is a subset of  $\mathcal{E}$  that yields a sparse step.

A straightforward algorithm to make all steps sparse would be to iterate over the steps and calculate such a MUS. However, the disadvantage of this is that sequences may no longer be pertinent: if the input of some step is reduced, then the output of a previous step may no longer be needed. And conversely, naively making a sequence pertinent may make it no longer sparse, as a smaller output for a step may reduce the number of input literals needed.

Instead, we propose a method that makes a sequence sparse by relaxing its steps from back to front. Additionally, this will make some steps loose, allowing the method to remove these steps. Hence, we call it *RELAXATION-BASED FILTERING*.

Algorithm 2 shows the pseudocode. It loops from back to front over the sequence and keeps a set of *required* literals  $Req$  that still need to be explained by some step earlier in the sequence.  $Req$  is initialized as the target set  $\mathcal{T}$ . For each step in the sequence, the algorithm first keeps only those output literals that are also in  $Req$ , since keeping more would make the sequence non-pertinent (they are not required for the rest of the sequence). Next, it relaxes the step by calculating a MUS.

In the general case, many such MUSes may exist, and choosing one MUS over the other can have a profound impact on the sequence. This is clearly visible in Example 7 where the difference between both sequences arises from choosing a different MUS to relax the last step.

In this setting, we want a MUS allowing us to keep the set of required literals  $Req$  as small as possible. The reason for this is twofold: we want to keep future relaxations as small as possible and increase the number of detected loose steps. To compute a MUS satisfying this preference, we can split the calculation into two parts: first, minimize the extra input literals needed, and next minimize the subset of  $Req$ . Other techniques for this exist and include the OCUS-algorithm from [15].

The above process yields a sparse step deriving all required literals the step originally derived. However, the sparse input and constraints may imply more output literals than the originally *new* literals. To detect this, we make  $\mathcal{N}$  maximal in line 8 of the algorithm. Any output this step can derive in this way can be removed from  $Req$ . But any input the step uses should be added to  $Req$ . This is what happens in the last line.

RELAXATION-BASED FILTERING guarantees sparsity by design since the input of each step is generated by a MUS call. Transforming the sequence into a pertinent one can be done trivially by removing literals from each  $\mathcal{N}$  if they are not used later in the sequence or if they are derived as well at some point earlier in the sequence, as argued in Section 4.2.



## 11:10 Simplifying Step-Wise Explanation Sequences

Ensuring pertinence of this sequence in this way will not break sparsity in case the original sequence was maximal (which is guaranteed by our construction). To see this note that by definition in a maximal sequence, each literal is derived as soon as it can possibly be derived. Hence, any literal in  $\mathcal{N}$  after Line 3 of the algorithm will *not* be derived at any earlier point in the sequence. Since these literals are required, they are part of some later input, and hence will not be removed from the output. For this reason, the step will stay sparse after making the sequence pertinent.

If the input sequence was maximal, then after RELAXATION-BASED FILTERING and making the sequence pertinent, there will also be no loose steps. To see this, note if RELAXATION-BASED FILTERING keeps a step, it is because some of its newly derived literals were required. If the input sequence is maximal, all literals are derived as early in the sequence as possible, meaning that this literal will not be derived earlier (and hence the pertinence making of the sequence will never remove it).

► **Example 7.** Below are two filtered versions of the maximal sequence in Example 4

■ **Table 3** Two sparse explanation sequences that could be produced using RELAXATION-BASED FILTERING by filtering the maximal sequence of example 4.

$i$	$\mathcal{E}$	$\mathcal{S}$	$\mathcal{N}$	$i$	$\mathcal{E}$	$\mathcal{S}$	$\mathcal{N}$
1	$\emptyset$	$p + q \leq 1$	$p \in [0..1]$ $q \in [0..1]$	1	$\emptyset$	$3s + p + r \leq 1$	$p \in [0..1]$ $r \in [0..1]$ $s \in [0..1]$
2	$\emptyset$	$3s + p + r \leq 1$	$r \neq 2, r \neq 3$	2	$p \in [0..1]$ $r \in [0..1]$ $s \in [0..1]$	$\text{alldiff}(p, q, r, s)$	$\perp$
3	$p \in [0..1]$ $q \in [0..1]$ $r \in [0..1]$	$\text{alldiff}(p, q, r, s)$	$\perp$				

## 6 Deletion-based filtering

The techniques proposed so far are essential to make explanation sequences more easily understandable by users. The experiments in Section 7 will show that RELAXATION-BASED FILTERING significantly reduces the number of steps for many sequences. However, more steps could be removed if we allow to change the input and output of other, later steps. In the left sequence of Example 7, step 1 can be omitted from the sequence, as shown on the right in that same example. However, depending on the exact MUSes (and hence, relaxations) calculated by RELAXATION-BASED FILTERING, it may miss this, and the step remains *redundant*.

► **Property 5 (Redundant).** *Given an explanation sequence  $\langle (\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i) \rangle_{1 \leq i \leq n}$  of  $\mathcal{T}$  from  $\mathcal{G}$ , the step  $(\mathcal{E}_j, \mathcal{S}_j, \mathcal{N}_j)$  is redundant if the maximal explanation sequence arising from  $\mathcal{G}$  and  $\langle \mathcal{S}_i \rangle_{i \neq j}$  is an explanation sequence of  $\mathcal{T}$ . An explanation sequence is redundant if one of its steps is redundant.*

Intuitively, a redundant step is a step for which all useful literals it derives could also be derived by later steps in the sequence. The first and second steps in Example 4 are redundant, as there exists a maximal sequence with the constraints of steps three and four: it is exactly the right sequence of Example 7.

The literature approaches [7, 15] may very well lead to redundant sequences, as once a step is added to the sequence under construction, no check is later made whether the step could be subsumed by one added later. Notice this also holds for the application of explaining the unique solution of a logic puzzle, whereas *loose* steps could not occur in that setting.

■ **Algorithm 3** DELETION-BASED FILTERING.

---

**Input:** Explanation sequence  $Seq$  with maximal inputs and outputs, given set  $\mathcal{G}$ , target set  $\mathcal{T}$

```

1 for  $(\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i) \in reverse(Seq)$  do
2   let  $Sub$  be a copy of  $Seq$  from  $i + 1$  to end
3   if  $TRYDELETION(\mathcal{E}_i, Sub)$  then
4     shrink  $Seq$  to size  $i - 1$ 
5     append  $Sub$  to  $Seq$ 
6 Function  $TRYDELETION(\mathcal{E}_i, Sub)$ :
7    $\mathcal{N}_i \leftarrow \emptyset$ 
8   for  $(\mathcal{E}_j, \mathcal{S}_j, \mathcal{N}_j) \in Sub$  do
9      $\mathcal{E}_j \leftarrow \mathcal{G} \cup \mathcal{E}_{j-1} \cup \mathcal{N}_{j-1}$ 
10     $\mathcal{N}_j \leftarrow FULLPROPAGATE(\mathcal{E}_j \cup \mathcal{S}_j)$ 
11    if  $\mathcal{T} \subseteq \mathcal{N}_j$  then
12      return True
13  return False

```

---

Algorithm DELETION-BASED FILTERING filters redundant steps. For this, it assumes a maximal sequence as input – if not, the sequence can be made maximal by a linear iteration over all steps from front to back. Next, DELETION-BASED FILTERING iterates over the sequence from back to front. In every iteration, it leaves out the current step, calculates the resulting maximal explanation sequence, and checks if it still explains the target. Calculating the resulting maximal sequence requires a call to a full propagation routine for  $\mathcal{E} \cup \mathcal{S}$  for each later step to derive the new outputs.

In the worst case where no step can be removed from a sequence of length  $n$ , this results in  $n(n - 1)/2$  calls to a full propagation routine. Although the pseudo-Boolean solver Exact allows for stateful computation of these maximal outputs, this remains a computationally expensive task and several optimizations are needed to make the algorithm work in practice, as we now explain.

## 6.1 Necessary condition on the existence of a sequence

For a maximal sequence with given set  $\mathcal{G}$  and target set  $\mathcal{T}$ , DELETION-BASED FILTERING has to check for a step  $(\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i)$  whether the maximal sequence determined by  $\langle \mathcal{S}_j \rangle_{i < j \leq n}$  and  $\mathcal{E}_i$  as given set, still derives  $\mathcal{T}$ . A necessary condition is that  $\mathcal{E}_i \cup \bigcup_{i < j \leq n} \mathcal{S}_j \models \mathcal{T}$  – if not,  $\mathcal{T}$  cannot be explained with just the constraints from the remaining steps in the sequence. This can be checked via a simple solve call with the constraints  $\mathcal{E}_i \cup \bigcup_{i < j \leq n} \mathcal{S}_j \cup \neg \mathcal{T}$ . In case this returns a solution, step  $i$  cannot be dropped from the sequence.

## 6.2 Partial propagation

Full propagation – calculating the maximal output of a step – is an expensive operation. However, weaker and more efficient propagation algorithms are researched [5, 38] and are implemented in several state-of-the-art constraint solvers. When DELETION-BASED FILTERING has to check whether a target set  $\mathcal{T}$  can be derived by the maximal sequence arising from  $\langle \mathcal{S}_i \rangle_{j \leq i \leq n}$  and given set  $\mathcal{E}_i$ , it can first compute a “partial maximal” sequence using a computationally cheaper propagation algorithm. If the target set is explained according to this weaker sequence, it also will be explained by the maximal sequence, so the step can be dropped and DELETION-BASED FILTERING can continue to the next iteration.

### 6.3 Caching

As explained in Section 3.2, we can cache the result of calls to a full propagation algorithm. In addition, we can cache the result of computing a maximal sequence (containing multiple maximal outputs), and we can cache the result of checking the necessary condition for a maximal sequence as described in Section 6.1. These caching strategies become particularly useful given the monotonicity of entailment, as formalized in the following lemma.

► **Lemma 8.** *For any set of constraints  $\mathcal{S}$  and any two sets of literals  $\mathcal{E} \subseteq \mathcal{E}'$ , if  $\mathcal{E} \cup \mathcal{S} \models \mathcal{N}$  then  $\mathcal{E}' \cup \mathcal{S} \models \mathcal{N}$ .*

This entails that if a maximal sequence determined by  $\langle \mathcal{S}_j \rangle_{i < j \leq n}$  with given set  $\mathcal{E}_i$  entails a target set  $\mathcal{T}$ , so will the maximal sequence using the same constraints with a superset of  $\mathcal{E}_i$ . Conversely, if that maximal sequence does *not* entail  $\mathcal{T}$ , neither will the maximal sequence with any subset of  $\mathcal{E}_i$ .

We propose to generate explanation sequences using CONSTRUCT-GREEDY and filter these sequences using DELETION-BASED FILTERING so no *redundant* steps remain. Lastly, these non-redundant sequences need to undergo RELAXATION-BASED FILTERING so all steps are made sparse and can be made pertinent. In Appendix A, we construct a formal argument to prove why this pipeline ensures *atomicity*, *sparsity*, *pertinence* and *non-redundancy* for general explanation sequences.

Notice that filtering explanation sequences using DELETION-BASED FILTERING can potentially filter more steps compared to RELAXATION-BASED FILTERING but may increase the complexity of individual steps as more literals are used/explained at a time. Nevertheless, the constraint sets of each step are not altered by any of the algorithms we proposed. These are entirely determined by the construction algorithm.

## 7 Experimental results

We evaluate and review the algorithms and ideas presented in previous sections using three benchmark sets consisting of unsatisfiable CSP instances.

**Sudoku.** 50 9x9 sudoku instances in which an empty cell is given a wrong value, such that this variable assignment is non-trivial, i.e., it does not directly falsify any constraint. The original puzzles were generated by the QQwing [35] tool using the “Intermediate” difficulty setting. The sudoku constraints are modelled with `AllDifferent` global constraints.

**Jobshop.** 50 jobshop instances generated by the approach in [40] with 5 machines and 5 tasks per job. The time horizon is equal to 50 units. Unsatisfiability arises from restricting the makespan to a better-than-optimal value. All instances are modelled using `Cumulative` constraints [1] as is usual in scheduling problems.

**Debug.** Using the diverse set of CSP models from Håkan Kjellerstrand,<sup>2</sup> we construct a set of 202 unsatisfiable CSPs by introducing artificial errors in the models to mimic a modelling error by a user – an unsatisfiability *bug*. We follow a similar approach to [29]. The errors are introduced such that each constraint separately remains satisfiable.

<sup>2</sup> <http://www.hakank.org/cpmpy/>

These three benchmark families correspond to three use cases where step-wise explanations can be of help. **Sudoku** simulates the counterfactual explanation provided during an interactive collaboration (e.g., [23]) by a human agent and an automated system. Here the user wants to know why the system derived that a certain value was no longer possible for a certain value. **Jobshop** corresponds to explaining to a user why an inferred objective value is optimal for some optimization problem. **Debug** simulates the situation where a user is modelling a CSP but discovers the model has no solutions and the user needs to “debug” it.

For every instance, we compute three explanation sequences using different random seeds. As metrics, we consider the number of steps in sequences, as well as the complexity of individual steps. This first metric is different from what literature approaches consider [7, 15] as in their use-case of explaining a unique solution, they focus only on the complexity of individual steps.

**RQ1.** How many redundant steps do explanation sequences of unsatisfiable CSPs contain?

**RQ2.** How well can the proposed filtering techniques simplify explanation sequences?

**RQ3.** How much time do the proposed algorithms take to run?

**RQ4.** How does simplifying the set of input constraints affect the explanation sequence?

## 7.1 Experimental setup

All experiments were run on a single core of an Intel(R) Xeon(R) Silver 4214 CPU with 128GB of RAM. FULLPROPAGATE is handled by the stateful `pruneDomains` functionality of the Exact solver [12, 14] v1.0.0.<sup>3</sup> Partial propagation is calculated with OR-Tools [36] v9.6 presolve routine. All algorithms are implemented using a custom branch of CPMpy [19] v0.9.12, embedded in Python v3.10.9. To calculate MUSes we employ the `mus` tool provided by CPMpy. Cardinal-minimal MUSes are computed using the hitting-set approach of [25]. All code and benchmarks are available on GitHub: <https://github.com/ML-KULEuven/SimplifySeq>

## 7.2 Approaches under investigation

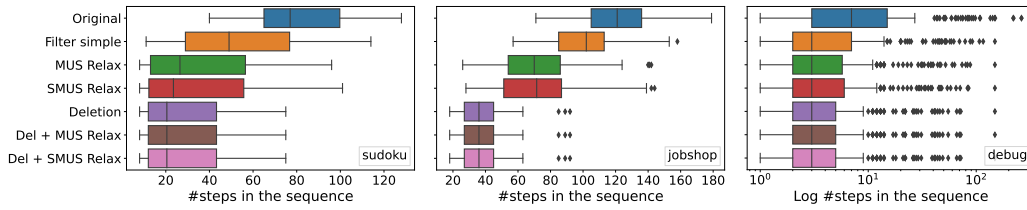
Throughout this section, we investigate all algorithms and combinations of them presented in this paper. All sequences are generated using CONSTRUCT-GREEDY and are indicated using **Original** in each of the plots. By making each step sparse, we can mimic the explanation sequences generated by approaches described in the literature [7, 15]. This can simply be done by finding a minimal unsatisfiable subset with hard constraints  $\mathcal{S} \cup \{\neg\mathcal{N}\}$  and soft constraints  $\mathcal{E}$ . Afterwards filtering *loose* steps and making the sequence *pertinent* using the approach described in Section 4 is denoted as **Filter simple** and is considered current state-of-the-art. We consider two versions of RELAXATION-BASED FILTERING differing in the type of MUS extracted: *a* MUS and *a* smallest MUS (SMUS), shown as **MUS Relax** resp. **SMUS Relax**. Lastly, we investigate sequences exposed to the entire pipeline using DELETION-BASED FILTERING and both versions of RELAXATION-BASED FILTERING. These are shown as **Del + MUS Relax** and **Del + SMUS Relax**.

## 7.3 Redundant steps

The first research question is straightforward: how many redundant steps are there in naively generated sequences, and how many can we filter using our proposed techniques? From Figure 1 it is clear greedy explanation sequences contain a lot of redundant steps. For all

<sup>3</sup> <https://gitlab.com/JoD/exact> git commit 34c4fa46

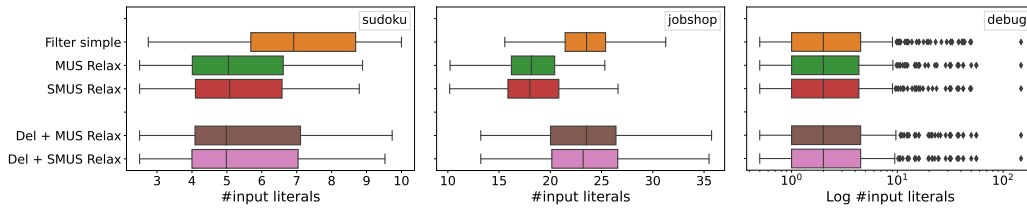
## 11:14 Simplifying Step-Wise Explanation Sequences



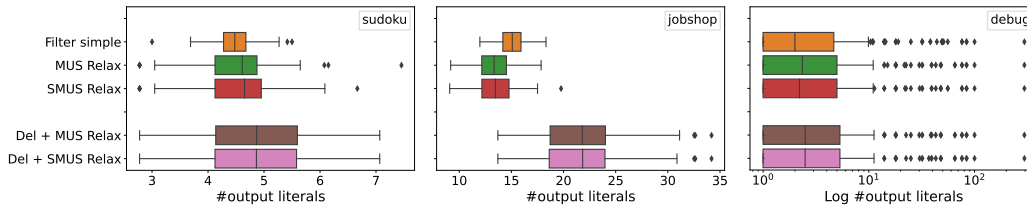
■ **Figure 1** Comparison of different filter algorithms in terms of **number of steps** in the sequence.

benchmarks, the deletion-based method leads the way in terms of the number of steps filtered from the sequence. The price to be paid is a significant increase in computational effort for some benchmarks and more complex individual steps as outlined in the next sections. Simple filtering of loose steps as described in Section 4 is able to reduce the number of steps, but falls short compared to fully-fledged relaxation-based methods. This is most prominently visible in the **Jobshop** benchmark set. Interestingly, both variants of RELAXATION-BASED FILTERING (**MUS Relax** and **SMUS Relax**) show similar performance in terms of steps left in the sequence after filtering.

### 7.4 Complexity of individual steps



(a) Comparison in terms of **average number of input literals** for each step in an explanation sequence.



(b) Comparison in terms of **average number of output literals** for each step in an explanation sequence.

■ **Figure 2** Complexity of individual steps in terms of literals, algorithms producing maximal sequences are omitted from these plots.

Next, we investigate how different filtering methods compare in terms of input and output literals for each step. The simple filtering method is able to keep the number of input literals relatively low. As expected, it again falls short compared to relaxation-based methods as **Filter simple** does not modify the set of input literals based on the output literals that are actually useful. While the deletion-based methods are able to remove more steps, it increases the number of input literals for the remaining steps compared to relaxation-based methods. This is similar to the number of output literals, shown in Figure 2b. Deletion-based methods require steps to derive more output while relaxation-based methods work better for this metric. We argue this larger number of output literals is less of a concern compared to the increase of input literals as steps can be split up so they derive fewer literals at a time.

The left-hand side of Table 4 shows that the number of constraints is low for each individual step, ranging from 1 to a few constraints propagated at a time. Notice how during filtering, some steps occupying little constraints are deleted, while the largest step for each sequence is always present after filtering still.

## 7.5 Explaining a MUS

■ **Table 4** Effect of explaining a MUS of the unsatisfiable problem instead of all constraints in the problem. Metrics are shown before filtering → after filtering using DELETION-BASED FILTERING.

	Explaining all constraints			Explaining a MUS		
	#steps	Max $ \mathcal{S} $	Avg $ \mathcal{S} $	#steps	Max $ \mathcal{S} $	Avg $ \mathcal{S} $
<b>Sudoku</b>	79.62 → 26.33	1.00 → 1.00	1.00 → 1.00	46.10 → 27.87	2.04 → 2.04	1.07 → 1.08
<b>Jobshop</b>	118.03 → 35.69	1.02 → 1.02	1.00 → 1.00	96.55 → 38.10	1.35 → 1.35	1.01 → 1.01
<b>Debug</b>	18.70 → 5.91	1.32 → 1.32	1.10 → 1.17	6.86 → 5.51	1.48 → 1.48	1.20 → 1.26

In the literature, the most common way to explain why a model is unsatisfiable is to first extract a MUS. What happens to step-wise explanations when we first extract a MUS and build the explanation sequence from that MUS?

The result of this experiment is summarized in Table 4. Here we observe that when extracting a MUS, the explanation sequence is much shorter before any filtering is applied. Interestingly, explaining the full CSP may actually lead to shorter sequences after filtering, as observed for **Sudoku** and **Jobshop**. Equally remarkable, limiting the explanation sequence to a MUS increases the number of constraints used in individual steps. This is most clear from **Sudoku** where the maximal number of constraints in a step increases from 1 constraint to 2.04 constraints on average. Intuitively, limiting the constraints with which to build an explanation sequence to a smaller unsatisfiable subset reduces the ease with which explanations can be built. E.g., the Sudoku problem contains many redundant constraints that are implied by some subset of other constraints. Nevertheless, these redundant constraints can be very useful to elegantly explain something to a user.

## 7.6 Runtime and optimizations

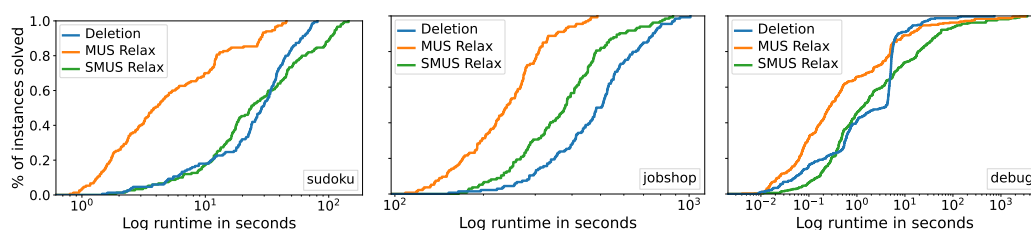
Figure 3 displays the runtime for each of the filtering algorithms for all benchmark sets. For **Jobshop**, the deletion-based method is computationally more expensive compared to any of the relaxation-based variants. This is not the case for **Sudoku** and **Debug**. As **Cumulative** constraints are harder to fully propagate compared to **AllDifferent** constraints, the quadratic number of full-propagation calls is much more prominent for the **Jobshop** benchmark. For all benchmarks, calculating a smallest-minimal MUS requires more computational effort while not impacting the sequence enough to justify this extra cost.

The optimizations implemented for DELETION-BASED FILTERING as described at the end of Section 6 are most effective with the **Debug** benchmark, where only 7% of the theoretic number of full-propagation calls are executed. For **Sudoku** and **Jobshop**, this is 19% resp. 34%. Overall, the proposed optimizations are necessary to reach the observed performance.

## 8 Conclusion and Future work

We investigated the problem of step-wise explaining unsatisfiable problems, with applications in debugging unsatisfiability, explaining optimality, and explanations in interactive configuration. To ensure a *simple* step-wise explanation, both the number of steps and the

## 11:16 Simplifying Step-Wise Explanation Sequences



■ **Figure 3** ECDF plot relating solved instances to runtime for different filtering algorithms.

number of constraints and literals in each explanation step must be kept low. We proposed the formal properties of *atomicity*, *pertinence*, *sparsity*, and *(ir)redundancy*, to which an explanation sequence should strive to adhere. For this, we proposed a workflow combining greedy construction, deletion-based filtering and relaxation-based filtering that guarantees all properties are satisfied.

We presented extensive experimental results comparing current approaches to our proposed techniques. This showed filtering of explanation sequences is especially essential in the challenging setting of explaining unsatisfiability and is key in creating easy-to-understand explanation sequences. Our methods are able to considerably shorten explanation sequences with deletion-based filtering leading to sequences without any redundant steps. Compared to relaxation-based techniques which may produce longer sequences, deletion-based methods may lead to more complex steps after filtering.

For future work, observe that none of the filtering algorithms considered in this paper alter the set of constraints for explanation steps, nor the order of explanation steps. These are entirely determined by the construction algorithm. Changing either of these aspects might further simplify an explanation sequence. We leave these areas for future work, where both new sequence construction methods and more elaborate post-processing are viable options to explore.

Finally, the small number of constraints in most explanation steps shows that computing explanation sequences seem a worthwhile approach to help a user understand unsatisfiable constraints. Still, the expressivity of explanation steps deserves further study with our results opening the door for further investigation into what domain experts perceive as “simple” explanation sequences. User studies or learning techniques can investigate how it helps them to understand the interplay of complex constraints.

---

### References

- 1 Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. In Jean-Paul Delahaye, Philippe Devienne, Philippe Mathieu, and Pascal Yim, editors, *JFPL'92, 1<sup>ères</sup> Journées Francophones de Programmation Logique, 25-27 Mai 1992, Lille, France*, page 51, 1992.
- 2 Mario Alviano, Carmine Dodaro, Johannes Klaus Fichte, Markus Hecher, Tobias Philipp, and Jakob Rath. Inconsistency proofs for ASP: the ASP - DRUPE format. *Theory Pract. Log. Program.*, 19(5-6):891–907, 2019. doi:10.1017/S1471068419000255.
- 3 Haniel Barbosa, Andrew Reynolds, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Arjun Viswanathan, Scott Viteri, Yoni Zohar, Cesare Tinelli, and Clark W. Barrett. Flexible proof production in an industrial-strength SMT solver. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*,

- volume 13385 of *Lecture Notes in Computer Science*, pages 15–35. Springer, 2022. doi:10.1007/978-3-031-10769-6\_3.
- 4 Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE)*, 2023. Accepted for publication.
  - 5 Christian Bessiere. Constraint propagation. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 29–83. Elsevier, 2006. doi:10.1016/S1574-6526(06)80007-6.
  - 6 Christian Bessiere, Clément Carbonnel, Martin C. Cooper, and Emmanuel Hebrard. Complexity of Minimum-Size Arc-Inconsistency Explanations. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CP.2022.9.
  - 7 Bart Bogaerts, Emilio Gamba, and Tias Guns. A framework for step-wise explaining how to solve constraint satisfaction problems. *Artif. Intell.*, 300:103550, 2021. doi:10.1016/j.artint.2021.103550.
  - 8 Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelfth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 3698–3707. AAAI Press, 2022. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/20283>.
  - 9 Jens Claes, Bart Bogaerts, Rocsildes Canoy, Emilio Gamba, and Tias Guns. ZebraTutor: Explaining how to solve logic grid puzzles. In Katrien Beuls, Bart Bogaerts, Gianluca Bontempi, Pierre Geurts, Nick Harley, Bertrand Leblot, Tom Lenaerts, Gilles Louppe, and Paul Van Eecke, editors, *Proceedings of the 31st Benelux Conference on Artificial Intelligence (BNAIC 2019) and the 28th Belgian Dutch Conference on Machine Learning (BeneLearn 2019), Brussels, Belgium, November 6-8, 2019*, volume 2491 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019. URL: <http://ceur-ws.org/Vol-2491/demo96.pdf>.
  - 10 Jens Claes, Bart Bogaerts, Emilio Gamba, Rocsildes Canoy, and Tias Guns. Human-oriented solving and explaining of logic grid puzzles, November 2019. BNAIC 2019 ; Conference date: 07-11-2019 Through 08-11-2019.
  - 11 Broes De Cat, Bart Bogaerts, Maurice Bruynooghe, Gerda Janssens, and Marc Denecker. Declarative logic programming. In Michael Kifer and Yanhong Annie Liu, editors, *Declarative Logic Programming: Theory, Systems, and Applications*, chapter Predicate Logic As a Modeling Language: The IDP System, pages 279–323. Association for Computing Machinery and Morgan & Claypool, New York, NY, USA, 2018. doi:10.1145/3191315.3191321.
  - 12 Jo Devriendt. Exact solver, 2023. URL: <https://gitlab.com/JoD/exact>.
  - 13 William Dumez, Simon Vandeveld, and Joost Vennekens. Step-wise explanations of sudokus using IDP. In *BNAIC/BeNeLearn*, November 2022.
  - 14 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1291–1299. ijcai.org, 2018. doi:10.24963/ijcai.2018/180.
  - 15 Emilio Gamba, Bart Bogaerts, and Tias Guns. Efficiently explaining CSPs with unsatisfiable subset optimization. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1381–1388. International Joint Conferences on Artificial Intelligence Organization, August 2021. Main Track. doi:10.24963/ijcai.2021/191.
  - 16 Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187:52–89, 2012. doi:10.1016/j.artint.2012.04.001.



## 11:18 Simplifying Step-Wise Explanation Sequences

- 17 Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael A. Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In Francesco Bonchi, Foster J. Provost, Tina Eliassi-Rad, Wei Wang, Ciro Cattuto, and Rayid Ghani, editors, *5th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2018, Turin, Italy, October 1-3, 2018*, pages 80–89. IEEE, 2018. doi:10.1109/DSAA.2018.00018.
- 18 Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, volume 235 of *LIPICs*, pages 25:1–25:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.CP.2022.25.
- 19 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.
- 20 Tias Guns, Milan Pesa, Maxime Mulamba, Ignace Bleukx, Emilio Gamba, and Senne Berden. Sudoku assistant – an AI-powered app to help solve pen-and-paper sudokus, 2022.
- 21 Sharmi Dev Gupta, Begum Genc, and Barry O’Sullivan. Explanation in constraint satisfaction: A survey. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4400–4407. ijcai.org, 2021. doi:10.24963/ijcai.2021/601.
- 22 Sharmi Dev Gupta, Begum Genc, and Barry O’Sullivan. Finding counterfactual explanations through constraint relaxations. *CoRR*, abs/2204.03429, 2022. doi:10.48550/arXiv.2204.03429.
- 23 Pieter Van Hertum, Ingmar Dasseville, Gerda Janssens, and Marc Denecker. The KB paradigm and its application to interactive configuration. *Theory Pract. Log. Program.*, 17(1):91–117, 2017. doi:10.1017/S1471068416000156.
- 24 Marijn J. H. Heule. Proofs of unsatisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 635–668. IOS Press, 2021. doi:10.3233/FAIA200998.
- 25 Alexey Ignatiev, Alessandro Previti, Mark H. Liffiton, and João Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 173–182. Springer, 2015. doi:10.1007/978-3-319-23219-5\_13.
- 26 Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI’01 Workshop on Modelling and Solving problems with constraints*, volume 4. Citeseer, 2001.
- 27 Pat Langley, Ben Meadows, Mohan Sridharan, and Dongkyu Choi. Explainable agency for intelligent autonomous systems. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4762–4764. AAAI Press, 2017. URL: <http://aaai.org/ocs/index.php/IAAI/IAAI17/paper/view/15046>.
- 28 Niklas Lauffer and Ufuk Topcu. Human-understandable explanations of infeasibility for resource-constrained scheduling problems. In *ICAPS 2019 Workshop XAIP*, 2019.
- 29 Kevin Leo and Guido Tack. Debugging unsatisfiable constraint models. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 2017. doi:10.1007/978-3-319-59776-8\_7.

- 30 Mark H. Liffiton, Alessandro Previti, Ammar Malik, and João Marques-Silva. Fast, flexible MUS enumeration. *Constraints An Int. J.*, 21(2):223–250, 2016. doi:10.1007/s10601-015-9183-0.
- 31 Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reason.*, 40(1):1–33, 2008. doi:10.1007/s10817-007-9084-z.
- 32 João Marques-Silva. Minimal unsatisfiability: Models, algorithms and applications (invited paper). In *40th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2010, Barcelona, Spain, 26-28 May 2010*, pages 9–14. IEEE Computer Society, 2010. doi:10.1109/ISMVL.2010.11.
- 33 João Marques-Silva. Logic-based explainability in machine learning. In Leopoldo E. Bertossi and Guohui Xiao, editors, *Reasoning Web. Causality, Explanations and Declarative Knowledge - 18th International Summer School 2022, Berlin, Germany, September 27-30, 2022, Tutorial Lectures*, volume 13759 of *Lecture Notes in Computer Science*, pages 24–104. Springer, 2022. doi:10.1007/978-3-031-31414-8\_2.
- 34 Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints An Int. J.*, 14(3):357–391, 2009. doi:10.1007/s10601-008-9064-x.
- 35 Stephen Ostermiller. QQwing. URL: <https://qqwing.com/>.
- 36 Laurent Perron and Vincent Furnon. Or-tools, November 2022. URL: <https://developers.google.com/optimization/>.
- 37 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. URL: <https://www.sciencedirect.com/science/bookseries/15746526/2>.
- 38 Christian Schulte and Peter J. Stuckey. Efficient constraint propagation engines. *ACM Trans. Program. Lang. Syst.*, 31(1):2:1–2:43, 2008. doi:10.1145/1452044.1452046.
- 39 Ilankaikone Senthoran, Matthias Klapperstück, Gleb Belov, Tobias Czauderna, Kevin Leo, Mark Wallace, Michael Wybrow, and Maria Garcia de la Banda. Human-centred feasibility restoration. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPICs*, pages 49:1–49:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CP.2021.49.
- 40 Eric Taillard. Benchmarks for basic scheduling problems. *European journal of operational research*, 64(2):278–285, 1993.

## **A** Satisfying all desirable properties

To transform a maximal sequence generated by CONSTRUCT-GREEDY to one that is *sparse*, *pertinent*, *atomic* and contains no *redundant* (and hence no *loose*) steps, we propose to generate sequences with CONSTRUCT-GREEDY, then filter those with DELETION-BASED FILTERING, finally followed by RELAXATION-BASED FILTERING.

The application of RELAXATION-BASED FILTERING in the last step guarantees sparsity. Modifying the output of steps to guarantee pertinence will not break sparsity as the output of DELETION-BASED FILTERING is maximal (see the argument of Section 5).

After the application of DELETION-BASED FILTERING, all redundant steps have been removed. In other words, for any  $i$ , there exists no strict subsequence  $\langle \mathcal{S}_i \rangle_{1 \leq i \leq n, i \neq j}$  for which its maximal sequence with the same given set  $\mathcal{G}$  explains the same target set  $\mathcal{T}$ . Hence, RELAXATION-BASED FILTERING will not actually filter any more steps, as also shown in the experiments in Section 7. Moreover, RELAXATION-BASED FILTERING does not change any of the constraint subsets  $\mathcal{S}_i$ , so the maximal sequence after RELAXATION-BASED FILTERING is equal to the one after DELETION-BASED FILTERING. Therefore, RELAXATION-BASED FILTERING preserves the guarantee of DELETION-BASED FILTERING that no redundant steps exist.

To argue atomicity, we will consider three sequences

## 11:20 Simplifying Step-Wise Explanation Sequences

- $\langle (\mathcal{E}_i, \mathcal{S}_i, \mathcal{N}_i) \rangle_{1 \leq i \leq n}$  denotes the sequence obtained after construction,
  - $\langle (\mathcal{E}'_i, \mathcal{S}_i, \mathcal{N}'_i) \rangle_{i \text{ not deleted}}$  the sequence obtained after DELETION-BASED FILTERING, and
  - $\langle (\mathcal{E}''_i, \mathcal{S}_i, \mathcal{N}''_i) \rangle_{i \text{ not deleted}}$  the sequence obtained after RELAXATION-BASED FILTERING.
- Moreover, we will write  $\mathcal{I}_i$  (resp.  $\mathcal{I}'_i, \mathcal{I}''_i$ ) for  $\mathcal{G} \cup \bigcup_{j < i} \mathcal{N}_j$  (resp.  $\mathcal{G} \cup \bigcup_{j < i} \mathcal{N}'_j, \mathcal{G} \cup \bigcup_{j < i} \mathcal{N}''_j$ ). Intuitively,  $\mathcal{I}_i$  represents the set of all facts available prior to step  $i$ . By definition of an explanation sequence, we have that  $\mathcal{E}_i \subseteq \mathcal{I}_i$  (and similar for the other two sequences).

To argue that  $(\mathcal{E}''_i, \mathcal{S}_i, \mathcal{N}''_i)$  is still atomic, we first prove four claims.

1.  $\mathcal{I}_i \supseteq \mathcal{I}'_i \supseteq \mathcal{I}''_i$ ;
2.  $\mathcal{N}'_i \subseteq \mathcal{I}_{i+1}$  and  $\mathcal{N}''_i \subseteq \mathcal{I}_{i+1}$
3. For each literal  $l$  in  $\mathcal{N}_i \setminus \mathcal{I}_i$  (i.e., every newly derived literal in step  $i$ ), and every subset  $S \subsetneq \mathcal{S}_i, \mathcal{I}_i \cup S \not\models l$ ;
4. If step  $i$  is not deleted, then  $\mathcal{N}'_i \cap (\mathcal{N}_i \setminus \mathcal{I}_i) \neq \emptyset$  and  $\mathcal{N}''_i \cap (\mathcal{N}_i \setminus \mathcal{I}_i) \neq \emptyset$ , i.e., at least one literal that was newly derived at step  $i$  is still derived at that step.



Claim 1 follows from the fact that, by construction, the first two sequences are maximal (in their inputs as well as outputs), and hence derive everything that is derivable at each point. Since the constraint sets in the three sequences are the same, nothing extra can then be derived in the trimmed sequences. Claim 2 again follows directly from the fact that the initial sequence is maximal: what is derived at step  $i$  can be at most  $\mathcal{I}_i \cup \mathcal{N}_i$ , which equals  $\mathcal{I}_{i+1}$ . Claim 3 holds by construction: our greedy construction algorithm only generates steps for which there is no smaller set of constraints that can derive something. Claim 4 holds since, using Claim 2, if that intersection would be empty,  $\mathcal{N}'_i$  would consist only of literals of  $\mathcal{I}_i$ , in which case DELETION-BASED FILTERING would clearly delete this step.

Now, take any  $i$ . We continue to show that  $(\mathcal{E}''_i, \mathcal{S}_i, \mathcal{N}''_i)$  is in fact atomic. Using Claim 4,  $\mathcal{N}''_i \cap (\mathcal{N}_i \setminus \mathcal{I}_i) \neq \emptyset$ . From Claim 2 it follows that  $\mathcal{N}''_i$  is the union of two sets:  $O := \mathcal{N}''_i \cap \mathcal{I}_i$  and  $D := \mathcal{N}''_i \cap (\mathcal{N}_i \setminus \mathcal{I}_i)$ , where  $O$  is the set of literals that were originally “old” at step  $i$  (derived before  $i$ ) and  $D$  are those that were actually derived at step  $i$ . By Claim 4,  $D$  is non-empty. Now assume towards contradiction that our step  $(\mathcal{E}''_i, \mathcal{S}_i, \mathcal{N}''_i)$  is *not* atomic. In that case, there is an explanation sequence that derives  $\mathcal{N}''_i$  from  $\mathcal{E}''_i$  in which *each* step uses a *strict* subset of  $\mathcal{S}_i$ . From that sequence, consider the first step that derives an element from  $D$ . By definition, the input of that step can consist at most of  $\mathcal{E}''_i \cup O$ . Since  $\mathcal{E}''_i \cup O \subseteq \mathcal{E}_i$ , we find an explanation step that violates Claim 3, and we can conclude our proof by contradiction.

# Towards More Efficient Local Search for Pseudo-Boolean Optimization

Yi Chu  

Institute of Software, Chinese Academy of Sciences, Beijing, China

Shaowei Cai<sup>1</sup>  

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Chuan Luo  

School of Software, Beihang University, Beijing, China

Zhendong Lei  

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Cong Peng  

Finovation in CCBFT, Beijing, China

---

## Abstract

Pseudo-Boolean (PB) constraints are highly expressive, and many combinatorial optimization problems can be modeled using pseudo-Boolean optimization (PBO). It is recognized that stochastic local search (SLS) is a powerful paradigm for solving combinatorial optimization problems, but the development of SLS for solving PBO is still in its infancy. In this paper, we develop an effective SLS algorithm for solving PBO, dubbed *NuPBO*, which introduces a novel scoring function for PB constraints and a new weighting scheme. We conduct experiments on a broad range of six public benchmarks, including three real-world benchmarks, a benchmark from PB competition, an integer linear programming optimization benchmark, and a crafted combinatorial benchmark, to compare *NuPBO* against five state-of-the-art competitors, including a recently-proposed SLS PBO solver *LS-PBO*, two complete PB solvers *PBO-IHS* and *RoundingSat*, and two mixed integer programming (MIP) solvers *Gurobi* and *SCIP*. *NuPBO* has been exhibited to perform best on these three real-world benchmarks. On the other three benchmarks, *NuPBO* shows competitive performance compared to state-of-the-art competitors, and it significantly outperforms *LS-PBO*, indicating that *NuPBO* greatly advances the state of the art in SLS for solving PBO.

**2012 ACM Subject Classification** Theory of computation → Randomized local search

**Keywords and phrases** Pseudo-Boolean Optimization, Stochastic Local Search, Scoring Function, Weighting Scheme

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.12

**Supplementary Material** *Software (Source Code)*: <https://github.com/filyouzicha/NuPBO>

**Funding** This work is supported by the Strategic Priority Research Program of the Chinese Academy of Sciences, Grant No. XDA0320000 and XDA0320300, the National Natural Science Foundation of China under Grant 62302492, and Grant 62202025, the Research Program of CCBFT, Grant No. KT2100040, and CCF-Huawei Populus Grove Fund under Grant CCF-HuaweiSY20231.

---

<sup>1</sup> Corresponding author.



## 1 Introduction

Recent progress in the theory and application of the Boolean satisfiability (SAT) and maximum satisfiability (MaxSAT) problems has led to the development of high-performance complete solvers [11, 23, 1, 32, 2, 4, 31] and stochastic local search (SLS) solvers [8, 28, 7, 6, 27, 25, 5, 45]. SAT and MaxSAT solvers can address challenging problems in a wide variety of fields, and they are usually designed to deal with formulas encoded in conjunctive normal form (CNF).

For problems involving cardinality constraints, CNF solvers usually become ineffective, since expressing such constraints in CNF would dramatically increase the size of the formula and introduce many auxiliary variables and clauses [30]. Linear pseudo-Boolean (PB) constraints provide a more natural and direct way to express cardinality constraints than CNF. Meanwhile, linear PB constraints stay close to CNF and can benefit from advancements in SAT solving [33]. In practice, PB constraints occur in many areas, including VLSI design, economics, computer vision, and manufacturing [43, 44, 33]. The pseudo-Boolean optimization (PBO) problem is to find a satisfying assignment to a set of PB constraints that minimizes a given objective function.

### 1.1 Related Work

Existing pseudo-Boolean solvers are primarily based on complete methods. A number of PB solvers are based on resolution: they express the PB constraints in CNF and then call conflict-driven clause learning (CDCL) solvers, such as *MINISAT+* [13], *Open-WBO* [29], and *NaPS* [34]; alternatively, they deal with the PB constraints but derive new information only in the form of clauses [17]. CDCL is somewhat limited in its reasoning in that it is based on a resolution-proof system, for which exponential lower bounds are known for simple combinatorial principles [20, 15]. Another method requires going beyond resolution and using cutting planes, which can be found in recent PB solvers such as *Sat4j* [24], *RoundingSat* [14, 12] and *RoundingSat-Card* [15]. The success of the implicit hitting set (IHS) method in MaxSAT motivates another work, i.e., the implementation of the *PBO-IHS* solver for solving PBO [36, 37]. In addition, since PB constraints can be considered as 0-1 linear constraints, mixed integer programming (MIP) solvers can be directly applied to solving PBO. Representative and high-performance MIP solvers include *SCIP* [16] - one of the fastest non-commercial solvers, and *Gurobi* [19] - one of the most powerful commercial solvers.

Stochastic local search (SLS) is recognized to be one of the most powerful techniques for solving computationally hard problems in many areas of computer science, operations research, and engineering, and it has shown great success in solving SAT and MaxSAT [22]. In the book [21], a model for local search to solve constraint problems is presented. Somewhat surprisingly, there are only a few research works on using SLS for solving PBO [3, 39, 26].

Since the introduction of dynamic local search (DLS) methods [35, 9, 40, 41], weighting schemes play critical roles in the development of high-performance SLS algorithms. Modern SLS solvers for MaxSAT employ a scoring function defined as the weighted cost of unsatisfied clauses and incorporate a clause weighting scheme to adjust the weights during the search. In particular, most SLS solvers for MaxSAT focus on improving the scoring function through carefully designed weighting schemes. In recent years, the introduction of new weighting schemes has led to breakthroughs in the SLS algorithms for the (weighted) partial MaxSAT ((W)PMS) problem. The newly proposed weighting scheme, Weighting-PMS, in the SATLike [25] algorithm significantly improves the performance of SLS for (W)PMS. Currently, the state-of-the-art SLS algorithms for (W)PMS all employ the Weighting-PMS technique [5, 45].

It is intuitive that the scoring function commonly used in SLS algorithms for MaxSAT (i.e., measuring the total weight of all unsatisfied clauses) does not work well for the PBO problem, because it does not take into account the unsatisfied degree of PB constraints. Indeed, this issue has already attracted attention in the literature, and a scoring function that considers the unsatisfied degree of PB constraints is proposed in [42] and can effectively handle linear PB-constrained problems.

A recent SLS solver called *LS-PBO* [26] has been proposed for PBO and currently represents the state of the art in SLS for PBO. For *LS-PBO*, its scoring function measures the sum of the product between the degree of violation of all unsatisfied constraints and the weights of the constraints. However, such scoring function does not consider the balance between the degree of violation among different constraints. In addition, weighting schemes have not been applied to PBO until the introduction of *LS-PBO*. The weighting scheme in *LS-PBO* resembles the existing one named Weighting-PMS, which aims to increase the weights of unsatisfied constraints and the objective function when the algorithm falls into a local optimum, and to set an upper bound on the weight of the objective function. However, in the context of PBO solving, the research of designing weighting schemes is still in its infancy, which urgently calls for more powerful weighting schemes for PBO.

## 1.2 Contributions

In this work, we focus on improving the performance of SLS for solving PBO. In particular, we propose two main ideas. The first idea is a novel scoring function that considers the violation degree of unsatisfied constraints and utilizes a smooth function to balance the violation degree of different constraints. For each constraint, its smooth function is instantiated as the average of the coefficients of all variables appearing in the corresponding constraint. Since our scoring function is equipped with a weighting scheme, our second idea is a novel weighting scheme for PBO. Rather than setting an upper bound on the weight of the objective function, we adopt a weighting scheme with a stricter condition for updating the weight of the objective function.

On the basis of these two ideas, we develop a new SLS algorithm, named *NuPBO*. We conduct experiments on 6 benchmarks, which include 3 benchmarks encoded from real-world applications, and 3 standard benchmarks. On these 6 benchmarks, *NuPBO* is compared to 5 solvers, including *LS-PBO* [26], *PBO-IHS* [37], *RoundingSat* [12], *Gurobi* [19], and *SCIP* [16]. On the 3 application benchmarks, *NuPBO* achieves improvement over *LS-PBO*, and significantly outperforms other competitors. On the other 3 benchmarks, *NuPBO* exhibits competitive performance compared to its competitors, including the commercial solver *Gurobi*. This represents a significant advance in the research of SLS solvers for PBO. In addition, we evaluate the effectiveness of the underlying ideas on all benchmarks.

## 2 Preliminaries

Given a set of  $n$  Boolean variables  $V = \{x_1, x_2, \dots, x_n\}$ , a literal is either a variable  $x_i$  or its negation  $\neg x_i$ . A clause is a disjunction of literals, i.e.,  $c_i = l_{i_1} \vee l_{i_2} \vee \dots \vee l_{i_k}$ , where  $k$  denotes the length of clause  $c_i$ . A CNF formula  $F$  is a conjunction of clauses. An assignment is a mapping that assigns a Boolean value (*True* (i.e., 1) or *False* (i.e., 0)) to each variable. Given an assignment  $\alpha$ , a clause  $c$  is satisfied if at least one literal in  $c$  is *True*; otherwise,  $c$  is unsatisfied. Given a CNF formula  $F$ , the Boolean satisfiability (SAT) problem is to decide whether an assignment exists such that all clauses are satisfied, and the maximum satisfiability (MaxSAT) problem is to find an assignment that maximizes the number of satisfied clauses.

## 12:4 Towards More Efficient Local Search for Pseudo-Boolean Optimization

A linear pseudo-Boolean constraint (LPB constraint, PB constraint for short) has the following form:

$$\sum_{j=1}^n a_j l_j \triangleright b, \quad a_j, b \in \mathbb{Z} \quad (1)$$

where  $b$  is called the degree of the constraint,  $l_j$  is a literal,  $a_j$  is the coefficient of  $l_j$ ,  $n$  is the length of the constraint,  $\triangleright$  is one of the classical relational operators ( $=, >, \geq, < \text{ or } \leq$ ), and  $\mathbb{Z}$  is the integer set.

For each Boolean variable,  $x_i = 1 - \neg x_i$ . It is important to note that for an equality constraint, there need to be two normalized constraints to represent it. Therefore, all PB constraints can be normalized into the following form:

$$\sum_{j=1}^n a_j l_j \geq b, \quad a_j, b \in \mathbb{N}_0^+ \quad (2)$$

where  $\mathbb{N}_0^+$  is the non-negative integer set [33].

In the following sections, we assume that the PB constraints are of the normalized form. A PB formula  $F$  is a conjunction of PB constraints. An assignment is a mapping that assigns a Boolean value to each variable. Given an assignment  $\alpha$ , a PB constraint  $c$  is satisfied if the corresponding inequality holds under  $\alpha$ ; otherwise,  $c$  is unsatisfied. If an assignment  $\alpha$  satisfies all constraints in  $F$ , then we say  $\alpha$  is a feasible solution (or solution for short).

A pseudo-Boolean optimization (PBO) instance consists of a PB formula  $F$  and a linear Boolean objective function  $\sum_{j=1}^n e_j l_j + d$ ,  $e_j \in \mathbb{N}^+, d \in \mathbb{Z}$ , and the task is to find an assignment that satisfies all PB constraints in  $F$  and minimizes the objective function. Given an assignment  $\alpha$ , we use  $obj(\alpha)$  to denote the value of the objective function. Given a solution  $\alpha$ , the cost of the solution  $\alpha$  is equal to  $obj(\alpha)$ . We say a solution  $\alpha_1$  is better than another solution  $\alpha_2$ , if  $obj(\alpha_1) < obj(\alpha_2)$ .

The average coefficient of a PB constraint  $c$  is denoted as  $avg_{coe}(c) = (\sum_{j=1}^n a_j)/n$ . The average coefficient of an objective function  $o$  is denoted as  $avg_{coe}(o) = (\sum_{j=1}^n e_j)/n$ . Because PB constraints must be satisfied in a PBO problem, the PB constraints are referred to as *hard constraints*.

Given an assignment  $\alpha$ , we define the value of violation of a hard constraint  $c$  as

$$viol(c) = \max \left( 0, b - \sum_{j=1}^n a_j l_j \right)$$

In other words, if the hard constraint  $c$  is satisfied under  $\alpha$ , then  $viol(c) = 0$ ; otherwise (i.e.,  $c$  is unsatisfied),  $viol(c)$  is the integer distance of  $c$  from being satisfied. In a PBO instance, a solution is an assignment, under which all hard constraints are satisfied.

The SLS algorithms that employ constraint weighting schemes usually maintain weight for each constraint. We use  $w(c)$  to represent the weight of each hard constraint  $c$ , and  $w(o)$  to represent the weight of the objective function  $o$ .

### 3 Main Ideas

In general, the search directions of SLS algorithms are guided by the scoring function. It is recognized that the effectiveness of the scoring function could be enhanced through working with a weighting scheme. In this section, we first propose a new scoring function, and then design a new weighting scheme to work with it.

■ **Table 1** The violation and objective value under all assignments of the PBO instance  $I_1$  in Example 3.1.

$viol/obj$	Assignment $(x_1, x_2, x_3)$							
	(0, 0, 0)	(0, 0, 1)	(0, 1, 0)	(0, 1, 1)	(1, 0, 0)	(1, 0, 1)	(1, 1, 0)	(1, 1, 1)
$viol(c_1)$	0	0	0	0	4	3	3	2
$viol(c_2)$	1	0	0	0	4	2	3	1
$viol(c_3)$	29	13	14	0	0	0	0	0
$obj(\alpha)$	1	1	0	0	2	2	1	1

### 3.1 A New Scoring Function

Given a PBO instance, which consists of  $n$  PB constraints (hard constraints) and one objective function, we assume that the current assignment is  $\alpha$ .

**Scoring Function in *LS-PBO*.** Before introducing our new scoring function, we first describe the existing scoring function proposed in *LS-PBO* [26], which is presented as follows.

- For a hard constraint  $c$ , the penalty function is defined as  $penalty(c) = w(c) \times viol(c)$ ; then the hard score of a variable  $x$  is defined as the decrement of the sum of the penalty function values of all hard constraints caused by flipping  $x$ , which is denoted by  $hscore(x)$ .
- For the objective function  $o$ , the value of the objective function is  $obj(\alpha)$ , and the penalty function is defined as  $penalty(o) = w(o) \times obj(\alpha)$ ; then the soft score of a variable  $x$  is defined as the decrement of the penalty function value of the objective function caused by flipping  $x$ , which is denoted by  $sscore(x)$ .
- The score of a variable  $x$  is defined as  $score(x) = hscore(x) + sscore(x)$ .

**An Intuitive View.** The above scoring function has a drawback, which is due to its underlying penalty function. The penalty function above considers the weights and  $viol$  (resp.  $obj$ ) values of hard (resp. soft) constraints. In this way, it measures the importance of a variable in a constraint  $c$  by the coefficient of the corresponding variable in  $c$ . Nevertheless, it should be noted that, as the value of  $score(x)$  is determined by all hard constraints and the objective function involving  $x$ , the above penalty function may overemphasize the importance of  $x$  in constraints with relatively large coefficients, resulting in an unreasonable value of its  $score$ . In the following, we illustrate our intuition through a simple PBO instance.

► **Example 3.1.** Let us consider a PBO instance  $I_1$ , which consists of three hard constraints and an objective function:  $c_1 : 4\neg x_1 + x_2 + x_3 \geq 4$ ,  $c_2 : 3\neg x_1 + x_2 + 2x_3 \geq 4$ ,  $c_3 : 29x_1 + 15x_2 + 16x_3 \geq 29$ , minimize:  $x_1 + \neg x_2$ . The values of  $viol$  of hard constraints and the value of  $obj$  of the objective function under all assignments are presented in Table 1. Solutions for  $I_1$  are those resulting in the zero value of  $viol$  for all hard constraints. From Table 1, the optimal solution is  $\alpha^* = (0, 1, 1)$ , and  $obj(\alpha^*)$  is 0.

Given instance  $I_1$ , consider a scoring function without any weighting scheme, or equivalently, the weight of each constraint is 1, i.e.,  $w(c_1) = 1$ ,  $w(c_2) = 1$ ,  $w(c_3) = 1$ ,  $w(o) = 1$ ; the initial assignment  $\alpha = (0, 0, 0)$ . In accordance with the definition of the scoring function in *LS-PBO*,  $score(x_1) = 21$ ,  $score(x_2) = 17$ , and  $score(x_3) = 17$ . Actually, in order to optimize the assignment, SLS algorithms tend to select the variable to be flipped as the one with the largest score, so in this situation,  $x_1$  is picked. After flipping  $x_1$ , the assignment becomes  $\alpha = (1, 0, 0)$ , and the  $score$  value of each variable becomes  $score(x_1) = -21$ ,  $score(x_2) = 3$ ,  $score(x_3) = 3$ . Then, no matter whether  $x_2$  or  $x_3$  is flipped, the Hamming distance between



the current assignment and the optimal solution is the same as that between the initial assignment and the optimal solution, which is two. The search is not progressing in the direction towards the optimal solution.

In practice, PBO instances encoded from real-world problems are much more complex than the given illustrative example. If SLS algorithms conduct the search in incorrect directions, it would be difficult to identify a promising search space that is more likely to contain the optimal solution or those close to optimality.

As presented in Table 1, when we focus on the value of  $viol(c_3)$ , for those cases where the value of  $viol(c_3)$  is not 0, its value is much larger than the  $viol$  value of other hard constraints. Considering that each hard constraint has a *penalty* value directly proportional to its  $viol$  value, utilizing scoring function aims to guide the search towards the area with a lower sum of *penalty* values. Consequently, through making use of such scoring function, the algorithms would prefer the falsified literal with the largest coefficient to be *True* (in instance  $I_1$ , under the assumption that the current assignment is  $(0, 0, 0)$ , this falsified literal is  $x_1$  in the hard constraint  $c_3$ ).

**Our New Scoring Function.** In our opinion, a good scoring function for PBO should balance the  $viol$  values of different constraints. To this end, we propose to smooth the *penalty* values of constraints. For simplicity, we denote the smoothing function of a hard constraint  $c$  as  $smooth(c)$ , and the smoothing function of the objective function  $o$  as  $smooth(o)$ . Based on the idea of balancing the  $viol$  value, we propose the following, new scoring function:

- **For a hard constraint  $c$ , the penalty function is defined as  $penalty(c) = \frac{w(c) \times viol(c)}{smooth(c)}$** ; then the hard score of a variable  $x$  is defined as the decrement of the sum of the penalty function values of all hard constraints caused by flipping  $x$ , which is denoted by  $hscore(x)$ .
- **For the objective function  $o$ , the value of the objective function is  $obj(\alpha)$ , and the penalty function is defined as  $penalty(o) = \frac{w(o) \times obj(\alpha)}{smooth(o)}$** ; then the soft score of a variable  $x$  is defined as the decrement of the penalty function value of the objective function caused by flipping  $x$ , which is denoted by  $sscore(x)$ .
- The score of a variable  $x$  is defined as  $score(x) = hscore(x) + sscore(x)$ .

In order to instantiate the above scoring function, we propose to use a method for smoothing by using the average of the constraint coefficients, i.e.,  $smooth(c) = round(avg_{coe}(c))$ ,  $smooth(o) = round(avg_{coe}(o))$  ( $round$  is a rounding function). Consider the PBO instance  $I_1$  in Example 3.1, which has  $smooth(c_1) = 2$ ,  $smooth(c_2) = 2$ ,  $smooth(c_3) = 20$ , and  $smooth(o) = 1$ . Assume that each hard constraint and the objective function have a weight of 1 and the current assignment  $\alpha = (0, 0, 0)$ . Based on the new scoring function,  $score(x_1) = -3.05$ ,  $score(x_2) = 2.25$ , and  $score(x_3) = 1.3$ . Hence, SLS algorithms would select variable  $x_2$  to be flipped. Flipping  $x_2$  would change the current assignment  $\alpha$  to  $(0, 1, 0)$ , and the  $score$  value of each variable would become  $score(x_1) = -3.3$ ,  $score(x_2) = -2.25$ ,  $score(x_3) = 0.7$ . Afterward, SLS algorithms would select variable  $x_3$  to be flipped. If  $x_3$  is flipped, assignment  $\alpha$  becomes  $(0, 1, 1)$ , which is the optimal solution of instance  $I_1$ .

According to this illustrative example, it can be observed that, by using the average of constraint coefficients to smooth the *penalty* value, the issue of the large difference among coefficients of a variable in various constraints can be alleviated, resulting in a more effective scoring function.

### 3.2 A New Weighting Scheme

Combinatorial optimization problems with both hard and soft constraints require effective weighting schemes that balance the weights of hard and soft constraints. A potential problem was pointed out in a previous study [10]: the excessive weight given to soft constraints may make it difficult to satisfy all the hard constraints, thereby hindering the algorithm's capability of finding solutions. Moreover, an existing study [38] demonstrates that designing a weighting scheme for problems with hard constraints is challenging as it requires weighting unsatisfied constraints while maintaining the distinction between hard and soft constraints.

To alleviate the above problem, the weighting scheme proposed in *LS-PBO* sets an upper bound  $\zeta$  (an integer parameter) to the maximum value of the objective function weight. We use *unsat\_hard\_set* to denote the set of unsatisfied hard constraints. For a PBO instance  $I$ , the average of the product of the  $avg_{coe}(c)$  and  $w(c)$  of all constraints  $c$  is denoted as  $wavg_{coe}(I)$ , that is,  $wavg_{coe}(I) = (\sum_{i=1}^m (avg_{coe}(c_i) \times w(c_i))) / m$ . Assuming that the current assignment is  $\alpha$ , the best solution that has been found is  $\alpha^*$ , and its corresponding objective function value is  $obj(\alpha^*)$ .

The weighting scheme adopted in *LS-PBO* is described as follows:

- Initialization phase: at the start of the local search process, the weight of each hard constraint  $c$  is initialized as 1, i.e.,  $w(c) := 1$ ; the weight of the objective function  $o$  is also initialized as 1, i.e.,  $w(o) := 1$ .
- Update phase: when the search is trapped in a local optimum (i.e., there is no variable whose *score* value is greater than 0), for each  $c$  in *unsat\_hard\_set*,  $w(c) := w(c) + 1$ ; if  $obj(\alpha) \geq obj(\alpha^*)$  and  $w(o) \times avg_{coe}(o) - wavg_{coe}(I) \leq \zeta$ ,  $w(o) := w(o) + 1$ , where  $\zeta$  is a parameter introduced by *LS-PBO*.

In fact, the setting of  $\zeta$  greatly affects the performance of *LS-PBO*, and if the average of the coefficients of the objective function is much greater than the average of the coefficients of the hard constraints, the weight of the objective function basically would not be updated. In addition, varying the timing of weighting constraints could be a promising strategy to improve the performance of the weighting scheme.

We propose to deal with these problems by modifying the condition of updating weights. Specifically, we propose a stricter condition for increasing objective function weight. Our proposed weighting scheme is as follows:

- Initialization phase: at the start of the local search process, the weight of each hard constraint  $c$  is initialized as 1, i.e.,  $w(c) := 1$ ; **the weight of the objective function  $o$  is initialized as 0, i.e.,  $w(o) := 0$ .**
- Update phase: when the search is trapped in a local optimum (i.e., there is no variable whose *score* value is greater than 0), for each  $c$  in *unsat\_hard\_set*,  $w(c) := w(c) + 1$ ; **if *unsat\_hard\_set* is empty,  $w(o) := w(o) + 1$ .**

In the beginning, the weight of the objective function is initialized as 0, so that the algorithm would first focus on finding solutions. If the search is trapped in a local optimum, the weight of the objective function is increased only when the current assignment  $\alpha$  is a solution (all hard constraints are satisfied under  $\alpha$ ). Accordingly, if the algorithm frequently visits solutions, then the objective function would have a greater chance to increase its weight. Otherwise, there would be limited opportunities to increase the objective function weight.

■ **Algorithm 1** The *NuPBO* Algorithm.

---

**Input:** A PBO instance  $I$ , *cutoff time*.  
**Output:** The best solution ( $\alpha^*$ ) found and its objective function value  $obj^*$ , or “No solution found”.

```

1  $\alpha^* := \emptyset$ ;  $obj^* := +\infty$ ;
2 while no terminating criteria are met do
3    $\alpha :=$  an initial assignment;
4   for each  $c$  in hard constraints do
5      $w(c) := 1$ ;
6   for the objective function  $o$ ,  $w(o) := 0$ ;
7    $L := 10000000$ ;
8   for  $step = 0$ ;  $step < L$ ;  $step++$  do
9     if  $\alpha$  is feasible and  $obj^* > obj(\alpha)$  then
10       $\alpha^* := \alpha$ ;  $obj^* := obj(\alpha)$ ;  $L := step + 10000000$ ;
11      if  $D := \{x \mid score(x) > 0\} \neq \emptyset$  then
12         $v :=$  a variable in  $D$  with the highest score;
13      else
14        update constraints weights by the new weighting scheme described in Section 3.2;
15        if  $\exists$  unsatisfied hard constraints then
16           $c :=$  a random unsatisfied hard constraint;
17           $v :=$  the variable whose literal is false with highest score in  $c$ ;
18        else
19           $v :=$  a randomly chosen variable with score  $> 0$ ;
20       $\alpha := \alpha$  with  $v$  flipped;
21 if  $\alpha^* \neq \emptyset$  then return  $\alpha^*$  and  $obj^*$ ;
22 else return No solution found;

```

---

#### 4 The *NuPBO* Algorithm

In this section, we develop a new SLS algorithm named *NuPBO*, which is based on the main ideas proposed in Section 3. *NuPBO* adopts the Dynamic Local Search (DLS) framework as does *LS-PBO*. The pseudo-code of *NuPBO* is outlined in Algorithm 1. We use  $\alpha^*$  and  $obj^*$  to denote the best-found solution and the corresponding objective function value (i.e., the cost of the best-found solution), while  $\alpha$  denotes the current assignment which is maintained during the search.

In the beginning,  $\alpha^*$  is initialized as an empty set, and  $obj^*$  is initialized as  $+\infty$  (Line 1). *NuPBO* then iteratively calls the local search process until reaching a terminating criterion (e.g., reaching a preset cutoff time, or achieving a feasible assignment  $\alpha$  whose corresponding  $obj$  value is equal to 0) (Lines 2–20).

In the local search process, an initial assignment is generated by assigning each Boolean variable to a default value 0 (as in *LS-PBO*) (Line 3). *NuPBO* then initializes the weights of all hard constraints as 1, and sets the weight of objective function to 0 according to our proposed weighting scheme. After initialization, *NuPBO* conducts the search process (Lines 8–20). During the search process, whenever *NuPBO* finds a solution whose  $obj$  is lower than  $obj^*$ , then  $\alpha^*$  and  $obj^*$  are updated accordingly.

In each search step, *NuPBO* selects a variable and flips it based on two situations: (I) If the set  $D$  of decreasing variables (i.e., the variable  $x$  with  $score(x) > 0$ ) is not empty, a variable with the highest score is selected from  $D$ , breaking ties by preferring the variable

that has been flipped least recently. (II) When  $D$  is empty, which indicates that the search is trapped in a local optimum, then *NuPBO* updates the weights of constraints according to our new weighting scheme. Then, if there exist unsatisfied hard constraints, an unsatisfied hard constraint  $c$  is randomly picked. As we know, for a MaxSAT instance, if a clause  $cm$  is unsatisfied, then it would become satisfied after flipping any variables in  $cm$ . For a PBO instance, if a hard constraint  $cp$  is unsatisfied (i.e.,  $viol(cp) > 0$ ), only flipping variables whose literals are *False* under the current assignment can reduce  $viol(cp)$  (Assuming under the current assignment  $\alpha$ ,  $x_1$  is 1, then the literal  $x_1$  is *True*, while the literal  $\neg x_1$  is *False*). Therefore, *NuPBO* picks the variable whose literal is *False* with the highest *score* in  $c$ . Otherwise (i.e., all the hard constraints are satisfied), *NuPBO* randomly chooses a variable whose *sscore* is greater than 0.

Finally, when any terminating criterion is met, *NuPBO* stops and reports the best solution  $\alpha^*$  and  $obj^*$  if a solution is found; otherwise, it reports “No solution found”.

## 5 Experimental Evaluations

In this section, we introduce experimental preliminaries and then conduct extensive experiments on 6 PBO benchmarks. First, we compare *NuPBO* with 5 state-of-the-art PBO solvers. Second, we conduct experiments to show that combining *NuPBO* with complete solvers can lead to better portfolios. Third, we report experimental results to demonstrate the effectiveness of our main ideas. Finally, we examine the stability of the SLS solvers by running each SLS solver 10 times with seeds ranging from 1 to 10.

### 5.1 Experimental Preliminaries

**Benchmarks.** We evaluate *NuPBO* on 6 benchmarks, which are described as follows:

- **PB16:** the OPT-SMALLINT-LIN benchmark from the latest 2016 pseudo-Boolean competition. As a mainstream benchmark for evaluating the performance of PBO solvers, it consists of 1600 instances of various categories.<sup>2</sup>
- **MIPLIB:** 0-1 integer linear programming optimization problems. This benchmark contains 291 instances of various categories, provided in the literature [12].<sup>3</sup>
- **CRAFT:** crafted combinatorial benchmarks whose coefficients are small integers. This benchmark contains 955 instances of various categories, provided in the literature [12].<sup>4</sup>
- **MWCB:** the Minimum-Width Confidence Band Problem. This benchmark contains 24 instances.
- **SAP:** the Seating Arrangements Problem. This benchmark contains 21 instances.
- **WSNO:** the Wireless Sensor Network Optimization Problem. This benchmark consists of 18 instances.

For the benchmarks of **MWCB**, **SAP**, and **WSNO**, the descriptions, the downloading websites, and the methods of converting the real-world applications into PBO instances and the encoded PBO instances are presented in the literature [26].<sup>5</sup>

<sup>2</sup> <http://www.cril.univ-artois.fr/PB16/bench/PB16-used.tar>

<sup>3</sup> <https://zenodo.org/record/3870965>

<sup>4</sup> <https://zenodo.org/record/4036016>

<sup>5</sup> <https://lcs.ios.ac.cn/%7ecaisw/Resource/LS-PBO/>

**State-of-the-Art Competitors.** We compare *NuPBO* with 5 state-of-the-art solvers, including one SLS solver (i.e., *LS-PBO*) and 4 complete solvers. The 4 complete solvers include 2 PB solvers (i.e., *PBO-IHS* and *RoundingSat*) and 2 MIP solvers (i.e., *Gurobi* and *SCIP*):

- *LS-PBO* [26]: the state-of-the-art SLS algorithm for solving PBO. Adopt the parameter setting recommended by its authors. It outperforms *Gurobi* and *RoundingSat* on many real-world application benchmarks.<sup>5</sup>
- *PBO-IHS* [37]: a recent IHS PBO solver building upon *RoundingSat* [14].<sup>6</sup>
- *RoundingSat* [12]: a recent PBO solver combining core-guided search with cutting planes reasoning.<sup>7</sup>
- *Gurobi* [19]: one of the most powerful commercial MIP solvers (Version 9.1.2). The default configuration is used, along with a single thread.<sup>8</sup>
- *SCIP* [16]: one of the fastest non-commercial solvers for MIP (Version 7.0.3, using *SoPlex* 5.0.2 as its internal LP solver).<sup>9</sup>

**Experimental Setup.** *LS-PBO* and *NuPBO* are implemented in C++, and compiled with g++ (version 8.5.0) using the option “-O3”. Installation procedures for other solvers follow their detailed guidelines. All the experiments are carried out on a workstation under the operating system CentOS, with the AMD EPYC7702 2.0GHz CPU.

In these experiments, we adopt two cutoff times of 300 CPU seconds (300s) and 3600 CPU seconds (1h). Each solver performs one run within a given cutoff time on each instance, and we record the cost of the best solution found by solver  $S_j$  on instance  $I_k$ , denoted as  $sol_{S_j I_k}$ . The cost of the best solution found among all solvers in the same table within the same cutoff time on instance  $I_k$  is denoted as  $best_{I_k}$ . For each solver  $S$  solving a benchmark  $B_i$  within a cutoff time, we use 3 *metrics* to evaluate the performance of  $S$ .

- *#win.*: the number of instances where the corresponding  $best_{I_k}$  can be obtained by solver  $S$  on  $B_i$  (i.e., the number of winning instances).
- *avg<sub>score</sub>*: in our experiments, the competition score of solver  $S_j$  on instance  $I_k$  is represented by  $score_{S_j I_k} = \frac{best_{I_k} + 1}{sol_{S_j I_k} + 1}$ , which measures the gap between  $sol_{S_j I_k}$  and  $best_{I_k}$ . If solver  $S_j$  could not report a solution on instance  $I_k$ , then  $score_{S_j I_k} = 0$ . We use *avg<sub>score</sub>* to denote the average competition score of a solver on a benchmark. The competition score of each solver on each instance is the metric to measure the performance of solvers in the incomplete track of recent MaxSAT Evaluations (2017-2023).
- *#feas.*: the number of instances where solver  $S$  obtains solutions on  $B_i$ .

In our experiments, *avg<sub>score</sub>* is calculated by ignoring the instances that are proven to have no solution by the complete solvers. Based on the preliminary experiments, we conclude that at least 123 instances in the PB16 benchmark and at least 17 instances in the MIPLIB benchmark do not have solutions. All instances in the CRAFT, MWCB, SAP, and WSNO benchmarks have solutions.<sup>10</sup>

The number of instances in each benchmark is indicated by ‘#inst.’. For each of the above three metrics, if a solver obtains a larger metric value on a benchmark, then the solver exhibits better performance on the benchmark. The results highlighted in **bold** indicate the best performance for the corresponding metric.

<sup>6</sup> <https://bitbucket.org/coreo-group/pbo-ihs-solver/>

<sup>7</sup> <https://doi.org/10.5281/zenodo.4043124>

<sup>8</sup> <https://www.gurobi.com/products/gurobi-optimizer/>

<sup>9</sup> <https://www.scipopt.org/index.php#download>

<sup>10</sup> Note that the definition of the competition *score* (metric) in this subsection has no relationship to the definition of the *score* in the scoring function in subsection 3.1.

■ **Table 2** Experimental results of *NuPBO* and all the competitors on all the benchmarks (top: cutoff 300s, bottom: cutoff 1h) (Benc, i.e., Benchmark. *avg<sub>s</sub>*, i.e., *avg<sub>score</sub>*).

Benc	#inst.	SLS solvers				PB solvers				MIP solvers			
		<i>NuPBO</i>		<i>LS-PBO</i>		<i>PBO-IHS</i>		<i>RoundingSat</i>		<i>Gurobi</i>		<i>SCIP</i>	
		#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>
<i>cutoff 300s</i>													
PB16	1600	1049 1351	0.8800	700 1183	0.7516	930 <b>1401</b>	0.8712	964 1392	0.8916	<b>1158</b> 1365	<b>0.9011</b>	887 1244	0.7918
MIPLIB	291	130 242	<b>0.8480</b>	75 222	0.7375	84 230	0.7350	89 <b>245</b>	0.7855	<b>166</b> 235	0.8013	114 221	0.7360
CRAFT	955	<b>894</b> 943	0.9868	816 930	0.9682	809 930	0.9433	825 936	0.9639	893 <b>955</b>	<b>0.9961</b>	764 921	0.9583
MWCB	24	<b>24</b> <b>24</b>	<b>1.0000</b>	0 <b>24</b>	0.9448	0 19	0.4496	0 <b>24</b>	0.6247	0 12	0.4004	0 3	0.0761
SAP	21	<b>21</b> <b>21</b>	<b>1.0000</b>	0 <b>21</b>	0.9750	0 0	0.0000	0 0	0.0000	0 1	0.0395	0 0	0.0000
WSNO	18	<b>18</b> <b>18</b>	<b>1.0000</b>	11 <b>18</b>	0.9026	2 5	0.1738	4 <b>18</b>	0.6624	4 5	0.2431	0 6	0.1631
<i>cutoff 1h</i>													
PB16	1600	1064 1360	0.8897	814 1278	0.8164	990 <b>1412</b>	0.8875	1012 1401	0.9100	<b>1229</b> 1396	<b>0.9354</b>	1012 1304	0.8565
MIPLIB	291	127 243	0.8519	83 231	0.7673	100 239	0.7903	87 247	0.8099	<b>199</b> <b>253</b>	<b>0.9023</b>	142 238	0.8104
CRAFT	955	891 <b>955</b>	<b>0.9992</b>	844 932	0.9714	886 947	0.9841	873 <b>955</b>	0.9902	<b>930</b> <b>955</b>	0.9987	824 930	0.9704
MWCB	24	<b>23</b> <b>24</b>	<b>0.9998</b>	1 <b>24</b>	0.9690	0 <b>24</b>	0.5620	0 <b>24</b>	0.7116	0 <b>24</b>	0.7437	0 17	0.5058
SAP	21	<b>21</b> <b>21</b>	<b>1.0000</b>	0 <b>21</b>	0.9785	0 0	0.0000	0 0	0.0000	0 1	0.0451	0 0	0.0000
WSNO	18	<b>18</b> <b>18</b>	<b>1.0000</b>	15 <b>18</b>	0.9985	5 14	0.5989	11 <b>18</b>	0.8660	4 13	0.4904	0 4	0.0842

## 5.2 Comparisons with State-of-the-Art Solvers

The comparative results of *NuPBO* and all the competitors on all the benchmarks are shown in Table 2. We first analyze the results with a cutoff time of 300s.

- In terms of the number of winning instances, *NuPBO* gives the best performance on 4 benchmarks, including CRAFT and the 3 real-world application benchmarks, and ranked second on the PB16 and MIPLIB benchmarks (with *Gurobi* being the best).
- In terms of *avg<sub>score</sub>*, *NuPBO* outperforms all the competitors on 4 benchmarks, including MIPLIB and the 3 real-world application benchmarks. On the PB16 benchmark, its *avg<sub>score</sub>* ranks third after *Gurobi* and *RoundingSat*. The *avg<sub>score</sub>* value of *NuPBO* ranks second on the CRAFT benchmark behind *Gurobi*.
- In terms of #feas., *NuPBO* and *LS-PBO* find solutions for all instances in the 3 real-world application benchmarks, while *PBO-IHS*, *RoundingSat*, and *Gurobi* are respectively the best for finding solutions on PB16, MIPLIB, and CRAFT. Although the value of #feas. of *NuPBO* ranks second on MIPLIB and CRAFT benchmarks, and ranks fourth on PB16 benchmark, *NuPBO* performs considerably better than *LS-PBO*, an SLS solver for PBO.

■ **Table 3** Experimental results of  $VBS_{all}$ ,  $VBS_{exclude\_lspbo}$ , and  $VBS_{exclude\_nupbo}$  on all the benchmarks (top: cutoff 300s, bottom: cutoff 1h).

Benchmark	#inst.	$VBS_{all}$			$VBS_{exclude\_lspbo}$			$VBS_{exclude\_nupbo}$		
		#win.	avg <sub>score</sub>	#feas.	#win.	avg <sub>score</sub>	#feas.	#win.	avg <sub>score</sub>	#feas.
<i>cutoff 300s</i>										
PB16	1600	<b>1434</b>	<b>0.9709</b>	<b>1434</b>	1430	0.9703	<b>1434</b>	1356	0.9690	1433
MIPLIB	291	<b>254</b>	<b>0.9270</b>	254	<b>254</b>	<b>0.9270</b>	254	218	0.9119	254
CRAFT	955	<b>955</b>	<b>1.0000</b>	955	<b>955</b>	<b>1.0000</b>	955	939	0.9999	955
MWCB	24	<b>24</b>	<b>1.0000</b>	24	<b>24</b>	<b>1.0000</b>	24	0	0.9448	24
SAP	21	<b>21</b>	<b>1.0000</b>	21	<b>21</b>	<b>1.0000</b>	21	0	0.9750	21
WSNO	18	<b>18</b>	<b>1.0000</b>	18	<b>18</b>	<b>1.0000</b>	18	11	0.9032	18
<i>cutoff 1h</i>										
PB16	1600	<b>1440</b>	<b>0.9749</b>	<b>1440</b>	1438	0.9747	<b>1440</b>	1394	0.9730	1438
MIPLIB	291	<b>262</b>	<b>0.9562</b>	262	<b>262</b>	<b>0.9562</b>	262	238	0.9492	262
CRAFT	955	<b>955</b>	<b>1.0000</b>	955	<b>955</b>	<b>1.0000</b>	955	953	>0.9999	955
MWCB	24	<b>24</b>	<b>1.0000</b>	24	23	0.9998	24	1	0.9690	24
SAP	21	<b>21</b>	<b>1.0000</b>	21	<b>21</b>	<b>1.0000</b>	21	0	0.9785	21
WSNO	18	<b>18</b>	<b>1.0000</b>	18	<b>18</b>	<b>1.0000</b>	18	15	0.9985	18

With the cutoff time of 1h,  $NuPBO$  outperforms  $LS-PBO$  on the 3 real-world application benchmarks. On the other 3 benchmarks,  $NuPBO$  shows competitive performance compared to  $Gurobi$ , and significantly outperforms the state-of-the-art SLS solver  $LS-PBO$  in terms of all metrics of #win., avg<sub>score</sub>, and #feas..

### 5.3 Complementarity between SLS Solvers and Complete Solvers

In this subsection, we conduct experiments to investigate the complementarity between SLS solvers and complete solvers when solving PBO.

To investigate the complementarity between SLS solvers and complete solvers, we construct three perfect portfolio selectors: given a set of base solvers  $\Theta$ , for each instance, the solution of the perfect portfolio selector constructed on  $\Theta$  is the best among the entire collection of solutions reported by all solvers in  $\Theta$ . These three perfect portfolio selectors are built based on  $\Theta_1 = \{LS-PBO, NuPBO, PBO-IHS, RoundingSat, Gurobi, SCIP\}$ ,  $\Theta_2 = \{NuPBO, PBO-IHS, RoundingSat, Gurobi, SCIP\}$ , and  $\Theta_3 = \{LS-PBO, PBO-IHS, RoundingSat, Gurobi, SCIP\}$  dubbed  $VBS_{all}$ ,  $VBS_{exclude\_lspbo}$  and  $VBS_{exclude\_nupbo}$ , respectively. Then we conduct experiments to evaluate the performance of these three perfect portfolio selectors on all benchmarks. The related results are presented in Table 3.

The comparison between  $VBS_{all}$  and  $VBS_{exclude\_lspbo}$  reveals the number of instances where only the LS-PBO solver can achieve the optimal solution among all solvers. Similarly, comparing  $VBS_{all}$  and  $VBS_{exclude\_nupbo}$  shows the number of instances where only the NuPBO solver can obtain the optimal solution among all solvers. As shown in Table 3, taking the PB16 benchmark with a cutoff time of 300 seconds as an example, out of the 1600 instances, there are 4 instances where only LS-PBO can achieve the optimal solution among all solvers, and there are 78 instances where only NuPBO can obtain the optimal solution among all solvers. Additionally, in 1 instance, only NuPBO was able to find a feasible solution. The results in Table 3 demonstrate that, compared to the state-of-the-art SLS solver  $LS-PBO$ ,  $NuPBO$  is able to enhance the complementarity between SLS solvers and complete solvers, which indicates that a portfolio selector, which combines  $NuPBO$  and complete solvers, could advance the state of the art in PBO solving.

■ **Table 4** Experimental results of *NuPBO*, *NuPBO-alt(s)*, and *NuPBO-alt(w)* on all the benchmarks (top: cutoff 300s, bottom: cutoff 1h).

Benchmark	#inst.	<i>NuPBO</i>			<i>NuPBO-alt(s)</i>			<i>NuPBO-alt(w)</i>		
		#win.	avg <sub>score</sub>	#feas.	#win.	avg <sub>score</sub>	#feas.	#win.	avg <sub>score</sub>	#feas.
<i>cutoff 300s</i>										
PB16	1600	<b>1141</b>	<b>0.9026</b>	<b>1351</b>	1024	0.8288	1253	1034	0.8822	1323
MIPLIB	291	<b>182</b>	<b>0.8687</b>	<b>242</b>	147	0.8347	<b>242</b>	112	0.8312	239
CRAFT	955	<b>941</b>	<b>0.9874</b>	<b>943</b>	<b>941</b>	0.9864	942	848	0.9648	925
MWCB	24	<b>24</b>	<b>1.0000</b>	24	0	0.9466	24	0	0.9700	24
SAP	21	<b>13</b>	<b>0.9990</b>	21	9	0.9974	21	0	0.9782	21
WSNO	18	17	0.9995	18	<b>18</b>	<b>1.0000</b>	18	15	0.9704	18
<i>cutoff 1h</i>										
PB16	1600	<b>1170</b>	<b>0.9076</b>	<b>1360</b>	1113	0.8842	1329	1090	0.8911	1336
MIPLIB	291	<b>188</b>	<b>0.8754</b>	243	149	0.8520	244	121	0.8620	<b>245</b>
CRAFT	955	<b>953</b>	<b>&gt;0.9999</b>	<b>955</b>	<b>953</b>	<b>&gt;0.9999</b>	<b>955</b>	864	0.9683	926
MWCB	24	<b>24</b>	<b>1.0000</b>	24	0	0.9459	24	0	0.9655	24
SAP	21	11	0.9986	21	<b>13</b>	<b>0.9988</b>	21	0	0.9824	21
WSNO	18	18	1.0000	18	18	1.0000	18	18	1.0000	18

## 5.4 Analysis on the Underlying Ideas

In order to demonstrate the effectiveness of our two main ideas in our *NuPBO* solver, we conduct comparative experiments on 3 solvers. We develop two alternative versions of *NuPBO*, by replacing its scoring function and weighting scheme with the ones proposed in *LS-PBO*, dubbed *NuPBO-alt(s)* and *NuPBO-alt(w)*, respectively. The weighting scheme proposed in *LS-PBO* introduces a parameter  $\zeta$  for *NuPBO-alt(w)*, which is set to 100 as recommended by *LS-PBO*'s authors [26].

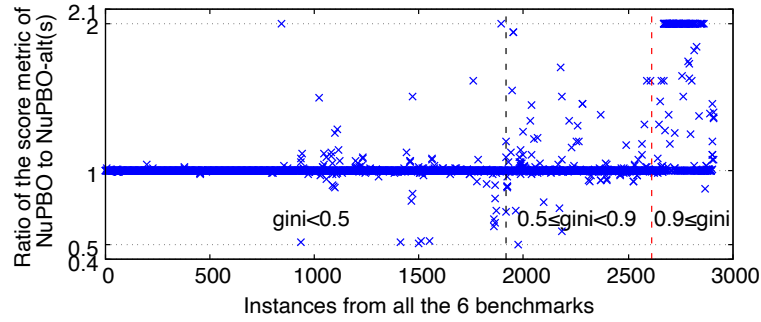
The comparative results of the cutoff time of 300 seconds and 1 hour are shown in Table 4. From Table 4, *NuPBO* outperforms its alternative versions on the majority of instances. We first discuss the effectiveness of the new scoring function.

**Regarding the New Scoring Function.** With the cutoff time of 300s, in terms of the metrics of #win. and avg<sub>score</sub>, *NuPBO* exhibits the best performance on 5 benchmarks. In terms of the metric of #feas., *NuPBO* performs better than *NuPBO-alt(s)* on the PB16 benchmark and CRAFT benchmark. On the remaining 4 benchmarks, the number of feasible solutions obtained by *NuPBO* is equal to that obtained by *NuPBO-alt(s)*.

With the cutoff time of 1h, *NuPBO* exhibits significantly better performance than *NuPBO-alt(s)* on 3 benchmarks including PB16, MIPLIB, and MWCB. On the remaining 3 benchmarks, namely CRAFT, SAP and WSNO, the performance of *NuPBO* is comparable to that of *NuPBO-alt(s)*.

To examine the intuition in Section 3.1, we conduct an experiment to analyze the relationship between the instance feature and the performance difference between *NuPBO* and *NuPBO-alt(s)*. Due to the difficulty of counting the coefficients of all variables in different constraints within an instance, we use the Gini coefficient [18] of the degree of hard constraints as the instance feature, denoted by  $Gini_d$ . For a PBO instance  $I_1$ , if the degree values of all hard constraints in  $I_1$  are arranged in ascending order,  $Gini_d$  can be calculated as follows:  $Gini_d = \frac{2}{n^2 \bar{d}} \sum_{i=1}^n i(d_i - \bar{d})$ , where  $n$  is the number of hard constraints,  $i$  is the





■ **Figure 1** The ratio of the *score* metric of *NuPBO* and *NuPBO-alt(s)* on instances from all the 6 benchmarks (cutoff 300s).

rank of degree values in ascending order,  $d_i$  is the degree of  $i$ -th hard constraint ( $d_i$  values are in ascending order), and  $\bar{d}$  is the mean value.<sup>11</sup> The greater  $Gini_d(I_1)$ , the greater the inequality between the degrees of constraints in instance  $I_1$ . In those instances whose  $Gini_d$  is large, the coefficient of a variable may differ greatly between constraints. For an instance  $I_1$ , we use  $score_{NuPBO I_1}$  to represent the competition score of *NuPBO*, and  $score_{NuPBO-alt(s) I_1}$  to represent the competition score of *NuPBO-alt(s)*.  $R(I_1) = \frac{score_{NuPBO I_1} + 1}{score_{NuPBO-alt(s) I_1} + 1}$  is used to denote the performance difference between *NuPBO* and *NuPBO-alt(s)*. Thus, if *NuPBO* finds a solution while *NuPBO-alt(s)* does not,  $R = 2$  (on the contrary,  $R = 0.5$ ). If  $R = 1$ , *NuPBO* and *NuPBO-alt(s)* obtained the same competition score (or no solution has been found).

We conduct an experiment on all 6 benchmarks with a cutoff time of 300s. The related results are presented in Figure 1. According to Figure 1, the x-axis represents 2909 instances of the 6 benchmarks, sorted by  $Gini_d$  in ascending order, and the y-axis represents the corresponding  $R$  values.

Results in Figure 1 demonstrate that *NuPBO* outperforms *NuPBO-alt(s)*, as the number of instances with  $R > 1$  exceeds those with  $R < 1$ . In addition, on instances with  $Gini_d \geq 0.9$ , *NuPBO* exhibits a significant performance advantage over *NuPBO-alt(s)*, and many instances in this category have an  $R$  value of 2, which indicates that *NuPBO* performs much better in terms of the metric of  $\#feas.$ . On instances with  $Gini_d \geq 0.5$ , *NuPBO* also shows performance improvement over *NuPBO-alt(s)*.

**Regarding the New Weighting Scheme.** With the cutoff time of 300s, in terms of the metrics of  $\#win.$  and  $avg_{score}$ , *NuPBO* outperforms *NuPBO-alt(w)* on all the benchmarks. In terms of the metric of  $\#feas.$ , *NuPBO* achieves better performance than *NuPBO-alt(w)* on 3 benchmarks. On the other 3 benchmarks, the value of  $\#feas.$  achieved by *NuPBO* is equal to that obtained by *NuPBO-alt(w)*.

With the cutoff time of 1h, regarding the metrics of  $\#win.$  and  $avg_{score}$ , *NuPBO* outperforms *NuPBO-alt(w)* on 5 out of 6 benchmarks, and achieves the same performance on the WSNO benchmark. Regarding the metric of  $\#feas.$ , *NuPBO* demonstrates better performance than *NuPBO-alt(w)* on 2 benchmarks. On the MIPLIB benchmark, the performance of *NuPBO-alt(w)* is only slightly better than that of *NuPBO*. On the 2 real-world application benchmarks, these SLS solvers achieve the same performance. The experimental results clearly indicate the effectiveness of our proposed new weighting scheme.

<sup>11</sup> [https://www.statsdirect.com/help/default.htm#nonparametric\\_methods/gini.htm](https://www.statsdirect.com/help/default.htm#nonparametric_methods/gini.htm)

■ **Table 5** Experimental results of *NuPBO*, *LS-PBO*, *NuPBO-alt(s)*, and *NuPBO-alt(w)* with seeds ranging from 1 to 10 on all the benchmarks (left: cutoff 300s, right: cutoff 1h).

Benchmark	#inst.	cutoff 300s			cutoff 1h		
		$avg_{avgsol}$	$avg_{stdev}$	$\frac{avg_{stdev}}{avg_{avgsol}}$	$avg_{avgsol}$	$avg_{stdev}$	$\frac{avg_{stdev}}{avg_{avgsol}}$
<i>NuPBO</i>							
PB16	1600	34567.32	259.79	0.75%	34721.64	123.30	0.36%
CRAFT	955	3035410.02	0.10	<0.01%	3000465.70	0.09	<0.01%
MIPLIB	291	59271045.04	240944.55	0.41%	58116554.74	299542.17	0.52%
MWCB	24	197890.62	1504.80	0.76%	193513.34	888.91	0.46%
SAP	21	1039.04	3.74	0.36%	1033.64	3.00	0.29%
WSNO	18	1301.21	225.55	17.33%	1158.61	0.00	0.00%
<i>LS-PBO</i>							
PB16	1600	30844.62	143.86	0.47%	34305.81	213.45	0.62%
CRAFT	955	3074484.99	1.69	<0.01%	3071190.03	1.11	<0.01%
MIPLIB	291	53327323.12	1262751.16	2.37%	50874231.31	924287.05	1.82%
MWCB	24	209821.48	1582.87	0.75%	201482.90	1525.52	0.76%
SAP	21	1066.74	4.61	0.43%	1059.17	3.43	0.32%
WSNO	18	1448.88	299.06	20.64%	1174.76	44.64	3.80%
<i>NuPBO-alt(s)</i>							
PB16	1600	36842.12	112.50	0.31%	35779.84	129.39	0.36%
CRAFT	955	3038627.52	0.10	<0.01%	3000465.70	0.10	<0.01%
MIPLIB	291	65612369.74	911077.41	1.39%	63282671.88	90491.84	0.14%
MWCB	24	210377.43	1688.62	0.80%	205368.10	1267.23	0.62%
SAP	21	1039.50	3.49	0.34%	1034.15	2.68	0.26%
WSNO	18	1295.50	192.98	14.90%	1158.65	0.12	0.01%
<i>NuPBO-alt(w)</i>							
PB16	1600	34535.79	312.20	0.90%	34594.61	214.89	0.62%
CRAFT	955	3094400.67	1.45	<0.01%	3094398.61	1.15	<0.01%
MIPLIB	291	59063484.68	259177.44	0.44%	57971941.39	350572.11	0.60%
MWCB	24	205188.29	1467.44	0.72%	201127.28	929.57	0.46%
SAP	21	1061.47	4.08	0.38%	1054.01	3.41	0.32%
WSNO	18	1293.97	143.69	11.10%	1159.25	2.02	0.17%

## 5.5 Stability of Local Search Solvers

In order to examine the stability of all four SLS solvers adopted in our experiments, each of the four SLS solvers runs 10 times with seeds ranging from 1 to 10 on all instances from all 6 benchmarks.

For a given solver  $S$  and an instance  $I$ :  $sol_{SIJ}$  denotes the cost of the best solution found by solver  $S$  with seed  $J$  on instance  $I$ ,  $avgsol$  denotes the average cost of best solutions obtained by solver  $S$  over all 10 runs on instance  $I$ , while  $stdev$  denotes the standard deviation of the cost of the best solutions obtained by solver  $S$  over all 10 runs on instance  $I$ . On a benchmark  $B$  consisting of multiple instances:  $avg_{avgsol}$  represents the average value of  $avgsol$  obtained by solver  $S$  over all instances where solutions are obtained, while  $avg_{stdev}$  stands for the average value of  $stdev$  obtained by solver  $S$  over all instances where solutions are found. The calculation of  $avg_{avgsol}$  is based on instances where solutions are found,

different solvers may find solutions on different subsets of instances for a given benchmark and cutoff time. In addition, for a given benchmark, it is possible that a solver finds solutions on more instances within a cutoff time of 1h than adopting a cutoff time of 300s. Moreover, according to the above definition of  $avg_{avg_{sol}}$ , we note that the value of  $avg_{avg_{sol}}$  cannot be used to compare the performance of different solvers.

The experimental results presented in Table 5 demonstrate that, with the cutoff time of 300s, all four SLS solvers exhibit stable performance on 5 out of 6 benchmarks, while on the WSNO benchmark, the performance is less stable compared to the other benchmarks. In addition, the values of  $\frac{avg_{stdev}}{avg_{avg_{sol}}}$  for *NuPBO* are less than 1% on all 5 benchmarks, which clearly indicates that *NuPBO* can achieve stable performance. With the cutoff time of 1h, all four SLS solvers perform stably on the 6 benchmarks.

## 6 Conclusions and Future Work

This paper is devoted to improving the performance of SLS solvers for solving the PBO problem via a new scoring function and a new weighting scheme. First, we introduced our new scoring function. Furthermore, we proposed a new weighting scheme that effectively determines when to increase the weight of the objective function. Based on these two main ideas, we developed a new SLS solver named *NuPBO*. Extensive experimental results demonstrate that *NuPBO* significantly outperforms *LS-PBO* on all testing benchmarks. *NuPBO* outperforms all its competitors on 3 real-world application benchmarks and shows competitive performance compared to state-of-the-art competitors on solving PB16, MIPLIB, and CRAFT benchmarks. In addition, *NuPBO* enhances the complementarity between SLS solvers and complete solvers on all testing benchmarks.

For future work, we would like to develop more efficient heuristic strategies and explore the effect of instance features on the performances of different categories of PBO solvers.

---

## References

- 1 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.
- 2 Carlos Ansótegui and Joel Gabàs. WPM3: An (in)complete algorithm for weighted partial MaxSAT. *Artificial Intelligence*, 250:37–57, 2017.
- 3 VL Beresnev, EN Goncharov, and AA Mel’nikov. Local search with a generalized neighborhood in the optimization problem for pseudo-boolean functions. *Journal of Applied and Industrial Mathematics*, 6:22–30, 2012.
- 4 Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. Core-boosted linear search for incomplete MaxSAT. In *Proceedings of CPAIOR 2019*, pages 39–56, 2019.
- 5 Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020.
- 6 Shaowei Cai, Chuan Luo, Jinkun Lin, and Kaile Su. New local search methods for partial MaxSAT. *Artificial Intelligence*, 240:1–18, 2016.
- 7 Shaowei Cai, Chuan Luo, and Kaile Su. Scoring functions based on second level score for k-sat with long clauses. *J. Artif. Intell. Res.*, 51:413–441, 2014. doi:10.1613/jair.4480.
- 8 Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring local search for partial maxsat. In *Proceedings of AAAI 2014*, pages 2623–2629, 2014.
- 9 Byungki Cha and Kazuo Iwama. Performance test of local search algorithms using new types of random CNF formulas. In *Proceedings of IJCAI 1995*, pages 304–311, 1995.
- 10 Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, and Shuichi Miyazaki. Local search algorithms for partial MAXSAT. In *Proceedings of AAAI 1997*, pages 263–268, 1997.

- 11 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proceedings of CP 2011*, pages 225–239, 2011.
- 12 Jo Devriendt, Stephan Gocht, Emir Demirovic, Jakob Nordström, and Peter J. Stuckey. Cutting to the core of pseudo-boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proceedings of AAAI 2021*, pages 3750–3758, 2021.
- 13 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.
- 14 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In Jérôme Lang, editor, *Proceedings of IJCAI 2018*, pages 1291–1299, 2018.
- 15 Jan Elffers and Jakob Nordström. A cardinal improvement to pseudo-Boolean solving. In *Proceedings of AAAI 2020*, pages 1495–1503, 2020.
- 16 Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schläpfer, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020. URL: [http://www.optimization-online.org/DB\\_HTML/2020/03/7705.html](http://www.optimization-online.org/DB_HTML/2020/03/7705.html).
- 17 Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012.
- 18 Corrado Gini. Concentration and dependency ratios. *Rivista di politica economica*, pages 769–792, 1997.
- 19 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL: <https://www.gurobi.com>.
- 20 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.
- 21 Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. The MIT press, 2009.
- 22 Holger H. Hoos, Laetitia Jourdan, Marie-Éléonore Kessaci, Thomas Stützle, and Nadarajan Veerapen. Special issue on "stochastic local search: Recent developments and trends". *International Transactions in Operational Research*, 27(1):697–698, 2020.
- 23 Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1-2):95–100, 2012.
- 24 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010.
- 25 Zhendong Lei and Shaowei Cai. Solving (weighted) partial MaxSAT by dynamic local search for SAT. In *Proceedings of IJCAI 2018*, pages 1346–1352, 2018.
- 26 Zhendong Lei, Shaowei Cai, Chuan Luo, and Holger H. Hoos. Efficient local search for pseudo boolean optimization. In *Proceedings of SAT 2021*, pages 332–348, 2021.
- 27 Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243:26–44, 2017.
- 28 Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su. CCLS: An efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7):1830–1843, 2015.
- 29 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular maxsat solver,. In *Proceedings of SAT 2014*, pages 438–445, 2014.
- 30 Rafiq Muhammad and Peter J. Stuckey. A stochastic non-CNF SAT solver. In *Proceedings of PRICAI 2006*, pages 120–129, 2006.
- 31 Alexander Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *Proceedings of FMCAD 2019*, pages 193–202, 2019.

- 32 Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Proceedings of AAAI 2014*, pages 2717–2723, 2014.
- 33 Olivier Roussel and Vasco Manquinho. Pseudo-boolean and cardinality constraints. In *Handbook of satisfiability*, pages 1087–1129. IOS Press, 2021.
- 34 Masahiko Sakai and Hidetomo Nabeshima. Construction of an ROBDD for a PB-constraint in band form and related techniques for pb-solvers. *IEICE Transactions on Information & Systems*, 98-D(6):1121–1127, 2015.
- 35 Bart Selman and Henry Kautz. Domain-independent extensions to GSAT: solving large structured satisfiability problems. In *Proceedings of IJCAI 1993*, pages 290–295, 1993.
- 36 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Pseudo-boolean optimization by implicit hitting sets. In *Proceedings of CP 2021*, pages 51:1–51:20, 2021.
- 37 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Improvements to the implicit hitting set approach to pseudo-boolean optimization. In *Proceedings of SAT 2022*, pages 13:1–13:18, 2022.
- 38 John Thornton and Abdul Sattar. Dynamic constraint weighting for over-constrained problems. In *Proceedings of PRICAI 1998*, pages 377–388, 1998.
- 39 Renato Tinós, Michal W Przewozniczek, and Darrell Whitley. Iterated local search with perturbation based on variables interaction for pseudo-boolean optimization. In *Proceedings of GECCO 2022*, pages 296–304, 2022.
- 40 Chris Voudouris and Edward Tsang. Partial constraint satisfaction problems and guided local search. *Proc., Practical Application of Constraint Technology (PACT'96), London*, pages 337–356, 1996.
- 41 Christos Voudouris, Edward PK Tsang, and Abdullah Alsheddy. Guided local search. In *Handbook of metaheuristics*, pages 321–361. Springer, 2010.
- 42 Joachim P. Walser. Solving linear pseudo-boolean constraint problems with local search. In *Proceedings of AAAI 1997*, pages 269–274, 1997.
- 43 Robert Wille, Hongyan Zhang, and Rolf Drechsler. ATPG for reversible circuits using simulation, boolean satisfiability, and pseudo boolean optimization. In *Proceedings of ISVLSI 2011*, pages 120–125, 2011.
- 44 Yuhang Zhang, Richard Hartley, John Mashford, and Stewart Burn. Superpixels via pseudo-boolean optimization. In *Proceedings of ICCV 2011*, pages 1387–1394, 2011.
- 45 Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, Chu-Min Li, and Felip Manyà. BandMaxSAT: A local search maxsat solver with multi-armed bandit. In *Proceedings of IJCAI 2022*, pages 1901–1907, 2022.

# Boosting Decision Diagram-Based Branch-And-Bound by Pre-Solving with Aggregate Dynamic Programming

Vianney Coppé  

UCLouvain, Louvain-la-Neuve, Belgium

Xavier Gillard  

UCLouvain, Louvain-la-Neuve, Belgium

Pierre Schaus  

UCLouvain, Louvain-la-Neuve, Belgium

---

## Abstract

Discrete optimization problems expressible as dynamic programs can be solved by branch-and-bound with decision diagrams. This approach dynamically compiles bounded-width decision diagrams to derive both lower and upper bounds on unexplored parts of the search space, until they are all enumerated or discarded. Assuming a minimization problem, relaxed decision diagrams provide lower bounds through state merging while restricted decision diagrams obtain upper bounds by excluding states to limit their size. As the selection of states to merge or delete is done locally, it is very myopic to the global problem structure. In this paper, we propose a novel way to proceed that is based on pre-solving a so-called aggregate version of the problem with a limited number of states. The compiled decision diagram of this aggregate problem is tractable and can fit in memory. It can then be exploited by the original branch-and-bound to generate additional pruning and guide the compilation of restricted decision diagrams toward good solutions. The results of the numerical study we conducted on three combinatorial optimization problems show a clear improvement in the performance of DD-based solvers when blended with the proposed techniques. These results also suggest an approach where the aggregate dynamic programming model could be used in replacement of the relaxed decision diagrams altogether.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial optimization

**Keywords and phrases** Discrete Optimization, Decision Diagrams, Aggregate Dynamic Programming

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.13

## 1 Introduction

On top of their use for Boolean encodings [27], formal verification [25], model checking [15], computer-aided design [29] and much more, *decision diagrams* (DDs) have recently emerged as a tool for discrete optimization. They provide a compact way to encode a set of solutions to a problem. Still, for large problems, DDs representing the whole solution space – called *exact* DDs – can quickly become intractable to compute. Two variants of DDs can be used instead: *restricted* [10] and *relaxed* [1, 8] DDs that respectively encode a subset and superset of the set of solutions. When compiled based on a *dynamic programming* (DP) model, these approximate DDs allow to compute bounds on the objective function for any subproblem while controlling the size of the DD compiled. Restricted DDs aim to find good admissible solutions by iteratively extending a bounded set of promising candidates while dropping others, in a beam search fashion. On the other hand, relaxed DDs rely on a problem-dependent state merging scheme to maintain an acceptable DD size while preserving all solutions of the problem. In [9], Bergman et al. presented a *branch-and-bound* algorithm solely based on these two ingredients, thus introducing a new general-purpose discrete optimization framework and solver.



© Vianney Coppé, Xavier Gillard, and Pierre Schaus;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 13; pp. 13:1–13:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In addition to exploiting the compactness of DP models, the main novelty of this approach is its unique way of deriving lower and upper bounds. In the last few years, some algorithmic improvements have been suggested to further strengthen these bounds. Assuming a minimization problem, Gillard et al. [19] showed how user-defined lower bound formulas can be integrated to prune DDs during their compilation and thus concentrate the search on promising parts of the search space. They also proposed a way to compute tighter lower bounds for all nodes contained in a relaxed DD through *local bounds*. Rudich et al. [30] introduced a *peeling* operator that splits a relaxed DD in two: one part containing all paths traversing a selected exact node and the other containing all remaining paths. It allows both to warm-start the compilation of subsequent relaxed DDs and to strengthen the bounds of the nodes inside the relaxed DD on which the peeling has been performed. More recently, [16] generalized the ideas of [19] and introduced the use of a cache storing new thresholds that further enhance the pruning power of the solver. Other factors impacting the quality of the bounds provided by relaxed DDs have been studied, including variable orderings [7, 11, 26] and alternative compilation schemes [24]. Yet, all these approaches rely on a problem-specific state merging operator at the heart of the relaxation, which does not yield tight relaxations for all problems, as our computational experiments show.

After covering the necessary background about DD-based optimization, this paper presents an alternate relaxation scheme for deriving good bounds by incorporating ideas from *aggregate dynamic programming* [2, 3] to the DD-based discrete optimization framework. The underlying idea of the approach is to deduce information about an original problem instance by creating and solving an aggregate – relaxed – version of it. This is achieved by *aggregating* the states of the DP model as to obtain a much smaller DP state space. If this aggregation is adequately specified, one can compute a lower bound for any original subproblem by finding the optimal solution of its aggregate version. Furthermore, this optimal aggregate solution can be *disaggregated* and transposed in the original problem to find good heuristic solutions. In practice, the aggregation-based lower bounds are used as additional pruning within the compilation of relaxed and restricted DDs. Moreover, aggregate solutions are translated into node selection heuristics to steer the compilation of restricted DDs toward resembling solutions to the original problem, which are thus expected to be good.

Throughout the paper, the framework is illustrated on three different combinatorial problems: the Talent Scheduling Problem, the Pigment Sequencing Problem and the Aircraft Landing Problem. They are then used for the experimental evaluation of the framework, the results of which show that the aggregation-based bound brings additional pruning and enables solving more instances. Furthermore, the aggregation-based node selection heuristic improves the quality of the solutions found early in the search and thus contributes to speeding up the overall resolution. Finally, we show that a DD-based solver using only the aggregation-based bound as relaxation performs almost equally well, which is a promising direction for problems for which defining a merging operator is difficult or inefficient.

Although this paper is – to the best of our knowledge – the first to combine aggregate dynamic programming with the DD-based branch-and-bound paradigm proposed by Bergman et al, there has already been some hybridization work to combine discrete optimization with DDs and other methods. For instance, in [12], Cappart et al. propose to use reinforcement learning to guess the variable ordering that should be used to derive the best possible bounds from the compiled approximate DDs. Other attempts combined DDs with Lagrangian relaxation [13, 23] or MIP [5, 22, 31, 32]. On a slightly different note, a method has been proposed where restricted DDs are used to generate good neighborhoods in a *large neighborhood search* framework [20].

## 2 Preliminaries

### 2.1 Discrete Optimization

A discrete optimization problem  $\mathcal{P}$  involves finding the best possible solution  $x^*$  from a finite set of feasible solutions  $Sol(\mathcal{P}) = D \cap C$ . This set is determined by the domain  $D = D_0 \times \dots \times D_{n-1}$  from which the variables  $x = \langle x_0, \dots, x_{n-1} \rangle$  each take on a value, i.e.  $x_j \in D_j$ , and by a set of constraints  $C$  imposed on the value assignments. The quality of the solutions is evaluated according to an objective function  $f(x)$  that must be minimized. Formally, the problem is defined as  $\min \{f(x) \mid x \in D \cap C\}$  and any optimal solution  $x^*$  must satisfy  $x^* \in Sol(\mathcal{P})$  and  $\forall x \in Sol(\mathcal{P}) : f(x^*) \leq f(x)$ . We describe below three optimization problems that will be utilized in the paper as illustrations for the aggregation-based framework.

**Talent Scheduling Problem.** The *Talent Scheduling Problem* (TalentSched) is a film shoot scheduling problem that considers a set  $N = \{0, \dots, n-1\}$  of scenes and a set  $A = \{0, \dots, m-1\}$  of actors. Each scene  $i \in N$  involves a required set  $R_i \subseteq A$  of actors for a duration  $D_i \in \mathbb{N}$ . Moreover, each actor  $k \in M$  has a pay rate  $C_k$  and is paid without interruption from their first to their last scheduled scene. The objective of TalentSched is to find a permutation of the scenes that minimizes the total cost of the film shoot.

**Pigment Sequencing Problem.** The *Pigment Sequencing Problem* (PSP) is a single-machine production planning problem that aims to minimize the stocking and changeover costs while satisfying a set of orders. There are different item types  $I = \{0, \dots, n-1\}$  with a given stocking cost  $S_i$  to pay for each time period between the production and the deadline of an order. For each pair  $i, j \in I$  of item types, a changeover cost  $C_{ij}$  is incurred whenever the machine switches the production from item type  $i$  to  $j$ . Finally, the demand matrix  $Q$  contains all the orders:  $Q_p^i \in \{0, 1\}$  indicates whether there is an order for item type  $i \in I$  at time period  $p$  with  $0 \leq p < H$  and  $H$  the time horizon.

**Aircraft Landing Problem.** The *Aircraft Landing Problem* (ALP) requires to schedule the landing of a set of aircrafts  $N = \{0, \dots, n-1\}$  on a set of runways  $R = \{0, \dots, r-1\}$ . The aircrafts have a target  $T_i$  and latest  $L_i$  landing time. Moreover, the set of aircrafts is partitioned in disjoint sets  $A_0, \dots, A_{c-1}$  corresponding to different aircraft classes in  $C = \{0, \dots, c-1\}$ . For each pair of aircraft classes  $a, b \in C$ , a minimum separation time  $S_{a,b}$  between the landings is given. The goal is to find the schedule that minimizes the total waiting time of the aircrafts – the delay between their target time and scheduled landing time – while respecting their latest landing time.

### 2.2 Dynamic Programming

*Dynamic programming* (DP) is a *divide-and-conquer* strategy introduced by Bellman [4] for solving discrete optimization problems with an inherent recursive structure. It works by recursively decomposing the problem in smaller and overlapping subproblems. The cornerstone of the approach is the caching of intermediate results that allows each distinct subproblem to be solved only once. A *DP model* of a discrete optimization problem  $\mathcal{P}$  can be defined as a labeled transition system consisting of:

- the *control variables*  $x_j \in D_j$  with  $j \in \{0, \dots, n-1\}$ .
- a set of *state-spaces*  $S = \{S_0, \dots, S_n\}$  among which one distinguishes the *initial state*  $r$ , the *terminal state*  $t$  and the *infeasible state*  $\hat{o}$ .



## 13:4 Boosting DD-Based Branch-And-Bound with Aggregate Dynamic Programming

- a set  $t$  of *transition functions* s.t.  $t_j : S_j \times D_j \rightarrow S_{j+1}$  for  $j = 0, \dots, n-1$  taking the system from one state  $s^j$  to the next state  $s^{j+1}$  based on the value  $d$  assigned to variable  $x_j$ , or to  $\perp$  if assigning  $x_j = d$  is infeasible. These functions should never allow one to recover from infeasibility, i.e.  $t_j(\hat{0}, d) = \hat{0}$  for any  $d \in D_j$ .
- a set  $h$  of *transition value functions* s.t.  $h_j : S_j \times D_j \rightarrow \mathbb{R}$  representing the immediate reward of assigning some value  $d \in D_j$  to the variable  $x_j$  for  $j = 0, \dots, n-1$ .
- a *root value*  $v_r$ .

On that basis, the objective function  $f(x)$  of  $\mathcal{P}$  is formulated as follows:

$$\begin{aligned} & \text{minimize } f(x) = v_r + \sum_{j=0}^{n-1} h_j(s^j, x_j) \\ & \text{subject to } s^{j+1} = t_j(s^j, x_j), \text{ for all } j = 0, \dots, n-1, \text{ with } x_j \in D_j \\ & \quad s^j \in S_j, j = 0, \dots, n \text{ and } x \in C. \end{aligned} \tag{1}$$

**TalentSched.** A DP model for TalentSched was introduced in [17] that we slightly adapt here to make it suitable for the relaxation discussed in Section 2.3.1. States of this model are pairs  $(M, P)$  where  $M$  and  $P$  are disjoint sets of scenes that respectively must or might still be scheduled. The only case where  $P$  is non-empty happens when a state is relaxed.

- Control variables:  $x_j \in N$  with  $0 \leq j < n$  decides which scene is shot in  $j$ -th position.
- State spaces:  $S = \{(M, P) \mid M, P \subseteq N, M \cap P = \emptyset\}$ . The root state is  $r = (N, \emptyset)$  and the terminal states are of the form  $(\emptyset, P)$ .
- Transition functions:

$$t_j(s^j, x_j) = \begin{cases} (s^j.M \setminus \{x_j\}, s^j.P \setminus \{x_j\}) & \text{if } x_j \in s^j.M, \\ (s^j.M \setminus \{x_j\}, s^j.P \setminus \{x_j\}), & \text{if } x_j \in s^j.P \text{ and } |s^j.M| < n - j, \\ \hat{0}, & \text{otherwise.} \end{cases}$$

A scene from  $P$  can only be selected if there are more spots left than scenes in  $M$ .

- Transition value functions: let  $a(Q) = \cup_{i \in Q} R_i$  be the required set of actors for a set of scenes  $Q$ . Given a state  $s = (M, P)$ , the set of actors that are guaranteed to be on-location is computed as  $o(s) = a(s.M) \cap a(N \setminus (s.M \cup s.P))$  because they are required both for a scene that must still be scheduled and for another that is guaranteed to be scheduled. In the transition value functions, we add all the actors from  $R_{x_j}$  to this set and sum the individual costs:  $h_j(s^j, x_j) = D_{x_j} \sum_{k \in o(s^j) \cup R_{x_j}} C_k$ .
- Root value:  $v_r = 0$ .

**PSP.** The PSP was already tackled with a DD-based approach in [16, 20]. We hereby recall the DP model from [16] that allows the machine to be idle at some time periods. In this model, the decisions are made backwards – this allows to define transition functions that only lead to feasible production schedules. If variable  $x_j$  decides the type of item to produce at period  $j$ , the reverse variable ordering  $x_{H-1}, \dots, x_0$  is thus used. To simplify the transition functions, let us denote by  $P_r^i$  the time period at which the  $r$ -th item of type  $i$  must be delivered, i.e.  $P_r^i = \min\{0 \leq q < H \mid \sum_{p=0}^q Q_p^i \geq r\}$  for all  $i \in N, 0 \leq r \leq \sum_{0 \leq p < H} Q_p^i$ . Moreover, we define a dummy item type  $\perp$  used for idle periods and  $N' = N \cup \{\perp\}$ .

States are pairs  $(i, R)$  with  $i$  the item type that the machine is currently set to produce and  $R$  a vector that gives the remaining number  $R_i$  of demands to satisfy for each type  $i$ .

- Control variables:  $x_j \in N'$  with  $0 \leq j < H$  decides the item type to produce at period  $j$ .

- State space:  $S = \{s \mid s.i \in N', \forall i \in N, 0 \leq s.R_i \leq \sum_{0 \leq p < H} Q_p^i\}$ . The root state is given by  $r = \langle \perp, (\sum_{0 \leq p < H} Q_p^0, \dots, \sum_{0 \leq p < H} Q_p^{n-1}) \rangle$  and the terminal states are of the form  $\langle i, (0, \dots, 0) \rangle$  with  $i \in N'$ .
- Transition functions:

$$t_j(s^j, x_j) = \begin{cases} \langle t_j^i(s^j, x_j), t_j^R(s^j, x_j) \rangle, & \text{if } x_j \neq \perp \text{ and } s^j.R_{x_j} > 0 \text{ and } j \leq P_{s^j.R_{x_j}}^{x_j}, \\ \langle t_j^i(s^j, x_j), t_j^R(s^j, x_j) \rangle, & \text{if } x_j = \perp \text{ and } \sum_{i \in N} s^j.R_i < j + 1, \\ \hat{0}, & \text{otherwise.} \end{cases}$$

where

$$\begin{aligned} t_j^i(s^j, x_j) &= \begin{cases} x_j, & \text{if } x_j \neq \perp \\ s^j.i, & \text{otherwise.} \end{cases} \\ t_j^R(s^j, x_j) &= \begin{cases} (s^j.R_0, \dots, s^j.R_{x_j} - 1, \dots, s^j.R_{n-1}), & \text{if } x_j \neq \perp \\ s^j.R, & \text{otherwise.} \end{cases} \end{aligned}$$

In the transition function, the first condition ensures that there remains at least one item to produce for the chosen type and that the current time period  $j$  is earlier than its deadline. The second condition ensures that idle periods cannot be scheduled when the remaining quantity to produce is equal to the number of periods left.

- Transition value functions: the changeover and stocking costs are computed as:

$$h_j(s^j, x_j) = \left\{ \begin{array}{l} C_{x_j s^j.i}, \quad \text{if } x_j \neq \perp \text{ and } s^j.i \neq \perp \\ 0, \quad \text{otherwise.} \end{array} \right\} + \left\{ \begin{array}{l} S_{x_j} \cdot (j - P_{s^j.R_{x_j}}^{x_j}), \quad \text{if } x_j \neq \perp \\ 0, \quad \text{otherwise.} \end{array} \right\}$$

- Root value:  $v_r = 0$ .

**ALP.** We reproduce here the DP model presented in [28] where states are pairs  $(Q, ROP)$ , with  $Q$  a vector that gives the remaining number of aircrafts of each class to schedule and  $ROP$  a *runway occupation profile*: a vector containing pairs  $(l, c)$  that respectively give the time and aircraft class of the latest landing scheduled on each runway. Similarly to the PSP modeling, we denote by  $\perp$  either a dummy aircraft class or a dummy runway.

- Control variables: we use pairs of variables  $(x_j, y_j) \in (C \times R) \cup \{(\perp, \perp)\}$  with  $0 \leq j < n$  that represent the decision to place an aircraft of class  $x_j$  on runway  $y_j$ , or to schedule nothing at all in case of  $(\perp, \perp)$ .
- State spaces:  $S = \{(Q, ROP) \mid \forall i \in C : Q_i \geq 0, \forall k \in R : ROP_k.l \geq 0, ROP_k.c \in C \cup \{\perp\}\}$ . The root state is  $r = (\langle |A_0|, \dots, |A_{c-1}| \rangle, \langle (0, \perp), \dots, (0, \perp) \rangle)$  and the terminal states are of the form  $\langle (0, \dots, 0), ROP \rangle$ .
- Transition functions: if  $A_i^k$  gives the aircraft from class  $i$  that must be scheduled when there are  $k$  aircrafts left from this class, we can define the function computing the earliest landing time given a state  $s$ , a class  $x$  and a runway  $y$ :

$$E(s, x, y) = \begin{cases} T_{A_x^s.Q_x}, & \text{if } s.ROP_y.l = 0 \text{ and } s.ROP_y.c = \perp, \\ \max(s.ROP_y.l + \min_{i \in C} S_{i,x}, T_{A_x^s.Q_x}), & \text{if } s.ROP_y.l > 0 \text{ and } s.ROP_y.c = \perp, \\ \max(s.ROP_y.l + S_{s.ROP_y.c,x}, T_{A_x^s.Q_x}), & \text{otherwise.} \end{cases}$$

This allows us to define the transition functions as:

$$t_j(s^j, x_j, y_j) = \begin{cases} \langle t_j^Q(s^j, x_j, y_j), t_j^{ROP}(s^j, x_j, y_j) \rangle, & \text{if } x_j \neq \perp \text{ and } s^j.Q_{x_j} > 0 \\ & \text{and } E(s^j, x_j, y_j) \leq L_{A_{x_j}^{s^j.Q_{x_j}}}, \\ \langle t_j^Q(s^j, x_j, y_j), t_j^{ROP}(s^j, x_j, y_j) \rangle, & \text{if } x_j = \perp \text{ and } \sum_{i \in C} s^j.Q_i = 0, \\ \hat{0}, & \text{otherwise.} \end{cases}$$

where

$$\begin{aligned} t_j^Q(s^j, x_j, y_j) &= \begin{cases} \langle s^j.Q_0, \dots, s^j.Q_{x_j-1}, \dots, s^j.Q_{c-1} \rangle, & \text{if } x_j \neq \perp \\ s^j.Q, & \text{otherwise.} \end{cases} \\ t_j^{ROP}(s^j, x_j, y_j) &= \begin{cases} \langle s^j.ROP_0, \dots, (E(s^j, x_j, y_j), x_j), \dots, s^j.ROP_{r-1} \rangle, & \text{if } x_j \neq \perp \\ s^j.ROP, & \text{otherwise.} \end{cases} \end{aligned}$$

The first condition of the transition function ensures that there remains at least one aircraft of the chosen class and that its earliest landing time is not greater its latest landing time. The second condition only allows us to schedule dummy aircrafts when there are no aircrafts left to schedule.

- Transition value functions: the waiting time of the aircraft is computed as:

$$h_j(s^j, x_j, y_j) = \begin{cases} E(s^j, x_j, y_j) - T_{A_{x_j}}^{s^j.Q_{x_j}}, & \text{if } x_j \neq \perp \\ 0, & \text{otherwise.} \end{cases}$$

- Root value:  $v_r = 0$ .

Because the runways are identical and independent, there are many symmetries in this model. This can be mitigated by sorting the ROP of every state by increasing latest landing time, breaking ties according to the previous aircraft class scheduled.

### 2.3 Decision Diagrams

When used to manipulate the DP model of a discrete optimization problem  $\mathcal{P}$ , DDs are graphical encodings that represent a set of solutions of the problem. More precisely, a DD  $\mathcal{B} = (U, A, \sigma, l, v)$  is a layered directed acyclic graph composed of a set of nodes  $U$  interconnected by a set of arcs  $A$ . Starting from a single node  $u_r$  corresponding to a DP state given by the function  $\sigma(u_r)$ , the process of iteratively extending a set of partial solutions is called the *compilation* of a DD and is described by Algorithm 1. Note that the highlighted portions concern the ingredients introduced in Section 3 and can be ignored for now. The algorithm begins by initializing a layer  $L_i$  that only contains the *root node*  $u_r$ , assuming its state  $\sigma(u_r)$  belongs to the  $i$ -th stage of the DP model. The subsequent layers of the DD are then constructed sequentially by applying each valid transition of the DP model to every node of the last completed layer at lines 8–16. Each layer thus corresponds to a stage of the DP model and contains a single node for each state reached in order to preserve the compactness of the model. The arcs  $a \in A$  materialize the transitions that exist between the states of consecutive stages. In particular, the arc  $a = (u \xrightarrow{d} u')$  connecting nodes  $u \in L_j, u' \in L_{j+1}$  represents the transition between  $\sigma(u)$  and  $\sigma(u')$ . The decision associated with this transition is stored by the *label*  $l(a) = d \in D_j$  and the transition value is given by the arc *value*  $v(a)$ .

The algorithm completes when the last layer  $L_n$  is generated, constituted by a single node  $t$  called the *terminal* node. The DD thus constructed contains a set of  $u_r \rightsquigarrow t$  paths that can be combined with any previously discovered  $r \rightsquigarrow u_r$  path, connecting the *root* of the problem to  $u_r$ . Any  $r \rightsquigarrow t$  path  $p = (a_0, \dots, a_{n-1})$  represents a solution given by  $x(p) = (l(a_0), \dots, l(a_{n-1}))$ . The objective value of such solution can also be retrieved from the sequence of arcs by accumulating their values, and adding the root value:  $v(p) = v_r + \sum_{j=0}^{n-1} v(a_j)$ . The set of solutions contained in the DD is denoted as  $Sol(\mathcal{B}) = \{x(p) \mid \exists p : r \rightsquigarrow t, p \in \mathcal{B}\}$ . A DD rooted at a node  $u_r$  is *exact* if it perfectly represents the set of solutions of the corresponding subproblem  $\mathcal{P}|_{u_r}$ , i.e.  $Sol(\mathcal{B}) = Sol(\mathcal{P}|_{u_r})$  and  $v(p) = f(x(p)), \forall p \in \mathcal{B}$ . The best value among the  $u_1 \rightsquigarrow u_2$  paths in  $\mathcal{B}$  is denoted  $v^*(u_1 \rightsquigarrow u_2 \mid \mathcal{B})$ , and in particular  $v^*(u \mid \mathcal{B}) = v^*(r \rightsquigarrow u \mid \mathcal{B})$ .

■ **Algorithm 1** Compilation of DD  $\mathcal{B}$  rooted at node  $u_r$  with maximum width  $W$ .

---

```

1:  $\hat{i} \leftarrow$  index of the layer containing  $u_r$ 
2:  $L_{\hat{i}} \leftarrow \{u_r\}$ 
3:  $\tilde{P} \leftarrow \Delta(\tilde{p})$  with  $\tilde{p}$  the optimal solution for  $\pi(\sigma(u_r))$  // retrieve disaggregate solution
4: for  $j = \hat{i}$  to  $n - 1$  do
5:   if  $|L_j| > W$  then
6:     restrict or relax the layer to get  $W$  nodes with Algorithm 2
7:    $L_{j+1} \leftarrow \emptyset$ 
8:   for all  $u \in L_j$  do
9:      $v_{rlb}(\sigma(u)) \leftarrow \max\{v_{rlb}(\sigma(u)), v_{agg}(\pi(\sigma(u)))\}$  // inject aggregation-based bound
10:    if  $v^*(u | \mathcal{B}) + v_{rlb}(\sigma(u)) \geq \bar{v}$  then // rough lower bound pruning w.r.t. incumbent
11:      continue
12:    for all  $d \in D_j$  do
13:      create node  $u'$  with state  $\sigma(u') = t_j(\sigma(u), d)$  or retrieve it from  $L_{j+1}$ 
14:      create arc  $a = (u \xrightarrow{d} u')$  with  $v(a) = h_j(\sigma(u), d)$  and  $l(a) = d$ 
15:       $score(a) \leftarrow 1$  if  $l(a) \in \tilde{P}_j$ , 0 otherwise
16:      add  $u'$  to  $L_{j+1}$  and add  $a$  to  $A$ 

```

---

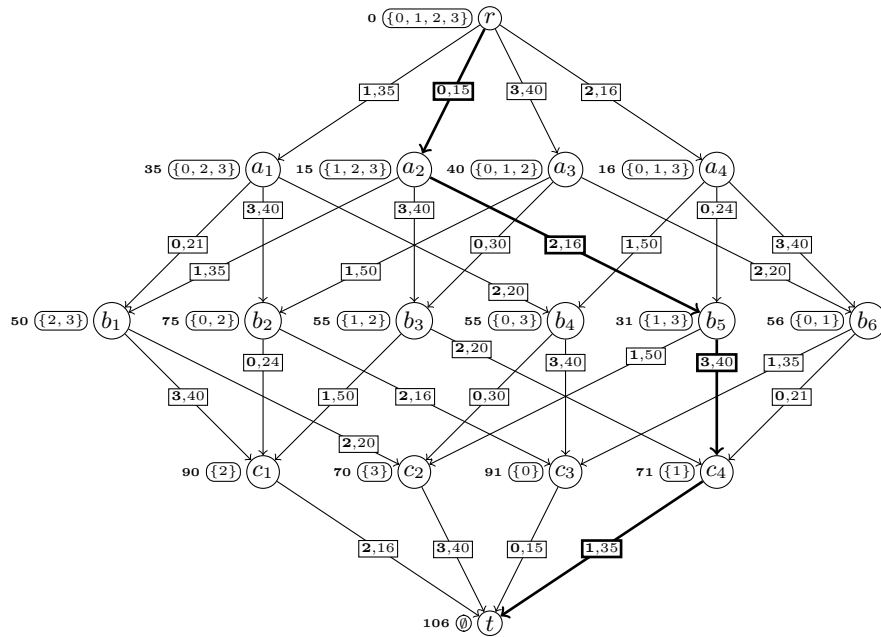
► **Example 1.** Let us define a TalentSched instance with 4 scenes with durations  $D = \langle 3, 5, 2, 4 \rangle$  and 4 actors with pay rates  $C = \langle 10, 20, 30, 40 \rangle$ . The actor requirements for each scene are given by  $R = \langle \{0, 3\}, \{0, 1, 3\}, \{0, 2, 3\}, \{0, 1, 2, 3\} \rangle$ . Figure 1 shows the exact DD compiled for this instance with the DP model recalled in Section 2.2. Note that for each state  $s = (M, P)$  corresponding to a node in the DD, we only show the set  $M$  since  $P$  is always empty in exact nodes. An optimal solution of the problem is  $\langle 0, 2, 3, 1 \rangle$ , which gives an objective value of 106.

As the reader might have guessed, the compilation of an exact DD for a combinatorial optimization problem suffers from the curse of dimensionality as much as the corresponding DP model. This is why DD-based discrete optimization rarely relies on exact DDs but rather on *restricted* and *relaxed* DDs. These two variants follow two distinct compilation schemes that allow to maintain the number of nodes of each layer – called the *width* – under a given parameter  $W$ . In Algorithm 1, this logic is performed at line 5 where the width of the current layer is compared with  $W$ . If needed, the layer is then either restricted or relaxed at line 6 by calling Algorithm 2.

### 2.3.1 Approximate Decision Diagrams

As stated by Algorithm 2, restricted DDs simply remove surplus nodes from the layer until it is reduced to  $W$  nodes. A heuristic is used to evaluate the nodes and drop the least promising ones. Restricted DDs thus generate a subset of the solutions of the corresponding problem, i.e.  $Sol(\bar{\mathcal{B}}) \subseteq Sol(\mathcal{P})$  and  $v(p) = f(x(p)), \forall p \in \bar{\mathcal{B}}$  for a restricted DD  $\bar{\mathcal{B}}$ . They thus provide upper bounds on the objective value.

As opposed to restricted DDs, a relaxed DD  $\underline{\mathcal{B}}$  yields lower bounds by representing a superset of the solutions of the corresponding problem:  $Sol(\underline{\mathcal{B}}) \supseteq Sol(\mathcal{P})$  and  $v(p) \leq f(x(p)), \forall p \in \underline{\mathcal{B}}$ . This is achieved through a problem-specific *state merging* operator  $\oplus(\sigma(\mathcal{M}))$  that defines an approximate representation that includes all states  $\sigma(\mathcal{M}) = \{\sigma(u) \mid u \in \mathcal{M}\}$  corresponding to the merged nodes  $\mathcal{M}$  and preserves all their outgoing transitions, although



■ **Figure 1** The exact DD for the TalentSched instance given in Example 1. Nodes are annotated with their state and the best prefix value. Arcs are labeled with the associated decision in bold and transition value. The arcs constituting one of the optimal solutions are highlighted in bold.

■ **Algorithm 2** Restriction or relaxation of layer  $L_j$  with maximum width  $W$ .

```

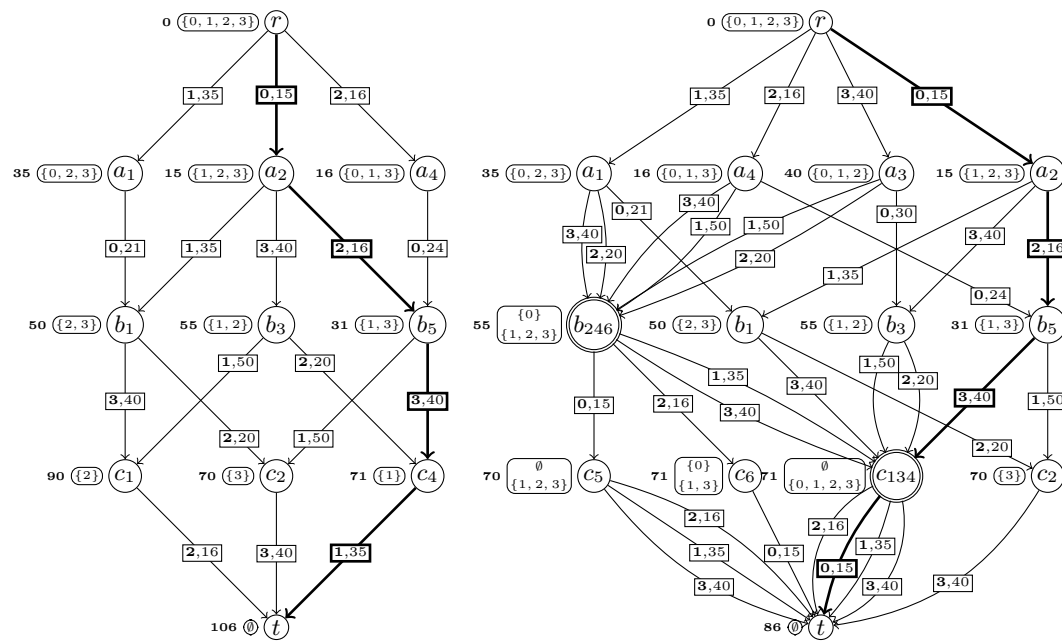
1: while  $|L_j| > W$  do
2:    $\mathcal{M} \leftarrow$  select nodes from  $L_j$  according to their score
3:    $L_j \leftarrow L_j \setminus \mathcal{M}$ 
4:   create node  $\mu$  with state  $\sigma(\mu) = \oplus(\sigma(\mathcal{M}))$  and add it to  $L_j$  // for relaxation only
5:   for all  $u \in \mathcal{M}$  and arc  $a = (u' \xrightarrow{d} u)$  incident to  $u$  do
6:     replace  $a$  by  $a' = (u' \xrightarrow{d} \mu)$  and set  $v(a') = \Gamma_{\mathcal{M}}(v(a), u)$ 

```

it may also introduce infeasible transitions. In Algorithm 2, a *meta*-node is created for this merged state at line 4 and the arcs pointing to the deleted nodes are redirected to this merged node at line 6. The operator  $\Gamma_{\mathcal{M}}$  permits to adjust the value of these arcs if needed. In all three formulations given below, this operator is the identity function.

**TalentSched.** The merging operator is defined by  $\oplus(\mathcal{M}) = (\oplus_M(\mathcal{M}), \oplus_P(\mathcal{M}))$  where  $\oplus_M(\mathcal{M}) = \bigcap_{s \in \mathcal{M}} s.M$  and  $\oplus_P(\mathcal{M}) = (\bigcup_{s \in \mathcal{M}} s.M \cup s.P) \setminus (\bigcap_{s \in \mathcal{M}} s.M)$ . The definition of  $\oplus_P(\mathcal{M})$  ensures that the resulting set of scenes that might be scheduled contains any scene that must or might be scheduled in any of the states, except those that still must be scheduled for all states.

**PSP.** A valid merging operator is  $\oplus(\mathcal{M}) = (\perp, \langle \min_{s \in \mathcal{M}} s.R_0, \dots, \min_{s \in \mathcal{M}} s.R_{n-1} \rangle)$ . The configuration of the machine is always reset to the dummy item type  $\perp$  as there is little chance that merged states agree on it. For each item type, the remaining number of demands is computed by taking the minimum value among all merged states, meaning that any demand satisfied by at least one state is considered satisfied in the merged state.



■ **Figure 2** Respectively on the left and the right, a restricted and relaxed DD for the TalentSched instance given in Example 1, compiled with  $W$  set to 3 and 4. Merged nodes are circled twice.

**ALP.** The merging operator is again defined separately for each component of the states:  $\oplus(\mathcal{M}) = (\oplus_Q(\mathcal{M}), \oplus_{ROP}(\mathcal{M}))$ . First, the minimum remaining quantity of aircrafts for each class is stored in the merged state:  $\oplus_Q(\mathcal{M}) = \langle \min_{s \in \mathcal{M}} s.Q_0, \dots, \min_{s \in \mathcal{M}} s.Q_{c-1} \rangle$ . For the ROP, the minimum latest landing time on each runway is kept and the last aircraft classes scheduled are reset to  $\perp$ :  $\oplus_{ROP}(\mathcal{M}) = \langle (\min_{s \in \mathcal{M}} s.ROP_0.l, \perp), \dots, (\min_{s \in \mathcal{M}} s.ROP_{r-1}.l, \perp) \rangle$ .

► **Example 2.** Figure 2 shows approximate DDs for the TalentSched instance introduced in Example 1. Despite having a maximum width of 3, the best solution contained in the restricted DD is the optimal solution previously found. With a maximum width of 4, the relaxed DD provides a global lower bound of 86. The path corresponding to this lower bound is given by the assignment  $\langle 0, 2, 3, 0 \rangle$ , which is infeasible because scene 0 is scheduled twice.

### 2.3.2 Branch-and-Bound

In [9], a branch-and-bound algorithm based only on restricted and relaxed DDs was introduced. It maintains a queue of open nodes that represent the set of subproblems that remain to process. For each of them, a restricted DD is compiled in an attempt to improve the incumbent solution. Then, a relaxed DD is constructed in order to both decompose the given subproblem into even smaller ones and to compute a lower bound for each of them. These nodes are then added to the branch-and-bound queue for further exploration, unless the lower bound permits their direct elimination. Ultimately, the optimality of the best solution discovered during the search is confirmed once the queue has been emptied.

### 2.3.3 Rough Lower Bound

The *rough lower bound* (RLB) [19] is an additional optional modeling component that can be specified to speed up the resolution of any optimization problem. For any node  $u$ , the RLB gives a lower bound on the best value one can obtain when solving the corresponding subproblem  $\sigma(u)$ , i.e.  $\underline{v}_{rlb}(\sigma(u)) \leq v^*(u \rightsquigarrow t \mid \mathcal{B})$  with  $\mathcal{B}$  the exact DD for the problem. It is used at line 10 of Algorithm 1 to filter nodes *a priori* by comparing this lower bound with the incumbent value  $\bar{v}$ . Since the RLB is computed for each node of the approximate DDs compiled throughout the branch-and-bound, it needs to be computationally cheap.

The RLB has the potential both to focus the compilation of restricted DDs on promising parts of the search space and to strengthen the bounds obtained through relaxed DDs. Furthermore, the branch-and-bound algorithm uses the RLB to make pruning decisions, if it happens to be tighter than the bound obtained with relaxed DDs.

**Example problems.** In our computational experiments, we use the lower bound given by Theorem 1 in [17] for TalentSched and the same RLB as in [20] for the PSP. We do not detail them in this article for the sake of conciseness.

## 3 Aggregate Dynamic Programming for Decision Diagrams

As stated in the introduction, optimizations techniques based on DP and DDs can prove highly effective [6, 13, 14, 18, 19]. In some cases, however, the state space of the DP models is simply too large and the bounds derived from restricted and relaxed DDs are of little to no use. This can be imputed either to the node selection heuristic or to the relaxation scheme. The *MinLP* heuristic traditionally used favors keeping nodes with the best prefix values. This locally-optimal selection policy may result in the elimination of all nodes that lead to the optimal solution, or even to any feasible solution, particularly in cases of highly constrained problems. In the latter case, the compilation of a restricted DD is a pure waste of time: no feasible solution is found at the end of the compilation, and not even a bound on the objective value can be exploited to reduce the optimality gap. The same phenomenon is detrimental to the usefulness of compiled relaxed DDs whose bounds might be of low quality when the node selection heuristic is oblivious to the global structure of the problem. Indeed, the merging operator yields a loose representation when applied to an arbitrary set of nodes for most problems. In the absence of a perfect heuristic, this situation will occur under certain conditions. It inspired our pursuit of a more globally-focused approach that could enhance the usefulness of the compiled DDs. This section presents a framework for integrating *aggregate dynamic programming* ideas with DD-based optimization that aims to address some of these shortcomings. Instead of relaxing the original problem by reasoning on merged states, it proposes to use problem instance and state aggregation operators that yield a simpler and relaxed version of the problem, which can be solved exactly. Solutions of the aggregated problem can provide bounds that capture the global problem structure, as well as guidance for the compilation of restricted DDs. This section details the role and meaning of the components of the framework one by one.

### 3.1 Preprocessing: Problem Instance Aggregation

The goal of this preprocessing step is to create an aggregate and simpler problem instance by reducing one or more dimensions of the problem. The *instance aggregation operator*  $\Pi$  must be defined such that the aggregate problem instance  $\mathcal{P}' = \Pi(\mathcal{P})$  is a relaxation of the original

problem instance  $\mathcal{P}$ . In practice, assuming the problem reasons over a set of *elements*, a clustering algorithm can be used to create clusters of such elements. Then, the aggregate problem instance can be obtained by considering *aggregate* elements that encompass all elements in a given cluster and by adapting the instance data accordingly. Formally, if a set  $E$  of elements is clustered into  $K$  clusters, we define two mapping functions:  $\Phi : E \rightarrow \{0, \dots, K-1\}$  that gives the cluster for each original element and  $\Phi^{-1} : \{0, \dots, K-1\} \rightarrow 2^E$  that gives the set of original elements for a given cluster.

**TalentSched.** In [17], it is proved that there always exists an optimal solution to the problem in which scenes with the same set of actors are scheduled together. This gives us the opportunity to aggregate the problem by creating  $K$  clusters of scenes that require a similar set of actors, which is plausible to occur in real film shoots. Scenes belonging to the same clusters can then be aggregated by taking the intersection of their actor requirements and adding up their durations. Formally, we write  $\Pi(\mathcal{P} = (N, A, R, D, C)) = (\Pi_N(N), A, \Pi_R(R), \Pi_D(D), C)$  with  $\Pi_N(N) = \{0, \dots, K-1\}$ . The aggregate actor requirements are computed as  $\Pi_R(R) = R'$  with  $R'_i = \bigcap_{j \in \Phi^{-1}(i)} R_j$  for all  $i \in \Pi_N(N)$  and the aggregate durations as  $\Pi_D(D) = D'$  with  $D'_i = \sum_{j \in \Phi^{-1}(i)} D_j$  for all  $i \in \Pi_N(N)$ .

**PSP.** The number of item types considered in a PSP instance dramatically impacts the size of the state space – for instance, the case with only one item type can be solved greedily. Therefore, and because it is not unlikely that the machine will produce several sets of similar items, we propose to cluster item types that have similar stocking and changeover costs. The instance aggregation operator is thus  $\Pi(\mathcal{P} = (I, S, C, H, Q)) = (\Pi_I(I), \Pi_S(S), \Pi_C(C), H, \Pi_Q(Q))$ , where the aggregate set of item types is given by  $\Pi_I(I) = \{0, \dots, K-1\}$ . Their stocking costs are computed as  $\Pi_S(S) = S'$  with  $S'_k = \min_{i \in \Phi^{-1}(k)} S_i$  for all  $k \in \Pi_I(I)$  and the changeover costs as  $\Pi_C(C) = C'$  with  $C'_{kl} = \min_{i \in \Phi^{-1}(k), j \in \Phi^{-1}(l)} C_{ij}$  for all  $k, l \in \Pi_I(I)$ . The aggregate demand matrix is defined as  $\Pi_Q(Q) = Q'$  with  $Q'_p = \sum_{i \in \Phi^{-1}(k)} Q_p^i$ . However, as the demand matrix is only supposed to contain unit demands, one must redistribute surplus demands in  $Q'$  to the left.

**ALP.** Similarly to the item types of the PSP, the aircraft classes can be aggregated to reduce the complexity of the problem. We thus propose to cluster them based on their minimum separation time with other classes and define the instance aggregation operator as  $\Pi(\mathcal{P} = (N, R, C, A, S, T, L)) = (N, R, \Pi_C(C), \Pi_A(A), \Pi_S(S), T, \Pi_L(L))$ . The set of aircrafts, their target landing time and the number of runways is kept. The aggregate set of classes is given by  $\Pi_C(C) = \{0, \dots, K-1\}$  and their corresponding set of aircrafts is computed as  $\Pi_A(A) = A'$  with  $A'_i = \bigcup_{j \in \Phi^{-1}(i)} A_j$  for all  $i \in \Pi_C(C)$ . The smallest separation times between aggregate classes are kept, as formalized by  $\Pi_S(S) = S'$  with  $S'_{kl} = \min_{i \in \Phi^{-1}(k), j \in \Phi^{-1}(l)} S_{i,j}$  for all  $k, l \in \Pi_C(C)$ . Finally, the aggregation operator adapts the latest landing times of all the aircrafts so that any aircraft with a given target landing time has a greater latest landing time than all other aircrafts of the same class with a smaller target landing time:  $\Pi_L(L) = L'$  with  $L'_i = \max\{L_j \mid \Phi(i) = \Phi(j), T_i \leq T_j\}$  for all  $i \in A$ . This property is assumed to hold for the original problem instance, and must be preserved so that aircrafts from the same class can be scheduled sequentially in the DP model.

► **Example 3.** Let us apply the problem instance aggregation to our running example by creating  $K = 2$  aggregate scenes. Assuming the following clustering is found:  $\Phi(0) = 0, \Phi(1) = 1, \Phi(2) = 0, \Phi(3) = 1$  or equivalently  $\Phi^{-1}(0) = \{0, 2\}, \Phi^{-1}(1) = \{1, 3\}$ . We thus compute the aggregate scene durations as:  $D' = \langle D_0 + D_2, D_1 + D_3 \rangle = \langle 5, 9 \rangle$  and the aggregate actor requirements as:  $R' = \langle \{0, 3\} \cap \{0, 2, 3\}, \{0, 1, 3\} \cap \{0, 1, 2, 3\} \rangle = \langle \{0, 3\}, \{0, 1, 3\} \rangle$ .



### 3.2 State Aggregation and Lower Bound

A second mapping function accompanies the problem instance aggregation operator: the *state aggregation operator*  $\pi : S \rightarrow S'$  that projects each state of the state space  $S$  of the original problem in the aggregate state space  $S'$ . The role of this operator is to translate each original state to its aggregate version by adapting the state information to fit the aggregate problem data. Let us denote by  $\mathcal{B}$  and  $\mathcal{B}'$  the exact DD for problem  $\mathcal{P}$  and  $\Pi(\mathcal{P})$ , respectively. If the aggregation operators  $\Pi$  and  $\pi$  are defined such that  $v^*(u \rightsquigarrow t \mid \mathcal{B}) \geq v^*(u' \rightsquigarrow t' \mid \mathcal{B}')$  for all  $u \in \mathcal{B}, u' \in \mathcal{B}'$  with  $\pi(\sigma(u)) = \sigma(u')$  and  $\pi(\sigma(t)) = \sigma(t')$ , then  $v^*(u' \rightsquigarrow t' \mid \mathcal{B}')$  can be used as a lower bound in the original problem, which we will denote by  $\underline{v}_{agg}(\pi(\sigma(u)))$ .

Assuming the aggregate problem can be pre-solved exactly and the solution of each subproblem is stored, this aggregation-based lower bound can be retrieved very quickly. One way to exploit it is to incorporate it in the RLB as shown at line 9 of Algorithm 1 so that it is used as often as possible. Another possibility would be to use the aggregate state space to replace the state merging scheme in relaxed DDs. Once a layer with greater width than  $W$  is reached, all the states contained in the nodes of the layer could be mapped to the aggregate state space to pursue the compilation in a lower dimensional space.

**TalentSched.** The state compression operator for TalentSched is somewhat complex because we can only map to states where complete aggregate scenes have yet to be scheduled. As a result, if a state  $s$  contains scenes in  $s.P$  that can optionally be scheduled, we map it to a dummy aggregated state. The same logic is applied when  $s.M$  only contains a subset of the scenes that compose an aggregate scene.

$$\pi(s) = \begin{cases} (\emptyset, \emptyset), & \text{if } s.P \neq \emptyset, \\ (\emptyset, \emptyset), & \text{if } \exists i \in \Pi_N(N) : (\Phi^{-1}(i) \cap s.M) \neq \emptyset \wedge \Phi^{-1}(i) \not\subseteq s.M, \\ (M', \emptyset), & \text{otherwise, with } M' = \{i \in \Pi_N(N) \mid \Phi^{-1}(i) \subseteq s.M\}. \end{cases}$$

**PSP.** If we extend the definition of  $\Phi$  such that  $\Phi(\perp) = \perp$ , the state aggregation operator can be defined as  $\pi(s) = (\Phi(s.i), R)$  with  $R_i = \sum_{j \in \Phi^{-1}(i)} s.R_j$  for all  $i \in \Pi_I(I)$ . The item type is projected to its corresponding aggregate type, and the remaining number of items to produce for each type is separately accumulated within each cluster.

**ALP.** Again, assuming  $\Phi(\perp) = \perp$ , the state aggregation operator is defined by  $\pi(s) = (Q', ROP')$  with the remaining quantities of aircrafts aggregated as  $Q'_i = \sum_{j \in \Phi^{-1}(i)} s.Q_j$  for all  $i \in \Pi_C(C)$ . For the ROP, one only needs to adapt the class of the last aircraft scheduled on each runway  $ROP'_i = (s.ROP_0.l, \Phi(s.ROP_0.c))$  for all  $i \in R$ .

If lower bounds for original states are obtained only by pre-solving the aggregate problem, it is unlikely that the solution of an aggregate subproblem mapped with the state aggregation operator will be available, since the aggregate separation times between aircraft classes lead to very different landing times. However, a lower bound for an aggregate state  $s^1 = (Q^1, ROP^1)$  can be provided by the solution of any state  $s^2 = (Q^2, ROP^2)$  such that  $Q^1 = Q^2$  and  $ROP_i^1.c = ROP_i^2.c$  and  $ROP_i^1.l \geq ROP_i^2.l$  for all  $i \in R$ .

► **Example 4.** Let us compute the aggregation-based lower bound for the root state of the running example  $r = (\{0, 1, 2, 3\}, \emptyset)$  given its aggregate version  $\pi(r) = (\{0, 1\}, \emptyset)$  and the clustering performed in Example 3. The aggregate version is trivial to solve since the objective function is symmetrical and there are only two scenes to schedule. We thus have  $\underline{v}_{agg}(r) = D'_0 \times (C_0 + C_3) + D'_1 \times (C_0 + C_1 + C_3) = 5 \times (1 + 4) + 9 \times (1 + 2 + 4) = 88$ , which is a slightly better lower bound than the one obtained with the relaxed DD of Example 2.

### 3.3 Solution Disaggregation and Node Selection Heuristic

In order to exploit the solution of the aggregate version of a subproblem to find good heuristic solutions for the original subproblem, we need to specify the correspondence between decisions in the aggregate problem with decisions in the original problem. We therefore define a last modeling component, called the *decision disaggregation operator*  $\delta(d) : D'_k \rightarrow 2^{D_i} \times \dots \times 2^{D_j}$  that maps the instantiation of a variable  $x'_k$  in the aggregate problem to a vector of possible corresponding assignments for variables  $x_i, \dots, x_j$  in the original problem.

Finally, we define the *path disaggregation operator* that transforms a sequence of decisions in the aggregate problem to a sequence of sets of possible decisions in the original problem:  $\Delta(p = (a_k, \dots, a_{n'-1})) = \delta(l(a_k)) \cdot \dots \cdot \delta(l(a_{n'-1}))$  where  $n'$  is the supposed number of aggregate variables and  $\cdot$  denotes the *concatenation* of two vectors. Using this operator, we can compute a *score* for each decision made during the compilation of restricted DDs. At line 3 of Algorithm 1, we first retrieve the optimal value assignment of the aggregate subproblem and apply the path disaggregation operator on it. Then, a binary *score* is attributed to each arc at line 15, depending on its compatibility with the disaggregated solution. At line 2 of Algorithm 2, the maximum score obtained along any path up to each node can then be used to order nodes from most to least promising, favoring nodes with incoming paths that are highly compatible with the disaggregated solution. By doing so, the width of restricted DDs is controlled in the same way as before, enabling the preference of solutions even when no feasible solution with the maximum possible score is available.

**TalentSched.** Each aggregate scene corresponds to a set of original scenes, we thus need to map each aggregate decision to a sequence of original decisions:  $\delta(i) = V$  where  $V_j = \Phi^{-1}(i)$  for all  $0 \leq j < |\Phi^{-1}(i)|$ . It corresponds to any of the scenes from the cluster  $i$ , duplicated  $|\Phi^{-1}(i)|$  times so that they are all scheduled one after another, preferably.

**PSP.** The operator is much simpler to define for the PSP, since each decision concerns the production of one unit of a chosen aggregate item type. It can thus be interpreted as the decision of producing one unit of any item type in the corresponding cluster:  $\delta(i) = \langle \Phi^{-1}(i) \rangle$ .

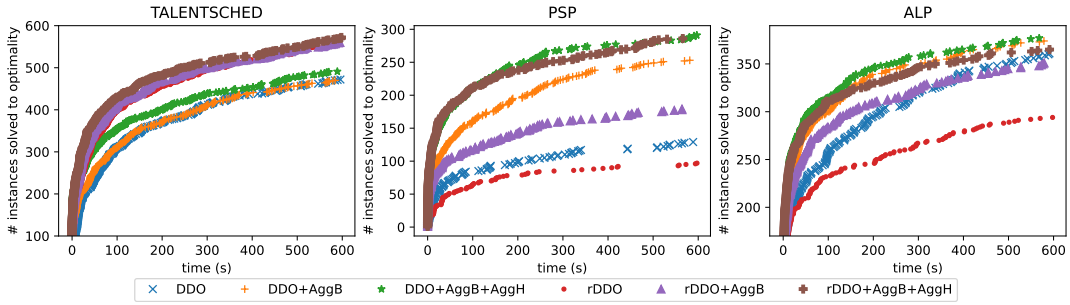
**ALP.** The only difference with the PSP is that decisions also contain the runway on which the aircraft is scheduled to land, which remains the same:  $\delta(a, r) = \langle \{(a', r) \mid a' \in \Phi^{-1}(a)\} \rangle$ .

► **Example 5.** As computed in Example 4, the schedule  $\langle 0, 1 \rangle$  is optimal for the aggregate problem. By disaggregating this solution, we get  $\langle \{0, 2\}, \{0, 2\}, \{1, 3\}, \{1, 3\} \rangle$ . We can notice that the optimal schedule  $\langle 0, 2, 3, 1 \rangle$  found in Example 1 is compatible with the disaggregated solution and would thus be favored by the aggregation-based node selection heuristic.

## 4 Computational Experiments

The impact of the aggregation-based bounds and heuristics was evaluated experimentally by extending the generic DD-based solver DDO [21] and injecting the modeling of the three discrete optimization problems presented throughout the paper. The version of DDO used includes the improvements introduced in [16, 19]. For each problem, random instances were generated with the following main parameters:

- TalentSched: number of scenes  $n \in \{20, 22, 24, 26, 28\}$ , number of actors  $m \in \{10, 15\}$  and average fraction of actors required for each scene  $\rho \in \{0.3, 0.4\}$ .



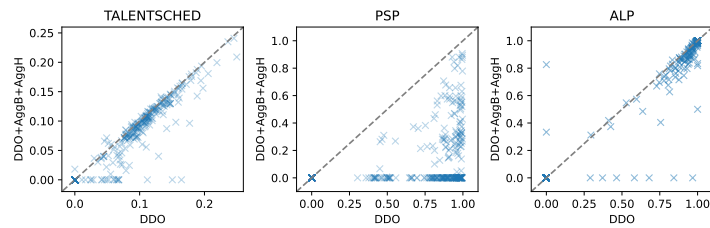
■ **Figure 3** Number of instances solved over time for each configuration and problem studied.

- PSP: number of item types  $n = 10$ , horizon  $H \in \{100, 150, 200\}$  and fraction of time periods with a demand  $\rho \in \{0.9, 0.95, 1\}$ .
- ALP: number of aircrafts  $n \in \{25, 50, 75, 100\}$ , number of runways  $r \in \{1, 2, 3, 4\}$ , number of aircraft classes  $c = 4$  and mean inter-arrival time  $40/r$  for generating the target landing times according to a Poisson arrival process.

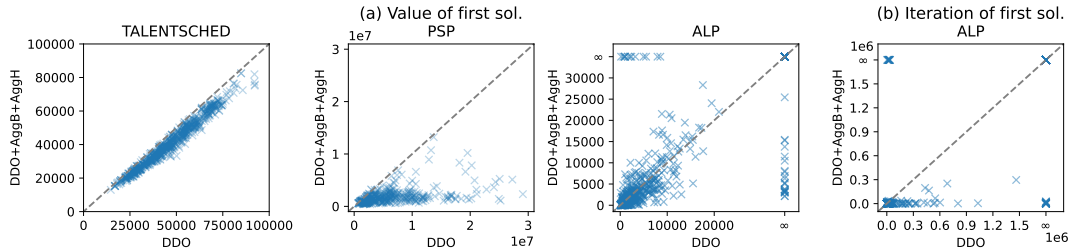
Furthermore, the instance generation tries to emulate an increasing number of groups of actor requirements, item types and aircraft classes that lend themselves more or less to aggregation. Each instance was presolved in its aggregate state space after aggregating its data according to  $k$ -means clustering for PSP and ALP and a custom hierarchical clustering for TalentSched that tries to maximize the remaining costs induced by the actor requirements. TalentSched instances can be presolved exactly with 20 aggregate scenes and PSP instances similarly with 5 aggregate item types. On the other hand, not all ALP instances reduced to 2 aggregate aircraft have a reasonable number of states so we employ a relaxed DD with maximum width 10000 for the presolving part instead. Note that the present approach does not compete with the state-of-the-art for TalentSched as it lacks much of the custom symmetry-breaking logic introduced in [17] and similarly for ALP regarding the dominance-breaking constraints presented in [28]. Six different configurations were created by combining the default DD-based solved DDO on one hand and a version using only restricted DDs and no relaxed DDs, denoted rDDO, on the other hand, with the aggregation-based bounds (AggB) and heuristics (AggH). Ten minutes were allotted for each configuration to solve each instance.

Figure 3 presents the cumulative number of instances solved with respect to the solving time. For TalentSched, it appears that any configuration of rDDO performs better than any of DDO. This suggests that the bounds provided by the relaxed DDs are looser than the RLB while being more expensive to compute. It confirms our intuition that the state merging scheme yields bounds with a limited impact for some problems, probably because the state information gets very dilute when many states are merged together. In this case, the RLB computation is also quite involved – see [17]. Still, adding the AggB and the AggH to either configurations improves the results by a small margin, although not that significant. This can be contrasted with the results obtained for the two other problems, which show a clear improvement when the AggB and the AggH are added to either configurations. Furthermore, in cases where rDDO alone yields the worst results, incorporating AggB leads to results that are similar to or better than those achieved by DDO. Combining it with the AggH performs better than DDO in both cases and almost equally well than DDO+AggB+AggH.

The impact of the AggB and the AggH can also be measured in terms of end gap  $\frac{UB-LB}{UB}$ . Figure 4 compares the end gap obtained for each instance by DDO and DDO+AggB+AggH. It shows that except for a few instances, DDO+AggB+AggH is always closer to terminating the search than DDO, especially for PSP. To validate the relevance of the AggH, we also



■ **Figure 4** Comparison of the end gap obtained for each instance by DDO and DDO+AggB+AggH.



■ **Figure 5** Comparison of the value of the first solution found by DDO and DDO+AggB+AggH, and of the iteration at which the solution is found for ALP.

compare the value of the first solution found by DDO and DDO+AggB+AggH on Figure 5(a). For TalentSched and PSP, the quality of the first solution is always better when using the AggH. However, there is no clear trend for the ALP. Unlike TalentSched and PSP, for which a solution is always found at the first iteration, the landing time windows of ALP make it difficult to find a feasible solution. This explains both the end gaps close to one in Figure 4 and the  $\infty$  values in Figure 5(a), which represent the absence of a feasible solution. We thus compare on Figure 5(b) the iteration at which the first solution is found. We observe that DDO+AggB+AggH finds a feasible solution much earlier than DDO in most cases. This showcases well the benefits of a node selection heuristic with a more global awareness.

## 5 Conclusion

This paper explained how ideas from aggregate dynamic programming can be incorporated in DD-based optimization solvers. We proposed to derive lower bounds and node selection heuristics from a pre-solved aggregate version of the original problem at hand, and explained how these can be seamlessly added to the DD-based optimization framework. Computational experiments on three different problems showed that they provide lower bounds that further strengthen the current approach, and that could even be used as a replacement for relaxed DDs in some cases. Furthermore, the aggregation-based node selection heuristics were shown very valuable as they manage to steer the compilation of relaxed DDs toward better solutions earlier in the search. When applying this idea to a highly constrained problem, the heuristics proved to quickly lead to feasible solutions that were hard to find otherwise. These results suggest that aggregation-based bounds and heuristics capture global problem structures well, as opposed to the greedy *MinLP* heuristic traditionally used to compile approximate DDs.

---

**References**

---

- 1 Henrik Reif Andersen, Tarik Hadzic, John N Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 118–132. Springer, 2007.
- 2 Sven Axsäter. State aggregation in dynamic programming—an application to scheduling of independent jobs on parallel processors. *Operations Research Letters*, 2(4):171–176, 1983.
- 3 James C Bean, John R Birge, and Robert L Smith. Aggregation in dynamic programming. *Operations Research*, 35(2):215–220, 1987.
- 4 Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, November 1954. URL: <https://projecteuclid.org/443/euclid.bams/1183519147>.
- 5 David Bergman and Andre A. Cire. On finding the optimal bdd relaxation. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, volume 10335 of *LNCS*, pages 41–50. Springer, 2017.
- 6 David Bergman, Andre A Cire, Ashish Sabharwal, Horst Samulowitz, Vijay Saraswat, and Willem-Jan van Hoeve. Parallel combinatorial optimization with decision diagrams. In *Integration of AI and OR Techniques in Constraint Programming: 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings 11*, pages 351–367. Springer, 2014.
- 7 David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Variable ordering for the application of bdds to the maximum independent set problem. In *International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming*, pages 34–49. Springer, 2012.
- 8 David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2014.
- 9 David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.
- 10 David Bergman, Andre A Cire, Willem-Jan van Hoeve, and Tallys Yunes. Bdd-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234, 2014.
- 11 Quentin Cappart, David Bergman, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Augustin Parjadis. Improving variable orderings of approximate decision diagrams using reinforcement learning. *INFORMS Journal on Computing*, 34(5):2552–2570, 2022.
- 12 Quentin Cappart, Emmanuel Goutierre, David Bergman, and Louis-Martin Rousseau. Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1443–1451, 2019.
- 13 Margarita P Castro, Andre A Cire, and J Christopher Beck. An mdd-based lagrangian approach to the multicommodity pickup-and-delivery tsp. *INFORMS Journal on Computing*, 32(2):263–278, 2020.
- 14 Margarita P Castro, Chiara Piacentini, Andre Augusto Cire, and J Christopher Beck. Solving delete free planning with relaxed decision diagram based heuristics. *Journal of Artificial Intelligence Research*, 67:607–651, 2020.
- 15 Edmund M Clarke, Orna Grumberg, and David E Long. Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.
- 16 Vianney Coppé, Xavier Gillard, and Pierre Schaus. Decision diagram-based branch-and-bound with caching for dominance and suboptimality detection, 2023. [arXiv:2211.13118](https://arxiv.org/abs/2211.13118).
- 17 Maria Garcia de la Banda, Peter J Stuckey, and Geoffrey Chu. Solving talent scheduling with dynamic programming. *INFORMS Journal on Computing*, 23(1):120–137, 2011.

- 18 Xavier Gillard. *Discrete optimization with decision diagrams: design of a generic solver, improved bounding techniques, and discovery of good feasible solutions with large neighborhood search*. PhD thesis, UCL-Université Catholique de Louvain, 2022.
- 19 Xavier Gillard, Vianney Coppé, Pierre Schaus, and André Augusto Cire. Improving the filtering of branch-and-bound mdd solver. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 231–247. Springer, 2021.
- 20 Xavier Gillard and Pierre Schaus. Large neighborhood search with decision diagrams. In *International Joint Conference on Artificial Intelligence*, 2022.
- 21 Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a generic and efficient framework for mdd-based optimization. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 5243–5245, 2021.
- 22 Jaime E Gonzalez, Andre A Cire, Andrea Lodi, and Louis-Martin Rousseau. Integrated integer programming and decision diagram search tree with an application to the maximum independent set problem. *Constraints*, pages 1–24, 2020.
- 23 John N. Hooker. Improved job sequencing bounds from decision diagrams. In Thomas Schiex and Simon de Givry, editors, *Principles and Practice of Constraint Programming*, volume 11802 of *LNCS*, pages 268–283. Springer, 2019.
- 24 Matthias Horn, Johannes Maschler, Günther R Raidl, and Elina Rönnberg. A\*-based construction of decision diagrams for a prize-collecting scheduling problem. *Computers & Operations Research*, 126:105125, 2021.
- 25 Alan J. Hu. *Techniques for efficient formal verification using binary decision diagrams*. PhD thesis, Stanford University, Department of Computer Science, 1995.
- 26 Anthony Karahalios and Willem-Jan van Hoeve. Variable ordering for decision diagrams: A portfolio approach. *Constraints*, 27(1):116–133, 2022.
- 27 C.-Y. Lee. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999, 1959.
- 28 Alexander Lieder, Dirk Briskorn, and Raik Stolletz. A dynamic programming approach for the aircraft landing problem with aircraft classes. *European Journal of Operational Research*, 243(1):61–69, 2015.
- 29 Shin-ichi Minato. *Binary decision diagrams and applications for VLSI CAD*, volume 342. Springer Science & Business Media, 1995.
- 30 Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau. Peel-And-Bound: Generating Stronger Relaxed Bounds with Multivalued Decision Diagrams. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 31 Christian Tjandraatmadja. *Decision Diagram Relaxations for Integer Programming*. PhD thesis, Carnegie Mellon University Tepper School of Business, 2018.
- 32 Christian Tjandraatmadja and Willem-Jan van Hoeve. Target cuts from relaxed decision diagrams. *INFORMS Journal on Computing*, 31(2):285–301, 2019. doi:10.1287/ijoc.2018.0830.



# Fast Matrix Multiplication Without Tears: A Constraint Programming Approach

Arnaud Deza<sup>1</sup> ✉

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Chang Liu<sup>1</sup> ✉

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Pashootan Vaezipoor ✉

Department of Computer Science, University of Toronto, Canada

Elias B. Khalil ✉

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

---

## Abstract

It is known that the multiplication of an  $N \times M$  matrix with an  $M \times P$  matrix can be performed using fewer multiplications than what the naive  $NMP$  approach suggests. The most famous instance of this is Strassen's algorithm for multiplying  $2 \times 2$  matrices in 7 instead of 8 multiplications. This gives rise to the constraint satisfaction problem of fast matrix multiplication, where a set of  $R < NMP$  multiplication terms must be chosen and combined such that they satisfy correctness constraints on the output matrix. Despite its highly combinatorial nature, this problem has not been exhaustively examined from that perspective, as evidenced for example by the recent deep reinforcement learning approach of AlphaTensor. In this work, we propose a simple yet novel Constraint Programming approach to find algorithms for fast matrix multiplication or provide proof of infeasibility otherwise. We propose a set of symmetry-breaking constraints and valid inequalities that are particularly helpful in proving infeasibility. On the feasible side, we find that exploiting solver performance variability in conjunction with a sparsity-based problem decomposition enables finding solutions for larger (feasible) instances of fast matrix multiplication. Our experimental results using CP Optimizer demonstrate that we can find fast matrix multiplication algorithms for matrices up to  $3 \times 3$  with  $R = 23$  in a short amount of time.

**2012 ACM Subject Classification** Mathematics of computing

**Keywords and phrases** fast matrix multiplication, computer-assisted proofs, constraint programming, constraint satisfaction problem

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.14

**Supplementary Material**

*Software (Source Code):* <https://github.com/khalil-research/Matrix-Mult-CP/tree/main>

## 1 Introduction

Matrix multiplication is a fundamental operation in linear algebra with applications in virtually every computational domain. As a result, extensive research has been dedicated to the development of faster matrix multiplication algorithms.

The elementary way of multiplying two  $N \times N$  matrices requires  $N^3$  multiplications. For example, multiplying two  $2 \times 2$  matrices naively requires a total of  $2^3 = 8$  multiplications. In 1969, Strassen [16] constructed an algorithm that finds the product of two  $2 \times 2$  matrices in only 7 multiplications. This discovery has had significant implications as it opened up

---

<sup>1</sup> These authors contributed equally.





the door for potentially faster algorithms for large-scale matrix or tensor computations. Strassen’s algorithm has later been proved to be both canonical [4] (no smaller rank exists) and essentially unique [6] (all other solutions of the same rank are equivalent up to symmetry).

Currently, the best-known algorithm for multiplying  $3 \times 3$  matrices requires  $R = 23$  multiplications, compared to the naive elementary method that requires 27 multiplications. A known theoretical lower bound of  $R = 19$  exists [2], however, it remains unclear whether  $19 \leq R \leq 22$  is truly attainable. This is a testament to the difficulty of the *fast matrix multiplication* (FMM) problem, which has been intractable for existing methods even for tiny matrices.

In the literature, the general approach to finding FMM algorithms starts by representing matrix multiplication as a tensor operation using the multiplication tensor  $T_N$  followed by finding exact or approximate low-rank decompositions that represent  $T_N$ . The factor matrices that are used in the low-rank decomposition encode FMM algorithms. A rank-7 decomposition (i.e., a multiplication algorithm that uses 7 multiplication operations) of a  $2 \times 2$  matrix multiplication using Strassen’s algorithm is shown in Figure 1. Existing methods for finding such factor matrices have several limitations. The most successful and common methods include local search [13] techniques for low-rank approximation, which cannot guarantee optimality. A more recent successful approach [7] searches for low-rank decomposition using *reinforcement learning* (RL) and was successful in finding faster algorithms for  $N = 4$ . However, this method is not exhaustive and hence cannot prove the infeasibility of a given rank.

In this work, we propose a novel approach to finding FMM algorithms by formulating the tensor decomposition problem, for the first time, as a *constraint satisfaction problem* (CSP) that is solved using *Constraint Programming* (CP). We believe that this is a very natural formulation of this highly combinatorial problem. CP is advantageous for FMM in that it is a flexible framework that can bring to bear a wide range of search and logical inference techniques that have been developed over the last few decades. It provides the ability to prove infeasibility when it is not possible to multiply two matrices using a given number of multiplications.

Besides a base CP formulation for FMM, we propose a set of symmetry-breaking constraints and valid inequalities that are useful for infeasibility proofs. On the feasible side, we show that “performance variability” w.r.t. solver random seeds can be exploited in conjunction with a sparsity-based decomposition of FMM for faster solving. Our experimental results, while limited to matrices of size up to  $3 \times 3$ , demonstrate the effectiveness of the aforementioned constraints and techniques. The CP approach to FMM is uniquely positioned to close open questions such as whether it is possible to multiply two  $3 \times 3$  matrices in 19 to 22 multiplications. While we do not yet resolve this or other open questions, our work opens up the potential for further enhancements to the CP formulation and search such as customized branching strategies and CP-based heuristics.

## 2 Fast Matrix Multiplication: Problem Statement

The multiplication of two matrices  $A$  and  $B$  of sizes  $N \times M$  and  $M \times P$ , respectively, results in a product matrix  $C$  of size  $N \times P$ . This operation can be represented by a binary third-order tensor  $T_{NMP}$  ( $T_N$  for square matrices  $A$  and  $B$  of size  $N \times N$ ). An entry  $T_{i,j,k}$  of this tensor is equal to 1 if and only if the  $k^{\text{th}}$  entry in the output matrix  $C$  uses the scalar product of the  $i^{\text{th}}$  entry of  $A$  and the  $j^{\text{th}}$  entry of  $B$ . Here,  $i$ ,  $j$ , and  $k$  are indices of a matrix entry starting with 1 in the first row and column; and proceeding entry by entry, left to right, top

to bottom. For example, for  $N = M = P = 2$ , it must be that  $T_{2,3,1} = 1$  since the first entry of  $C$ ,  $c_1$ , is equal to  $a_1b_1 + \mathbf{a}_2\mathbf{b}_3$ . Similarly,  $T_{1,2,1} = 0$  must hold since  $a_1b_2$  is not part of  $c_1$ . Figures 1a and 1b show a complete example of the indexing and tensor representation.

The FMM problem for a given tensor  $T_{NMP}$ , rank  $R \in \mathbb{Z}^+$ , and field  $\mathbb{F}$  (e.g.,  $\mathbb{F} = \{-1, 0, 1\}$ ) asks: can each entry  $T_{i,j,k}$  of  $T_{NMP}$  be expressed as the sum of exactly  $R$  trilinear terms involving the *factor matrices*  $U \in \mathbb{F}^{N \cdot M \times R}$ ,  $V \in \mathbb{F}^{M \cdot P \times R}$ , and  $W \in \mathbb{F}^{N \cdot P \times R}$ , as follows:

$$T_{i,j,k} = \sum_{r=1}^R U_{i,r} \cdot V_{j,r} \cdot W_{k,r} \quad \forall i \in \{1, \dots, N \cdot M\}, j \in \{1, \dots, M \cdot P\}, k \in \{1, \dots, N \cdot P\}$$

Note that we use the notation  $\mathbb{F}^{L \times Q}$  to refer to the set of matrices of dimension  $L \times Q$  and entries in  $\mathbb{F}$ . The CSP is to find factor matrices with entries in  $\mathbb{F}$  that produce the tensor  $T_{NMP}$  for a given rank  $R$ .

This decomposition is also referred to as the polyadic decomposition and its associated rank is the minimal  $R$  needed. The rank can be interpreted as the number of multiplications required to compute the product. For example, for  $2 \times 2$  matrices, the rank of the decomposition using Strassen's algorithm is 7. Figure 1 walks through an example of the low-rank decomposition of a  $2 \times 2$  matrix multiplication using Strassen's algorithm. The matrix multiplication of the two  $2 \times 2$  matrices can be seen in Figure 1a, its associated tensor representation  $T_N$  in Figure 1b, the low-rank decomposition in Figure 1c, and the factor matrices  $U$ ,  $V$ , and  $W$  in Figure 1d.

### 3 Related Work

Since Strassen's discovery [16], there has been substantial research on finding faster algorithms for matrix multiplication. Mathematicians have discovered such algorithms manually over the years for a variety of matrix dimensions and ranks. In this section, however, we will focus on automated methods for discovering such algorithms and briefly discuss some of the existing methods. A recent survey on the topic can be found in [3].

#### 3.1 Continuous Local Search Methods

The most common approach in the literature to compute the factor matrices  $U$ ,  $V$ , and  $W$  is to use (heuristic, continuous) local search methods for low-rank tensor decomposition. The state-of-the-art local method [13] uses alternating least squares with regularization. This method has been the most successful in finding fast algorithms whilst remaining computationally tractable and has been scaled up to  $N = M = P = 4$ ,  $R = 49^2$ . However, this approach has limitations which include getting stuck at local minima, facing ill-conditioned linear least-squares problems, and solutions being only adequate up to machine precision. Additionally, these methods are not exhaustive and hence cannot be used to provide a proof of infeasibility for a given rank  $R$ .

#### 3.2 AlphaTensor

More recently, DeepMind released AlphaTensor [7], a deep RL method that searches this large combinatorial space by playing a single-player game, the TensorGame, formulated as a *Markov decision process* (MDP). At every step  $t$  of this MDP, the state is characterized

<sup>2</sup> Note that this particular result is not very useful as an  $R = 49$  solution can be obtained by applying Strassen's  $R = 7$  algorithm for  $2 \times 2$  matrices on the four  $2 \times 2$  blocks of the  $4 \times 4$  matrices.

## 14:4 Fast Matrix Multiplication Without Tears: A Constraint Programming Approach

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

(a) Multiplication of two  $2 \times 2$  matrices. We highlight the term  $c_1 = a_1b_1 + a_2b_3$ .

$$T_{::,1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad T_{::,2} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad T_{::,3} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad T_{::,4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(b) Tensor representation of the  $2 \times 2$  matrix multiplication operation.  $T_{::,1}$  represents  $c_1$ , the entry  $T_{2,3,1}$  (in yellow) is set to 1 because the product  $a_2b_3$  is required to compute  $c_1$  (similarly for  $T_{1,1,1}$  in red).

$$\begin{aligned} m_1 &= (a_1 + a_4)(b_1 + b_4) & m_5 &= (a_1 + a_2)(b_4) \\ m_2 &= (a_3 + a_4)(b_1) & m_6 &= (a_3 - a_1)(b_1 + b_2) \\ m_3 &= (a_1)(b_2 - b_4) & m_7 &= (a_2 - a_4)(b_3 + b_4) \\ m_4 &= (a_4)(b_3 - b_1) \end{aligned}$$

$$\begin{aligned} c_1 &= m_1 + m_4 - m_5 + m_7 \\ &= (a_1 + a_4)(b_1 + b_4) + (a_4)(b_3 - b_1) - (a_1 + a_2)(b_4) + (a_2 - a_4)(b_3 + b_4) \\ &= a_1b_1 + \cancel{a_1b_4} + \cancel{a_4b_1} + \cancel{a_4b_4} + a_4b_3 - \cancel{a_1b_1} - \cancel{a_1b_4} - \cancel{a_2b_4} + a_2b_3 + \cancel{a_2b_4} - \cancel{a_4b_3} - \cancel{a_4b_4} \\ &= a_1b_1 + a_2b_3 \\ c_2 &= m_3 + m_5 \\ c_3 &= m_2 + m_4 \\ c_4 &= m_1 - m_2 + m_3 + m_6 \end{aligned}$$

(c) A low-rank decomposition of the  $2 \times 2$  matrix multiplication using Strassen's algorithm. The  $m$  terms are the multiplication terms and the  $c$  terms represent the entries in the product matrix. Here  $c_1 = m_1 + m_4 - m_5 + m_7$  gives  $c_1 = a_1b_1 + a_2b_3$  after expansion.

$$\begin{aligned} U &= \begin{pmatrix} m_1 & m_2 & m_3 & m_4 & m_5 & m_6 & m_7 \\ 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} \begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{matrix} \\ V &= \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{matrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{matrix} \\ W &= \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{matrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{matrix} \end{aligned}$$

(d) The factor matrices  $U$ ,  $V$ , and  $W$  for Strassen's algorithm. The columns in  $U$  and  $V$  represent the coefficient of the  $a$  and  $b$  terms in each  $m$ . Each row in  $W$  represents the coefficient of the  $m$  terms in one  $c$  term.

■ **Figure 1** A low-rank decomposition of a  $2 \times 2$  matrix multiplication using Strassen's algorithm.

by a tensor  $S_t$  which is initially set to the target multiplication tensor, i.e.,  $S_0 = T_N$ . An action  $a_t$  at iteration  $t$  corresponds to the player selecting a triplet of vectors  $(u^{(t)}, v^{(t)}, w^{(t)})$  which in turn will provide the next state  $S_t = S_{t-1} - u^{(t)} \otimes v^{(t)} \otimes w^{(t)}$  where  $\otimes$  denotes the outer tensor product. The goal of the player is to reach the zero tensor  $S_t = \mathbf{0}$  in the fewest number of steps possible. This is done by providing a reward of  $-1$  to the player after every non-terminal state whereas a large negative reward  $-\gamma(S_{R_{\text{limit}}})$  is given to the player if the number of steps  $R_{\text{limit}}$  is met, where  $\gamma(S_{R_{\text{limit}}})$  upper bounds the rank of the tensor at iteration  $R_{\text{limit}}$ . If the agent successfully reaches the zero tensor, the sequence of actions taken constitutes a valid low-rank decomposition of  $T_N$ , and hence an FMM algorithm is found with the rank  $R$  corresponding to the number of steps taken by the agent.

This approach is the first to directly incorporate learning into the search which resulted in the discovery of new minimal ranks for certain non-trivial cases. The largest case tackled by this method is  $N = M = P = 5$ ,  $R = 98$ . The sole focus of this purely heuristic method is to find lower ranks than currently best-known ranks but it cannot prove the infeasibility of a given rank. Additionally, rather complex architectures and multiple training phases were required for successful learning. It is worth noting that AlphaTensor was trained for one week on 64 Tensor Processing Units (TPUs), Google’s proprietary chip. The paper [7] does not provide any estimates of the amount of computation required to produce the reported results, namely how long the trained “agent” must be run to discover FMM algorithms. Our CP runs use much fewer resources while leveraging thread parallelism in the CP solver on readily-available CPU machines.

### 3.3 Integer Programming

The work that is the most related to our approach tackles this problem through a *mixed-integer linear program* (MILP) formulation in an unpublished technical report [14]. The goal of this methodology is to linearize the trilinear products in the low-rank decomposition of  $T_N$  to a MILP that aims to 1) maximize the sparsity of the integer decision variables representing factor matrices  $U$ ,  $V$ , and  $W$  and 2) minimize the reconstruction loss (L1 norm) from the input  $T_N$  and the multiplication tensor attained by the decision variables representing factor matrices. The report [14] focuses solely on presenting the MILP formulation for square matrices but does not include any computational experiments. However, the MILP formulations for  $N \in \{2, 3\}$  are benchmark problems in MIPLIB 2017 [9]<sup>3</sup>. The linearization of the trilinear products likely leads to a weak linear programming relaxation as well as an explosion in the number of integer variables and constraints, which might explain why the MILP approach to FMM has not picked up significant interest. A CP formulation is more natural and compact, as we will show in this paper.

### 3.4 Classical AI Planning

Very recently, AI planning techniques were used for FMM [15]. They use a similar state space as AlphaTensor but use various planning tools (with and without exhaustive search) to solve this problem. They compared a number of heuristic and exact planning methods from the literature on matrices of size up to  $3 \times 3$ . However, the experiments show that planning approaches are severely limited, even failing to find Strassen’s algorithm for the  $2 \times 2$  case (see Table 1 in [15]). We will show that our CP approach is significantly more effective as we are able to attack the  $3 \times 3$  case with  $R = 23$ , matching the known upper bound from the literature.

<sup>3</sup> See [https://miplib.zib.de/instance\\_details\\_fastxgemm-n3r21s3t6.html](https://miplib.zib.de/instance_details_fastxgemm-n3r21s3t6.html) for example.

### 3.5 Boolean Satisfiability-based Approaches

*Boolean Satisfiability* (SAT) formulations to solve the FMM problem were first proposed in 2011 [5]. More specifically, the authors studied the multiplication of  $3 \times 3$  matrices and were able to find a new general algorithm with rank 23 with help from SAT solver tweaks and improvements in a few days with one CPU. More recently, Heule et al. improved on the SAT-based methods for the multiplication of  $3 \times 3$  matrices and have successfully found many thousands new general algorithms with rank 23 [10, 11]. In their SAT-formulation, the authors transformed the multiplication to ‘and’ clauses and the addition to ‘xor’ clauses, then a Tseitin transformation is performed to formulate the algebraic FMM problem into a SAT problem.

The authors propose a random pairing method where value assignments are initialized based on observed results from previously known solutions with streamlining constraints to find new general solutions; then local search is used to find additional solutions. The SAT-based formulations of the FMM problem demonstrates to be very difficult for complete SAT solvers, thus making it difficult to provide proof of infeasibility.

## 4 Constraint Programming for Fast Matrix Multiplication

In the FMM problem, all variables have the same domain  $\mathbb{F} = \{-1, 0, 1\}^4$ . Since the variable domains are small and this problem is highly structured, CP is a promising solution paradigm.

The base CP model for FMM is given in Equation (1). Let  $\mathcal{U}$  denote the set  $\{1, \dots, N \cdot M\}$ ,  $\mathcal{V}$  denote the set  $\{1, \dots, M \cdot P\}$ ,  $\mathcal{W}$  denote the set  $\{1, \dots, N \cdot P\}$ , and  $\mathcal{R}$  denote the set  $\{1, \dots, R\}$ . The CP model uses three sets of variables:  $u_{i,r}$  where  $i \in \mathcal{U}$ ,  $v_{j,r}$  where  $j \in \mathcal{V}$ , and  $w_{k,r}$  where  $k \in \mathcal{W}$ ;  $r \in \mathcal{R}$  in all three cases. Each variable  $u_{i,r}$ ,  $v_{j,r}$  and  $w_{k,r}$  represents the value of the  $i/j/k^{\text{th}}$  row and  $r^{\text{th}}$  column of the matrices  $U$ ,  $V$ , and  $W$ . The domain of all variables is  $\{-1, 0, 1\}$ . The set of constraints presented here requires that the decomposition algorithm’s output matches the original tensor multiplication  $T_{NMP}$ . Therefore the input to the CSP model is 4 integers:  $(N, M, P)$  and  $R$ . The model then reads as:

$$\begin{aligned} \sum_{r \in \mathcal{R}} (u_{i,r} \cdot v_{j,r} \cdot w_{k,r}) &= T_{i,j,k}, & \forall i \in \mathcal{U}, j \in \mathcal{V}, k \in \mathcal{W} \\ u_{i,r}, v_{j,r}, w_{k,r} &\in \{-1, 0, 1\}, & \forall i \in \mathcal{U}, j \in \mathcal{V}, k \in \mathcal{W}, r \in \mathcal{R} \end{aligned} \quad (1)$$

The search space for this (NP-complete) problem grows very quickly with increasing matrix sizes  $N, M, P$  and rank  $R$ . With only one set of equality constraints, a CP solver may struggle with constraint propagation, thus failing to scale with increasing  $N, M, P$ . To that end, we will introduce additional valid constraints to help CP prune and propagate more efficiently.

### 4.1 Symmetry Breaking

There are many symmetric solutions to the FMM problem. We can reduce the search space of our problem significantly by prohibiting symmetries.

---

<sup>4</sup> One can consider bigger fields such as  $\{-2, -1, 0, 1, 2\}$  but the bulk of the work in the literature has been with  $\{-1, 0, 1\}$ .

### 4.1.1 Permutation Symmetry

Since addition is commutative, i.e.,  $(a_1 + a_2) = (a_2 + a_1)$ , there are many equivalent solutions to the tensor decomposition problem. Therefore, any permutation of the columns of matrices  $U$ ,  $V$ , and  $W$  produces an equivalent solution. If we consider Strassen's solution for the  $2 \times 2$  case, Figure 2 provides an example of two equivalent solutions.

$$\begin{aligned} \text{sol1: } U &= \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix} & V &= \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix} & W &= \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \\ \text{sol2: } U &= \begin{pmatrix} -1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 1 & 0 \end{pmatrix} & V &= \begin{pmatrix} 1 & -1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 & 1 \end{pmatrix} & W &= \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{aligned}$$

■ **Figure 2** Two equivalent solutions for Strassen's solution of  $2 \times 2$  matrix multiplication. sol2 is the `lexicographic-strict` presentation of this solution.

In order to break this symmetry, we introduce a `lexicographic-strict`<sup>5</sup> constraint on the  $u_{i,r}$  and  $v_{j,r}$  variables. When applied to two variable arrays  $x$  and  $y$ , the lexicographic ordering constraint enforces that  $x$  is strictly less than  $y$  in the defined lexicographic order. Because of the strictness, this also enforces that the two variable arrays must be different. This set of symmetry-breaking constraints is modelled as follows:

$$\text{lexicographic-strict}([u_{:,r}; v_{:,r}], [u_{:,r+1}; v_{:,r+1}]), \quad \forall r \in \mathcal{R}$$

where  $[u_{:,r}; v_{:,r}]$  represents the vector concatenating the  $r^{\text{th}}$  column of the matrix  $U$  and  $V$ . In Figure 2, sol2 satisfies the `lexicographic-strict` constraint.

### 4.1.2 Sign Symmetry

For the multiplicative  $m_i$  terms, one can easily see that multiplying both sets of terms from  $A$  and  $B$  by  $-1$  will result in the same solution. For example,  $(a_1 + a_4)(b_1 + b_4) = (-a_1 - a_4)(-b_1 - b_4)$ , where we could multiply any subset of columns of  $U$  and  $V$  by  $-1$  to achieve the same solution. We call this symmetry the sign symmetry. In order to break it, we introduce the following constraints:

$$\begin{aligned} u_{1,r} &\leq 0 \\ u_{i,r} &\leq \sum_{i'=1}^{i-1} |u_{i',r}| \quad \forall r \in \mathcal{R}, i > 1, i \in \mathcal{U} \end{aligned}$$

The main idea of these constraints is to enforce that the first non-zero entry in a column of  $U$  can only take on the value of  $-1$ , enforcing that the first entry of the columns is either 0 or  $-1$ . The subsequent constraints ensure that for any column  $r$ , an entry in row  $i > 1$  can only be 1 if there has been an entry in the same column in an earlier row with value  $-1$ . This set of constraints applies to the concatenation of the columns in  $U$  and  $V$ , however, in modelling, it only needs to be applied to the columns of the  $U$  matrix as none of the columns can be zero, so the leading  $-1$  must appear in the  $U$  matrix. By applying these constraints, we make sure that  $\{-u_{i,r}\}$  is infeasible for any feasible  $\{u_{i,r}\}$ . Employing this sign symmetry breaking constraint to sol2 from Figure 2, we arrive at sol3 shown in Figure 3.

<sup>5</sup> <https://www.ibm.com/docs/en/icos/22.1.0?topic=variables-lexicographic-constraint>

$$\text{sol3: } U = \begin{pmatrix} -1 & 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 & -1 & 0 \end{pmatrix} \quad V = \begin{pmatrix} 1 & 1 & -1 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & -1 & -1 \end{pmatrix} \quad W = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

■ **Figure 3** sol3 is the solution derived from enforcing the sign symmetry constraints on sol2.

Similarly, the same type of sign symmetry-breaking constraints can be applied to the  $W$  factor matrix as follows:

$$w_{1,r} \leq 0$$

$$w_{k,r} \leq \sum_{k'=1}^{k-1} |w_{k',r}| \quad \forall r \in \mathcal{R}, k > 1, k \in \mathcal{W}.$$

The interpretation is as follows. In Figure 1c, consider  $c_4 = m_1 - m_2 + m_3 + m_6$ , and notice that one can redefine  $m_6 = (a_3 - a_1)(b_1 + b_2)$  to become  $m_6 = (a_3 - a_1)(-b_1 - b_2)$  and then rewrite  $c_4$  as  $c_4 = m_1 - m_2 + m_3 - m_6$ . Recall that the coefficients of the  $m$  terms in an output entry  $c_k$  are the entries of row  $k$  of factor matrix  $W$ . The transformation we just performed produces two equivalent solutions and is an instance of “value symmetry” that is broken by the above constraint set as it forces the first non-zero entry of a column of  $W$  to be  $-1$ .

## 4.2 Valid Inequalities

Based on the structure of this problem, we can also introduce a series of valid inequalities that could potentially help a CP solver with propagation.

First, for the  $W$  matrix, we know that each multiplicative term  $m_r$  must be used at least once for sufficiently small  $R$  (i.e., for non-trivial cases of the FMM problem where  $R \leq NMP$ ). This means that the sum of each column in  $W$  must be at least one:

$$\sum_{k \in \mathcal{W}} |w_{k,r}| \geq 1, \quad \forall r \in \mathcal{R}.$$

Each result term  $c_l$  must use at least  $M$  terms. This is due to a basic fact in algebraic complexity theory which states that the dot-product of two vectors of size  $M$  requires at least  $M$  multiplications [17]. This means that the sum of each row of  $W$  must be greater or equal to  $M$ :

$$\sum_{r \in \mathcal{R}} |w_{k,r}| \geq M, \quad \forall k \in \mathcal{W}.$$

Each result term  $c_l$  must differ in at least two  $m_r$  terms; a simple proof by contradiction is omitted for brevity. This can be modelled as follows:

$$\sum_{r \in \mathcal{R}} |w_{k,r} - w_{k',r}| \geq 2, \quad \forall k \neq k' \in \mathcal{W}.$$

Each term in the  $A$  and  $B$  matrices must appear in at least one of the multiplicative terms  $m_r$ . This translates to each row of  $U$  and  $V$  having at least one non-zero term as shown in the constraints below:

$$\sum_{r \in \mathcal{R}} |u_{i,r}| \geq 1, \quad \forall i \in \mathcal{U}$$

$$\sum_{r \in \mathcal{R}} |v_{j,r}| \geq 1, \quad \forall j \in \mathcal{V}.$$

Furthermore, each valid product of two terms from the  $A$  and  $B$  matrices, e.g.,  $a_2b_3$  for  $2 \times 2$  matrices, must appear in at least one of the  $R$  multiplication terms. For  $a_2b_3$  appears in  $c_1$  and  $c_2$ , see Figure 1a. This can be modelled as follows:

$$\sum_{r \in \mathcal{R}} |u_{i,r} \cdot v_{j,r}| \geq 1, \quad \forall \text{ valid } i, j.$$

### 4.3 Full CP Model

Finally, the full CP model is presented in Figure 4. The constraints in Equation (2) ensure that the output matches the original multiplication tensor and thus the validity of an assignment as a matrix multiplication algorithm. We enforce permutation symmetry-breaking with Equation (3) and sign symmetry-breaking with Equations (4)–(7). The valid inequalities are modelled through Equations (8)–(13).

$$\sum_{r \in \mathcal{R}} (u_{i,r} \cdot v_{j,r} \cdot w_{k,r}) = T_{i,j,k}, \quad \forall i \in \mathcal{U}, j \in \mathcal{V}, k \in \mathcal{W} \quad (2)$$

$$\text{lexicographic-strict}([u_{:,r}; v_{:,r}], [u_{:,r+1}; v_{:,r+1}]), \quad \forall r \in \mathcal{R} \quad (3)$$

$$u_{1,r} \leq 0 \quad \forall r \in \mathcal{R} \quad (4)$$

$$u_{i,r} \leq \sum_{i'=1}^{i-1} |u_{i',r}|, \quad \forall r \in \mathcal{R}, i > 1, i \in \mathcal{U} \quad (5)$$

$$w_{1,r} \leq 0 \quad \forall r \in \mathcal{R} \quad (6)$$

$$w_{k,r} \leq \sum_{k'=1}^{k-1} |w_{k',r}| \quad \forall r \in \mathcal{R}, k > 1, k \in \mathcal{W} \quad (7)$$

$$\sum_{k \in \mathcal{W}} |w_{k,r}| \geq 1, \quad \forall r \in \mathcal{R} \quad (8)$$

$$\sum_{r \in \mathcal{R}} |w_{k,r}| \geq M, \quad \forall k \in \mathcal{W} \quad (9)$$

$$\sum_{r \in \mathcal{R}} |w_{k,r} - w_{k',r}| \geq 2, \quad \forall k \neq k' \in \mathcal{W} \quad (10)$$

$$\sum_{r \in \mathcal{R}} |u_{i,r}| \geq 1, \quad \forall i \in \mathcal{U} \quad (11)$$

$$\sum_{r \in \mathcal{R}} |v_{j,r}| \geq 1, \quad \forall j \in \mathcal{V} \quad (12)$$

$$\sum_{r \in \mathcal{R}} |u_{i,r} \cdot v_{j,r}| \geq 1, \quad \forall \text{ valid } i, j \quad (13)$$

■ **Figure 4** Full CP Model with symmetry-breaking constraints and valid inequalities.

### 4.4 Sparsity-based Problem Decomposition

Given that the factor matrices that have been found for known decompositions tend to be sparse, we introduce some inexact inequalities to induce sparsity and trim candidate assignments that have a high likelihood to be infeasible or that are unnecessarily dense. For example, observe that Strassen's solution in Figure 1c leads to many zeros in the factor



matrices; no  $m$  term uses more than 2 out of 4 of the  $a$  or  $b$  terms, no  $c$  term uses more than 4 out of the 7  $m$  terms. It has been observed that as the matrix sizes grow, the best solutions become even sparser.

We first introduce a constraint limiting the number of active (i.e., nonzero) terms in each column  $r$  (i.e., multiplication term) of  $U$  and  $V$ . This constraint is written as:

$$\sum_{i \in \mathcal{U}} |u_{i,r}| + \sum_{j \in \mathcal{V}} |v_{j,r}| \leq K_1, \quad \forall r \in \mathcal{R}.$$

A similar constraint can be imposed on  $W$ , by restricting that each output must use at most  $K_2$  multiplication terms. This constraint is written as:

$$\sum_{r \in \mathcal{R}} |w_{k,r}| \leq K_2, \quad \forall k \in \mathcal{W}.$$

Based on these constraints,  $K_1$  has an upper bound of  $(NM + MP)$  and  $K_2$  is upper bounded by  $R$ . By observing decompositions for small to medium-scale matrices, we can estimate  $K_1$  and  $K_2$ . For example, for  $3 \times 3$  matrices with  $R = 23$ , we observe that  $K_1 = 9$  and  $K_2 = 10$  is the safest estimate possible compared to the upper bounds of 18 and 23, respectively, which could restrict the CP search dramatically. Note that one could start with any such estimates of the decomposition parameters  $K_1$  and  $K_2$ , iteratively increasing them if the restricted instances are found to be infeasible by the CP solver, eventually resulting in a complete resolution of the original problem.

## 4.5 Cyclic Invariant Formulation

In contrast to the symmetries of the factor matrices discussed in Section 4.1, there exists well-known cyclic symmetry for the multiplication tensors  $T_N$  of square matrices. More precisely, it is known that  $T_{i,j,k} = T_{j,k,i} = T_{k,i,j}$ . The authors in [1] proposed to leverage this cyclic symmetry property and parameterize FMM algorithms with cyclic invariant factor matrices:  $U = [ABCD]$ ,  $V = [ADBC]$ ,  $W = [ACDB]$  with  $A \in \{-1, 0, 1\}^{N^2 \times S}$  and  $B, C, D \in \{-1, 0, 1\}^{N^2 \times T}$  corresponding to a rank  $R = S + 3T$ .

Although this parametrization reduces the number of integer variables by a factor of three, helping with the combinatorial nature of the problem, there is no guarantee that the minimal rank decomposition corresponds to solutions that exhibit cyclic symmetry. That being said, Strassen's solution of  $R = 7$  for  $N = 2$ , which is optimal, exhibits such a symmetry ( $S \in \{1, 4\}$ ), as does the best-known rank of 23 for  $N = 3$  ( $S \in \{2, 5, 11\}$ ). Performing two steps of Strassen's algorithm for  $N = 2$  yields a rank 49 cyclic invariant solution for  $N = 4$ . It is currently unknown whether a solution of rank less than 49 exists for  $N = 4$ , let alone one exhibiting cyclic invariance.

We implement Ballard and Benson's cyclic invariant reduction [1] of the FMM problem for square matrices by reducing the decision variables of our CP formulation as required and imposing the invariant structure on the factor matrices.

## 5 Experiments

In this section, we present our experimental results starting by showing how our CP approach can recover the best-known upper bounds on the rank in a small amount of time on multiplication problems ranging from the trivial  $(N, M, P) = (1, 1, 1)$  case all the way up to the much harder  $(2, 2, 4)$  and  $(3, 3, 3)$  cases. We then present results for the infeasible cases

for (2, 2, 2). We used IBM’s CP Optimizer (CPO) 22.1.0<sup>6</sup> to solve our CP models. We ran our experiments on a compute cluster of AMD Ryzen Threadripper 2990WX cores with 128 GB of RAM per node.

## 5.1 Experimental Setup

To ensure the reproducibility and robustness of our results, all our experiments are run with multiple random seeds. This accounts for the often observed performance variability in combinatorial search; this is documented for example in MILP [12]. To that end, we ran each experiment with 10 different seeds. We assigned 8 cores (CPO’s `Workers` parameter) to the solver for each run (except for more compute-intensive experiments in Section 5.4 where we assigned 20 cores) and timed out the experiments after 2 hours.

## 5.2 Evaluation Metrics

We will report the solver runtime and the number of branches during the solution process for completed runs (i.e., runs that returned a feasible solution or a proof of infeasibility). Given that each problem is attempted with multiple random seeds, the shifted geometric mean with a shift of 0.0001<sup>7</sup>, median, minimum, and maximum of the time in seconds and the number of branches will be reported for a complete picture of the results. Runs that terminated due to the time or memory limits will be discussed where applicable.

■ **Table 1** Runtime results for the base CP model on various matrix dimensions. “geo mean” refers to the shifted geometric mean as described in Section 5.2; “med” refers to the median and “min”/“max” to the minimum and maximum, respectively.

$N$	$M$	$P$	$R$	Time (sec)	Num Branches
				geo mean (min, med, max)	geo mean (min, med, max)
1	1	1	1	0.00 (0.00, 0.00, 0.01)	$5.05 \times 10^1$ ( $4.50 \times 10^1$ , $5.00 \times 10^1$ , $5.80 \times 10^1$ )
1	1	2	2	0.00 (0.00, 0.00, 0.01)	$1.31 \times 10^2$ ( $1.00 \times 10^2$ , $1.27 \times 10^2$ , $2.13 \times 10^2$ )
1	2	1	2	0.00 (0.00, 0.01, 0.01)	$1.68 \times 10^2$ ( $1.08 \times 10^2$ , $1.69 \times 10^2$ , $2.31 \times 10^2$ )
1	1	3	3	0.00 (0.00, 0.01, 0.01)	$7.09 \times 10^2$ ( $3.47 \times 10^2$ , $7.78 \times 10^2$ , $1.38 \times 10^3$ )
1	3	1	3	0.00 (0.00, 0.01, 0.01)	$9.44 \times 10^2$ ( $3.68 \times 10^2$ , $9.83 \times 10^2$ , $1.82 \times 10^3$ )
1	2	2	4	0.01 (0.00, 0.01, 0.01)	$4.86 \times 10^3$ ( $1.68 \times 10^3$ , $5.27 \times 10^3$ , $7.68 \times 10^3$ )
2	1	2	4	0.01 (0.01, 0.01, 0.02)	$4.36 \times 10^3$ ( $2.63 \times 10^3$ , $4.39 \times 10^3$ , $7.14 \times 10^3$ )
1	2	3	6	0.05 (0.02, 0.04, 0.10)	$3.51 \times 10^4$ ( $1.41 \times 10^4$ , $2.96 \times 10^4$ , $9.83 \times 10^4$ )
1	3	2	6	0.05 (0.03, 0.06, 0.12)	$3.19 \times 10^4$ ( $1.40 \times 10^4$ , $3.19 \times 10^4$ , $7.18 \times 10^4$ )
2	1	3	6	0.05 (0.02, 0.04, 0.11)	$3.79 \times 10^4$ ( $1.45 \times 10^4$ , $3.33 \times 10^4$ , $8.93 \times 10^4$ )
2	2	2	7	0.74 (0.28, 0.75, 1.84)	$6.41 \times 10^5$ ( $2.03 \times 10^5$ , $6.69 \times 10^5$ , $1.70 \times 10^6$ )
1	3	3	9	0.37 (0.26, 0.36, 0.60)	$2.92 \times 10^5$ ( $1.84 \times 10^5$ , $3.09 \times 10^5$ , $4.16 \times 10^5$ )
3	1	3	9	0.42 (0.18, 0.52, 0.61)	$3.10 \times 10^5$ ( $1.64 \times 10^5$ , $3.35 \times 10^5$ , $4.36 \times 10^5$ )
2	2	3	11	49.64 (0.98, 71.82, 245.06)	$3.40 \times 10^7$ ( $6.90 \times 10^5$ , $4.71 \times 10^7$ , $1.74 \times 10^8$ )
2	3	2	11	26.47 (6.68, 29.41, 133.29)	$1.56 \times 10^7$ ( $3.52 \times 10^6$ , $1.39 \times 10^7$ , $9.00 \times 10^7$ )

<sup>6</sup> <https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-cp-optimizer>

<sup>7</sup> The shifted geometric mean of a set of  $n$  values  $t_1, \dots, t_n$  is defined as  $(\prod_{i=1}^n [t_i + \text{shift}])^{\frac{1}{n}} - \text{shift}$ . Compared to the arithmetic mean, it is less sensitive to large variations in the values.

### 5.3 Feasible Cases: Searching for Solutions with the Base CP Model

Table 1 shows the time and number of branches (Num Branches) required by the base CP model (i.e., without symmetry breaking or valid inequalities) to find solution for a range of problems. Our approach was able to find Strassen’s solution for the  $2 \times 2$  matrix multiplication in less than a second whereas the AlphaTensor paper [7] reports a few minutes of model inference to find that solution.

**Performance variability.** In Table 1, we can see that for  $(2, 2, 3)$  with  $R = 11$ , the worst seed took 245 seconds to find a feasible solution compared to 0.98 seconds for the best seed. This drastic difference in time (and ultimately the number of branches) is an indication that minute parameters such as the seed can significantly impact the CP search. For feasible instances, this phenomenon can be seen as a blessing rather than a curse if one has access to multiple cores: the randomness can be exploited by running multiple copies of the solver, terminating as soon as the first successful run is completed. This has been done in MILP [8].

### 5.4 Feasible Cases: Sparsity Constraints and Cyclic Invariance Help

Solving the base CP formulation, with our current time and memory budgets, does not yet yield feasible decompositions for dimensions higher than  $(2, 2, 3)$  or  $(2, 3, 2)$ . However, after increasing both the time and memory limits, we were able to find a solution to the problem for dimension  $(2, 2, 4)$  with  $R = 14$  in 19.6 hours using the inexact inequalities ( $K_1 = 11$  and  $K_2 = 7$ ) developed in Section 4.4. Furthermore, our cyclic invariant formulation (with  $S = 5$ ) with inexact inequalities ( $K_1 = 9$  and  $K_2 = 10$ ) was able to find a solution for  $(3, 3, 3)$ ,  $R = 23$ . More specifically, we ran the cyclic invariant formulation for 10 hours with 5 different seeds and observed that two seeds produced a feasible solution within one hour whereas the other three seeds hit the time limit. Once again, this indicates that performance variability in the CP search is significant for our problem. Additionally, we ran the base CP formulation for  $(3, 3, 3)$  without inexact inequalities for 5 seeds which all hit the time limit of 10 hours, demonstrating the benefit of the reduction of variables for the cyclic invariant formulation and sparsity constraints. We have yet to check whether using only inexact inequalities can help the base formulation for  $(3, 3, 3)$ . A similar result was observed for the  $(2, 2, 2)$  case in which the cyclic invariant formulation ( $S = 4$ ) with inexact inequalities ( $K_1 = 6$  and  $K_2 = 4$ ) produced an average solution time of 0.05 seconds across 10 seeds whereas the base CP model has an average of 1.46 seconds. Our current implementation is not able to find cyclic invariant solutions for  $N = 4$  with  $R = 48$ , but we have hope that this approach is a promising tool for the search for new cyclic invariant solutions for square matrix multiplication.

### 5.5 Infeasible Cases: The Importance of Symmetry Breaking

Since CP performs an exhaustive search, it can provide a proof of infeasibility if a given rank  $R$  is not achievable for certain matrix dimensions. As expected, the runtime to prove infeasibility significantly increases as we approach the known minimum rank; this can be seen in Table 2 for the  $(2, 2, 2)$  case. It is also apparent that the addition of symmetry-breaking constraints helps tremendously when proving infeasibility given that they reduce the search space significantly. More specifically, for  $R = 6$  in Table 2, it is not even currently possible to prove infeasibility without symmetry-breaking constraints in 2 hours whereas the CP model with symmetry-breaking constraints (B+S) requires around 7 minutes. These results highlight the importance of symmetry-breaking constraints when looking to prove infeasibility.

■ **Table 2** Runtime results for the base CP model and variants to prove infeasibility of  $R < 7$  for (2,2,2). “geo mean” refers to the shifted geometric mean as described in Section 5.2; “med” refers to the median and “min”/“max” to the minimum and maximum, respectively. Overall, the use of symmetry-breaking constraints (denoted by the letter “S”) on top of the base CP formulation (“B”) is crucial for efficient proofs of infeasibility. “V” refers to the valid inequalities of Section 4.2 which sometimes complement symmetry-breaking but are not always needed for the fastest results.

$R$	Method	Time (sec)	Num Branches
		geo mean (min, med, max)	geo mean (min, med, max)
1	B	0.01 (0.00, 0.01, 0.02)	$1.08 \times 10^3$ ( $1.06 \times 10^3$ , $1.08 \times 10^3$ , $1.10 \times 10^3$ )
	B+S	0.00 (0.00, 0.01, 0.01)	$1.00 \times 10^{-4}$ (0.00, 0.00, 0.00)
	B+V	0.00 (0.00, 0.00, 0.01)	$1.00 \times 10^{-4}$ (0.00, 0.00, 0.00)
	<b>B+V+S</b>	<b>0.00</b> (0.00, 0.00, 0.00)	$1.00 \times 10^{-4}$ (0.00, 0.00, 0.00)
2	B	0.01 (0.00, 0.01, 0.03)	$4.38 \times 10^3$ ( $3.72 \times 10^3$ , $4.41 \times 10^3$ , $5.30 \times 10^3$ )
	B+S	0.01 (0.01, 0.01, 0.02)	$1.08 \times 10^3$ ( $1.08 \times 10^3$ , $1.08 \times 10^3$ , $1.08 \times 10^3$ )
	<b>B+V</b>	<b>0.00</b> (0.00, 0.01, 0.02)	$1.33 \times 10^3$ ( $1.31 \times 10^3$ , $1.32 \times 10^3$ , $1.34 \times 10^3$ )
	B+V+S	0.01 (0.01, 0.01, 0.03)	$1.09 \times 10^3$ ( $1.07 \times 10^3$ , $1.08 \times 10^3$ , $1.10 \times 10^3$ )
3	B	0.21 (0.17, 0.21, 0.28)	$1.70 \times 10^5$ ( $1.42 \times 10^5$ , $1.70 \times 10^5$ , $1.95 \times 10^5$ )
	B+S	0.02 (0.01, 0.02, 0.03)	$4.13 \times 10^3$ ( $3.06 \times 10^3$ , $4.30 \times 10^3$ , $5.32 \times 10^3$ )
	B+V	0.20 (0.13, 0.21, 0.25)	$1.38 \times 10^5$ ( $1.14 \times 10^5$ , $1.35 \times 10^5$ , $1.78 \times 10^5$ )
	<b>B+V+S</b>	<b>0.02</b> (0.01, 0.02, 0.03)	$3.61 \times 10^3$ ( $2.52 \times 10^3$ , $3.67 \times 10^3$ , $4.92 \times 10^3$ )
4	B	43.79 (31.33, 41.89, 66.93)	$3.85 \times 10^7$ ( $3.06 \times 10^7$ , $3.80 \times 10^7$ , $5.00 \times 10^7$ )
	<b>B+S</b>	<b>0.12</b> (0.07, 0.13, 0.18)	$8.50 \times 10^4$ ( $6.94 \times 10^4$ , $8.54 \times 10^4$ , $1.03 \times 10^5$ )
	B+V	53.49 (39.10, 48.87, 69.35)	$3.70 \times 10^7$ ( $3.18 \times 10^7$ , $3.69 \times 10^7$ , $4.23 \times 10^7$ )
	B+V+S	0.15 (0.11, 0.15, 0.20)	$8.53 \times 10^4$ ( $7.34 \times 10^4$ , $8.10 \times 10^4$ , $1.06 \times 10^5$ )
5	B	T.O. (N/A, N/A, N/A)	$6.03 \times 10^9$ ( $5.56 \times 10^9$ , $5.75 \times 10^9$ , $7.14 \times 10^9$ )
	B+S	3.06 (2.28, 3.02, 4.15)	$2.22 \times 10^6$ ( $1.89 \times 10^6$ , $2.19 \times 10^6$ , $2.67 \times 10^6$ )
	B+V	T.O. (N/A, N/A, N/A)	$5.57 \times 10^9$ ( $3.97 \times 10^9$ , $5.83 \times 10^9$ , $6.16 \times 10^9$ )
	<b>B+V+S</b>	<b>2.98</b> (2.56, 2.94, 3.44)	$2.14 \times 10^6$ ( $1.91 \times 10^6$ , $2.12 \times 10^6$ , $2.56 \times 10^6$ )
6	B	T.O. (N/A, N/A, N/A)	$5.99 \times 10^9$ ( $4.53 \times 10^9$ , $5.79 \times 10^9$ , $6.93 \times 10^9$ )
	<b>B+S</b>	<b>429.26</b> (333.88, 441.63, 528.61)	$3.28 \times 10^8$ ( $2.94 \times 10^8$ , $3.31 \times 10^8$ , $3.76 \times 10^8$ )
	B+V	T.O. (N/A, N/A, N/A)	$4.67 \times 10^9$ ( $3.82 \times 10^9$ , $4.73 \times 10^9$ , $5.48 \times 10^9$ )
	B+V+S	517.33 (414.07, 522.81, 640.65)	$3.35 \times 10^8$ ( $2.97 \times 10^8$ , $3.28 \times 10^8$ , $3.95 \times 10^8$ )

## 6 Conclusion

We have proposed a novel CP approach to solve the fast matrix multiplication problem. We have provided a set of constraints for breaking permutation and sign symmetries as well as a set of valid inequality constraints to help CP prune and propagate more efficiently. We provide a decomposition framework that is beneficial for finding feasible solutions for the largest case we have attempted, i.e.,  $3 \times 3$  matrix multiplication. Based on our experimental results, we have been able to solve small instances of this problem within a reasonable amount of time. This is in contrast to some existing search-based approaches (MILP, planning) that seem to struggle. In contrast to the AlphaTensor approach [7], the CP model is far more natural for this combinatorial task and is uniquely positioned to provide proof of infeasibility for some open problems in this space.

While the results of our approach are promising given the limited amount of computing used, there are several limitations that we aim to address in future work. First, our algorithm struggles to scale for larger matrix dimensions or ranks due to the quick increase in the number of variables of the CP model. Secondly, we have found that the base CP model outperforms the addition of symmetry constraints and valid inequalities in the case of feasible

solutions, likely due to the latter’s tendency to prune symmetric solutions early in the tree search. However, we believe that our experiment’s small matrix dimensions may have skewed these results and valid inequalities may be crucial for larger sizes. Moving forward, we propose several areas for further exploration and improvement:

- Conduct larger-scale experiments using larger compute clusters to take advantage of the parallelizability of the CP solver’s search procedure.
- Analyze the highly structured nature of this problem to develop more valid inequalities that can further reduce the search space of our CP model, including inexact inequalities that may not hold for all matrix multiplication dimensions but help for some cases.
- Explore solver parameter tuning, particularly for branching strategies and other important search-related decisions.
- Further investigate the idea of sparsity-based problem decomposition as a means of improving the scalability and performance of our approach.

---

## References

- 1 Austin R. Benson and Grey Ballard. A framework for practical parallel fast matrix multiplication. *ACM SIGPLAN Notices*, 50(8):42–53, January 2015. doi:10.1145/2858788.2688513.
- 2 Markus Bläser. On the complexity of the multiplication of matrices of small formats. *Journal of Complexity*, 19(1):43–60, 2003.
- 3 Markus Bläser. *Fast Matrix Multiplication*. Number 5 in Graduate Surveys. Theory of Computing Library, 2013. doi:10.4086/toc.gs.2013.005.
- 4 Roger W Brockett and David Dobkin. On the optimal evaluation of a set of bilinear forms. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 88–95, 1973.
- 5 Nicolas T Courtois, Gregory V Bard, and Daniel Hulme. A new general-purpose method to multiply 3x3 matrices using only 23 multiplications, 2011.
- 6 Hans F de Groote. On varieties of optimal algorithms for the computation of bilinear mappings ii. optimal algorithms for  $2 \times 2$ -matrix multiplication. *Theoretical Computer Science*, 7(2):127–148, 1978.
- 7 Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatin, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, and Grzegorz Swirszcz. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.
- 8 Matteo Fischetti and Michele Monaci. Exploiting erraticism in search. *Operations Research*, 62(1):114–122, 2014.
- 9 Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.
- 10 Marijn JH Heule, Manuel Kauers, and Martina Seidl. Local search for fast matrix multiplication. In *Theory and Applications of Satisfiability Testing–SAT 2019: 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings 22*, pages 155–163. Springer, 2019.
- 11 Marijn JH Heule, Manuel Kauers, and Martina and Seidl. New ways to multiply  $3 \times 3$ -matrices. *Journal of Symbolic Computation*, 104:899–916, 2021.
- 12 Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. In *Theory driven by influential applications*, pages 1–12. INFORMS, 2013.
- 13 Alexey V. Smirnov. The bilinear complexity and practical algorithms for matrix multiplication. *Computational Mathematics and Mathematical Physics*, 53:1781–1795, 2013.
- 14 Laurent Sorber and Marc Van Barel. A mixed-integer linear program formulation for fast matrix multiplication, 2017.

- 15 David Speck, Paul Höft, Daniel Gnad, and Jendrik Seipp. Finding matrix multiplication algorithms with classical planning. In Sven Koenig, Roni Stern, and Mauro Vallati, editors, *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. AAAI Press, 2023.
- 16 Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.
- 17 Shmuel Winograd. On the number of multiplications necessary to compute certain functions. *Communications on Pure and Applied Mathematics*, 23(2):165–179, 1970.



# Probabilistic Inference by Projected Weighted Model Counting on Horn Clauses

Alexandre Dubray ✉ 

Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), UCLouvain, Belgium

Pierre Schaus ✉ 

Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), UCLouvain, Belgium

Siegfried Nijssen ✉ 

Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), UCLouvain, Belgium

---

## Abstract

Weighted model counting, that is, counting the weighted number of satisfying assignments of a propositional formula, is an important tool in probabilistic reasoning. Recently, the use of projected weighted model counting (PWMC) has been proposed as an approach to formulate and answer probabilistic queries. In this work, we propose a new simplified modeling language based on PWMC in which probabilistic inference tasks are modeled using a conjunction of Horn clauses and a particular weighting scheme for the variables. We show that the major problems of inference for Bayesian Networks, network reachability and probabilistic logic programming can be modeled in this language. Subsequently, we propose a new, relatively simple solver that is specifically optimized to solve the PWMC problem for such formulas. Our experiments show that our new solver is competitive with state-of-the-art solvers on the major problems studied.

**2012 ACM Subject Classification** Mathematics of computing → Probabilistic inference problems; Computing methodologies → Probabilistic reasoning; Mathematics of computing → Bayesian networks

**Keywords and phrases** Model Counting, Bayesian Networks, Probabilistic Networks

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.15

## Supplementary Material

*Software (Source Code)*: <https://github.com/aia-uclouvain/schlandals>  
archived at `swh:1:dir:38ff0f7ce1c12353938ed4056877ef497389440c`

## 1 Introduction

Weighted model counters are systems that calculate the weighted number of assignments that satisfy a formula in conjunctive normal form (CNF). Weighted model counting (WMC) is a hard problem: it is #P complete, and hence building efficient systems for this task is a challenge.

An important application of WMC is probabilistic inference, that is, calculating the probability of an observation according to a given probabilistic model. With increasing sizes of probabilistic models, the performance of inference remains important. Model counters have already been used to solve probabilistic inference problems on

- Bayesian networks (BNs) [6, 9, 20];
- Probabilistic networks (PNs) [12, 27]
- Probabilistic logic programs (PLPs) [13].

In each of these studies, approaches were proposed for how to model a probabilistic inference task as a WMC task on a CNF formula. Unfortunately, the resulting models are not always



© Alexandre Dubray, Pierre Schaus, and Siegfried Nijssen;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 15; pp. 15:1–15:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



simple. For Bayesian networks the CNF formulas are polynomial in size given the size of the BN, but a number of papers have proposed increasingly complex models to make solving efficient [2, 4, 5]; for PNs and PLPs the CNF can even be exponentially larger than the original probabilistic model [27]. A challenge remains how to model inference tasks in a simple manner and obtain good performance at the same time.

A promising solution to this modeling problem may be the use of *projected (weighted) model counting* (PWCM) [1] as a general approach for probabilistic inference; it was shown that using PWCM a CNF of polynomial size can be used to solve reachability problems on PNs [12]. Where in weighted model counting a sum is calculated over all models of a formula  $F$ , in PWCM the models are *projected* on a subset of the variables (the priority variables) and a weight is given only to each resulting projected model. For example, if  $F = (a \vee b) \wedge (b \vee c \vee d)$ , then a model counter will sum over the 11 models of  $F$ . On the other hand, if the priority variables are  $\{a, c\}$ , then a projected model counter sums over 4 projected models (as setting  $b = \top$  always makes  $F$  true).

In this work, we study the challenge of efficient and simple probabilistic inference using PWCM in more detail. We show that the aforementioned probabilistic inference tasks can be modeled as PWCM tasks over a simpler form of CNF formulas: CNF formulas of *Horn* clauses with a specific weighting scheme. We will call this task the task of *projected probabilistic Horn model counting* (PPHMC). We will argue that the resulting weighted CNF models are simple and polynomial in size given the original probabilistic models.

Subsequently, we will introduce a new solver, the Schlandals solver, which takes full advantage of the probabilistic weighting scheme and the fact that all clauses in our formulas are Horn clauses. The main intuition behind our solver is that it exploits the well-known fact that the SAT problem over conjunctions of Horn clauses can be solved in linear time. Using this observation, it is able to efficiently find assignments to the non-priority variables that greatly reduce the number of constrained clauses in the input formula (like  $b = \top$  in the example above). At the same time, it is still able to exploit optimizations found in other DPLL-style model counters, such as component caching, unit propagation and branching strategies, to run with a bounded use of memory.

An experimental evaluation of our solver shows that it is competitive with state-of-the-art existing tools. We compare our solver with the best performing solvers of the 2022 projected (weighted) model counting competition on two probabilistic inference tasks: inference in Bayesian Networks and reliability estimation in PNs. Our experiments show that our solver is able to solve most of the instances for the BN task, beating other tools when using simple forms of CNF models, and getting similar performance when compared to optimized complex models. On PNs, our solver even outperforms the state-of-the-art, without any optimization of our model. These results open up future possibilities for the use of PPHMC.

## 2 Related Work

This work is concerned with the task of *Projected Weighted Model Counting* (PWCM). In the classical *Model Counting* problem, one is interested in finding the number of models of a Boolean formula  $F$ , the assignments that makes the formula true. In this work we focus on the most common setting, in which  $F$  is a formula in conjunctive normal form (CNF). In *Projected Model Counting*, the goal is to count the assignments to a subset of the variables (the *priority variables*) such that there exists an assignment to the other variables that makes  $F$  true. If the set of priority variables contains all variables of  $F$ , the projected and unprojected problems are the same. In the rest of this work, we thus assume that the subset of priority variable is not empty. In the weighted version of these problems, each valid assignment is weighted and the goal is to return the sum of the weights of the models.

More formally, let  $F$  be a Boolean formula over a set of variables  $\mathcal{V}$  with  $\mathcal{P} \subseteq \mathcal{V}$  the priority variables and  $\mathcal{D} = \mathcal{V} \setminus \mathcal{P}$  the non-priority variables. In traditional projected weighted model counting each priority variable is given two weights,  $w(v)$  and  $w(\neg v)$ , one for each polarity. We denote by  $S_X$  an assignment to the variables in  $X \subseteq \mathcal{V}$  and  $S_X[x]$  the assigned value of  $x \in X$ . If  $X, Y \subset \mathcal{V}$  are two disjoint sets of variables, then  $S_X \cup S_Y$  is defined as the conjunction of the assignments. We define

$$\mathcal{S}_F = \{S_{\mathcal{P}} \mid \exists S_{\mathcal{D}} : F[S_{\mathcal{P}} \cup S_{\mathcal{D}}] = \top\}$$

as the set of assignments to the priority variables that can be extended with an assignment to the non-priority variables and satisfy the formula. The goal of the traditional PWMC task is then to find

$$\sum_{S \in \mathcal{S}_F} \left( \prod_{v \in \mathcal{V} \mid S[v] = \top} w(v) \times \prod_{v \in \mathcal{V} \mid S[v] = \perp} w(\neg v) \right).$$

Note that when weighted model counters are used to model probabilistic inference tasks, the weights of assignments are set such that this sum corresponds to a probability. If for every variable  $p \in \mathcal{P}$  it holds that  $w(p) + w(\neg p) = 1$  the resulting sum will always be  $\leq 1$ . Unfortunately, the models for some problems require that  $w(p) + w(\neg p) > 1$ , and hence solvers cannot assume that  $w(p) + w(\neg p) = 1$ .

Various solvers exist to solve the PWMC problem and its unweighted version. In [22], the authors present **Ganak** is a model counter build upon **sharpSAT**, a DPLL-style model counter with component caching [26]. **Ganak** uses probabilistic component caching while ensuring guarantees on the validity of the returned count. Furthermore, the authors propose to use information about the cache in the branching heuristics and show that this is beneficial to model counters. **Ganak** can also be used for projected model counting, by first branching on the priority variables, but it does not use other specialized techniques. Further modifications of **Ganak** have been proposed and in particular it has been shown that integrating tree decomposition in the branching heuristic can have a positive impact [15]. This has led to the development of **SharpSAT-TD** [15], a (weighted) model counter, and **GPMC** [25], which is also able to solve the PWMC problem.

The **projMC** solver [17] uses another approach to solve the PWMC problem<sup>1</sup>. To compute the count over a formula  $F$  they first compute a disjunctive decomposition from a model of  $F$ . They then use the pairwise incompatible parts of the decomposition to simplify  $F$ , and they recursively solve the new sub-problems.

The aforementioned solvers use a strategy in which a limited amount of memory is used. This is also the focus of our work. An alternative strategy is to use a model compilation in combination with dynamic programming. Recently, Dudek et al. proposed the two-phased **ProCount** solver [11]. In the first phase, the input formula is transformed into a graded project-join tree (by a *planner*). Then, in the second phase, an executor (based on algebraic decision diagrams) is used to compute the count, using a dynamic programming approach.

### 3 Problem Definition

First, let us note that, since this work focuses on probabilistic problems, the weights on the priority variables are used to model probabilities. Hence, we refer to those as *probabilistic variables* while the set of non-priority variables are called *deterministic variables*. In the rest

<sup>1</sup> Although the original paper only describes the unweighted problem, a parameter in the solver enables the retrieval of the count as a float, taking into account the weights

## 15:4 Probabilistic Inference by Projected Weighted Model Counting on Horn Clauses

of this paper, we denote with  $P(E)$  the probability that an event  $E$  occurs. In this work, we introduce a combination of two novelties in how to model problems using PWMCM. First, a constraint on the clauses in  $F$  is imposed: we only allow clauses to be *Horn* clauses.

► **Definition 1** (Horn clause). *A Horn clause  $C$  is a formula of the form*

$$v_1 \wedge \cdots \wedge v_n \Rightarrow v_t$$

where  $I = v_1 \wedge \cdots \wedge v_n$  is called the *implicant* of the clause and  $h = v_t$  is the *head* of the clause. Here  $v_i \in \mathcal{V}$  is a variable, and  $v_t$  is either a variable in  $\mathcal{V}$  or  $\perp$ . If  $n = 0$  (the implicant is empty) then the left-hand side reduces to  $\top$ .

It can be observed that when a Horn clause is written as an implication, as above, all the literals in the clause have the same (positive) polarity. Hence, in order to simplify our discussion and notation, we only talk about variables, and not literals. A Horn clause  $C_i$  can be identified uniquely by its implicant  $I_i$  and its head  $h_i$ . In the rest of this paper, we will use the notation  $C_i$  and  $(I_i, h_i)$  interchangeably. For simplicity of notation, we also denote by  $v \in I_i$  the fact that  $v \in \mathcal{V}$  is a variable of the implicant of  $C_i$ .

Horn clauses have been well studied in the literature. An important result is that the SAT problem over a CNF formula of Horn clauses can be solved in linear time [10]; given that the SAT problem in its general form is NP hard, this is a significant simplification.

Secondly, we add the notion of *distributions over the probabilistic variables*. We assume that each probabilistic variable  $p \in \mathcal{P}$  belongs to exactly one partition  $P_i \subseteq \mathcal{P}$  of the probabilistic variables. We define a *distribution* over each such partition, in the following simple manner: we require that one weight is specified for each probabilistic variable and require that  $\sum_{p \in P_i} w(p) = 1$  for all variables in the partition. Subsequently, we calculate the weight of an assignment  $S$  to the probabilistic variables as follows. The weight of a partition, given the assignment  $S$ , is defined as follows:

$$w_{P_i}(S) = \begin{cases} w(p) & \text{if there is exactly one } p \in P_i \text{ for which } S[p] = \top \\ 0 & \text{otherwise,} \end{cases}$$

or, in other words, if exactly one variable in the partition is set to  $\top$ , the weight of that variable is given to that partition; otherwise, the assignment is invalid. Implicitly, we allow only one variable within a partition to be true at the same time. The weight of the assignment  $S$  is the product of the partition weights given  $S$  and thus the solution of the PPHMC problem is given by

$$\sum_{S \in \mathcal{S}_F} \prod_i w_{P_i}(S),$$

where we assume there is at least one partition of probabilistic variables.

### 3.1 Models

As our earlier discussion makes clear, in this work we study a simpler modeling language in which non-Horn clauses are not allowed, and we combine this with a different approach to weighting. In this section, we will show that even though we apply the aforementioned restrictions, a number of different problems can still be modeled in our language.

► **Example 2** (Bayesian Networks). A *Bayesian Network* (BN) is a probabilistic model that can be represented by a directed acyclic graph. Random variables are represented by nodes and conditional dependency relationships by edges. Figure 1(a) shows an example of a BN with four nodes. For simplicity, we illustrate the encoding in our language for a network with nodes that have two values, but this generalizes for nodes with more than two values.

In this network, both  $B$  and  $C$  depend on  $A$ , while  $D$  depends on  $B$  and  $C$ . In addition to the network structure, *conditional probability tables* (CPTs) give, for each node, the probability of its values conditioned by its parent's value. These are called the *parameters* of the network and for conciseness we write  $P(x \mid \mathbf{u})$  for the parameter corresponding to the probability that a node  $X$  takes value  $x$  given that its parents take values  $\mathbf{u} = u_1, \dots, u_n$ .

Various CNF encodings have been proposed for Bayesian networks [2, 4, 5, 7]. Even though the author of these works use Horn clauses, the un-projected nature of their target solvers imposes additional, non-Horn, clauses.

Let us present our encoding for BN by first defining the logical variables. For every value  $x$  of node  $X$  in the network, we define one deterministic variable  $v^x$ . For the BN in Figure 1, we have the following variables:  $v^{a_0}, v^{a_1}, v^{b_0}, v^{b_1}, v^{c_0}, v^{c_1}, v^{d_0}, v^{d_1}$ . Moreover for each parameter  $P(x \mid \mathbf{u})$  we define a corresponding probabilistic variable  $p_{\mathbf{u}}^x$ . For the CPT of node  $B$ , we introduce four such variables:  $p_{a_0}^{b_0}, p_{a_0}^{b_1}, p_{a_1}^{b_0}, p_{a_1}^{b_1}$ .

In our approach, we have to define the distributions over the probabilistic variables. The CPTs of the network give a natural partition of the probabilistic variables. That is, for a node  $X$  in the network, we define one partition for each line of its CPT. Hence, for node  $B$  there are two partitions:  $P_{B_1} = \{p_{a_0}^{b_0}, p_{a_0}^{b_1}\}$  and  $P_{B_2} = \{p_{a_1}^{b_0}, p_{a_1}^{b_1}\}$ . Next, we define the weight on the variables. Contrary to previous encodings, only weights on probabilistic variables are needed, and we use as weight the parameter they represent:  $w(p_{\mathbf{u}}^x) = P(x \mid \mathbf{u})$ . For example, we have that  $w(p_{a_0}^{b_0}) = 0.6$  and  $w(p_{a_0}^{b_1}) = 0.4$ .

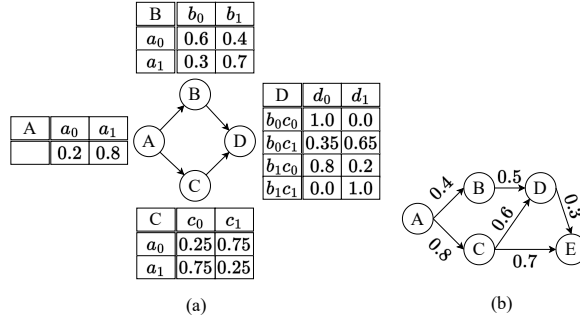
Finally, we define the clauses: for each parameter  $P(x \mid \mathbf{u})$  with  $\mathbf{u} = u_1, \dots, u_n$ , we introduce one clause  $v^{u_1} \wedge \dots \wedge v^{u_n} \wedge p_{\mathbf{u}}^x \Rightarrow v^x$ . The clauses for the BN in Figure 1 are shown below.

$$\begin{array}{lll}
 p^{a_0} \Rightarrow v^{a_0} & v^{a_0} \wedge p_{a_0}^{c_0} \Rightarrow v^{c_0} & v^{b_1} \wedge v^{c_0} \wedge p_{b_1 c_0}^{d_0} \Rightarrow v^{d_0} \\
 p^{a_1} \Rightarrow v^{a_1} & v^{a_0} \wedge p_{a_0}^{c_1} \Rightarrow v^{c_1} & v^{b_1} \wedge v^{c_0} \wedge p_{b_1 c_0}^{d_1} \Rightarrow v^{d_1} \\
 v^{a_0} \wedge p_{a_0}^{b_0} \Rightarrow v^{b_0} & v^{a_1} \wedge p_{a_1}^{c_0} \Rightarrow v^{c_0} & v^{b_0} \wedge v^{c_1} \wedge p_{b_0 c_1}^{d_0} \Rightarrow v^{d_0} \\
 v^{a_0} \wedge p_{a_0}^{b_1} \Rightarrow v^{b_1} & v^{a_1} \wedge p_{a_1}^{c_1} \Rightarrow v^{c_1} & v^{b_0} \wedge v^{c_1} \wedge p_{b_0 c_1}^{d_1} \Rightarrow v^{d_1} \\
 v^{a_1} \wedge p_{a_1}^{b_0} \Rightarrow v^{b_0} & v^{b_0} \wedge v^{c_0} \wedge p_{b_0 c_0}^{d_0} \Rightarrow v^{d_0} & v^{b_1} \wedge v^{c_1} \wedge p_{b_1 c_1}^{d_0} \Rightarrow v^{d_0} \\
 v^{a_1} \wedge p_{a_1}^{b_1} \Rightarrow v^{b_1} & v^{b_0} \wedge v^{c_0} \wedge p_{b_0 c_0}^{d_1} \Rightarrow v^{d_1} & v^{b_1} \wedge v^{c_1} \wedge p_{b_1 c_1}^{d_1} \Rightarrow v^{d_1}
 \end{array}$$

The clause  $v^{a_0} \wedge p_{a_0}^{b_0} \Rightarrow v^{b_0}$ , for instance, represents that if  $A$  has value  $a_0$  and we pick the variable  $p_{a_0}^{b_0}$  from the distribution  $\{p_{a_0}^{b_0}, p_{a_0}^{b_1}\}$ ,  $B$  will have value  $b_0$ ; we believe this is a natural and simple representation that directly reflects the BN.

Note that we can satisfy all clauses by setting the  $v$  variables to true. A common inference problem on Bayesian Networks is that of calculating a probability  $P(X = x)$ . We can solve this problem by adding a clause  $v^{x'} \Rightarrow \perp$  for each value  $x'$  of  $X$  such that  $x' \neq x$ . Effectively this removes from the sum those assignments in which  $X \neq x$ .

This encoding differs in a number of ways from the encodings used in WMC [2, 6]. The general idea is similar: for rows of the CPT, clauses are created; probabilistic variables receive weights that represent entries in the CPTs. Compared to earlier encodings, we do not generate clauses to impose that the indicators variables ( $v^x$ ) are mutually exclusive for a node  $X$ . Our weighting scheme takes care of this. Furthermore, the earlier encodings have



■ **Figure 1** a) An example of Bayesian Network with four binary variables. In this network  $B$  and  $C$  depends on  $A$  and  $D$  depends on  $B$  and  $C$ . The probability tables are given next to the nodes. b) An example of probabilistic network for reliability problems. The numbers labeled on the edges are their probability of being present.

the parameter variables as the head of the implications in the CNFs, while in our encoding they are in the implicants; while both representations are equivalent, we believe that in our representation the structure of the BN is more closely reflected in the clauses.

► **Example 3 (Reliability in Networks).** Reliability in network (RN) problems study the connectivity of nodes in probabilistic graphs. In such graphs, as shown in Figure 1(b), each edge has a probability of being present. In this work, we consider the computation of the probability that two nodes are connected.

More formally, let  $G = (V, E)$  be a probabilistic graph and  $f_w : E \mapsto [0, 1]$  a weighting function that assign to each edge a probability of being present or not. We denote  $s$  the source node,  $t$  the target node and  $R_t^s$  ( $\bar{R}_t^s$ ) the fact that  $t$  is (not) reachable from  $s$ . The goal is then to compute  $P(R_t^s)$ .

The encoding of this problem in our language is similar to that in [12], in which the authors propose to compute  $P(\bar{R}_t^s)$  and then use the fact that  $P(R_t^s) = 1 - P(\bar{R}_t^s)$  to answer the initial query.

Let us first define the logical variables. For each node  $X \in V$ , we introduce one deterministic variable  $v_X$ . For each edge  $e \in E$  from  $u$  to  $v$ , we introduce two probabilistic variables  $p_{uv}$  (the edge is present) and  $\bar{p}_{uv}$  (the edge is not present). The weighting scheme of the probabilistic variables uses the weighting function of the edges:  $w(p_{uv}) = f_w(e)$  and  $w(\bar{p}_{uv}) = 1 - f_w(e)$ . A distribution is defined for each edge, containing these two variables:  $P_e = \{p_{uv}, \bar{p}_{uv}\}$ . Since the probabilistic variables for each edge are in their own distribution, and each distribution contains no other variables, it is easy to see that an assignment to the probabilistic variables corresponds to a possible instance of the graph.

We will use the deterministic variables  $v_X$  to represent whether in the instance implied by the probabilistic variables,  $X$  is reachable from  $s$ . The clauses, that we define hereafter, must ensure that an assignment is a model only if  $v_t = \perp$  given the choices for the edges.

To ensure that, the clauses use the transitive nature of the connectivity. That is, for each edge  $e$  from  $u$  to  $v$ , a clause  $v_u \wedge p_{uv} \Rightarrow v_v$  is created. This can be interpreted as that if  $u$  is reachable from  $s$  and  $e$  is present, then  $v$  is also reachable from  $s$ . We impose that  $s$  is reachable from  $s$  by adding the clause  $\top \Rightarrow s$  and that  $t$  is not reachable by adding the clause  $v_t \Rightarrow \perp$ . The clauses for the query  $P(\bar{R}_E^A)$  for the graph in Figure 1 are shown below.

$$\begin{array}{llll}
 v_A \wedge p_{AB} \Rightarrow v_B & v_B \wedge p_{BD} \Rightarrow v_D & v_C \wedge p_{CE} \Rightarrow v_E & v_E \Rightarrow \perp \\
 v_A \wedge p_{AC} \Rightarrow v_C & v_C \wedge p_{CD} \Rightarrow v_D & v_D \wedge p_{DE} \Rightarrow v_E & \top \Rightarrow v_A
 \end{array}$$

► **Example 4** (Probabilistic programming). ProbLog is a probabilistic programming language that extends Prolog with probabilistic predicates [8]. It can be used to represent both the aforementioned inference problems. While a full discussion of this language is beyond the scope of this paper, we wish to illustrate how PPHMC can be used in ProbLog. Consider the following example ProbLog program, taken from the ProbLog website<sup>2</sup>:

```
0.4 :: heads.
0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).
win :- heads, col(_,red).
win :- col(1,C), col(2,C).
query(win).
```

For such a logic program, ProbLog performs *grounding*, creating a propositional version of the program. This propositional version can be represented as follows in our language:

$$\begin{aligned} p_{heads}, p_{col(1,red)} &\Rightarrow v_{win} & p_{col(1,blue)}, p_{col(2,blue)} &\Rightarrow v_{win} \\ p_{heads}, p_{col(2,red)} &\Rightarrow v_{win} & v_{win} &\Rightarrow \perp \\ p_{col(1,red)}, p_{col(2,red)} &\Rightarrow v_{win} & & \end{aligned}$$

Here we have these partitions:  $w(p_{heads}) = 0.4$ ,  $w(\bar{p}_{heads}) = 0.6$ ;  $w(p_{col(1,red)}) = 0.3$ ,  $w(p_{col(1,blue)}) = 0.7$ ;  $w(p_{col(2,red)}) = 0.2$ ,  $w(p_{col(2,green)}) = 0.3$ ,  $w(p_{col(2,blue)}) = 0.5$ . The probability of the query is obtained by PPHMC on this ground version. Given that our previous example showed how reliability problems can be modeled in polynomial space, while the model for this problem is exponential in the grounded ProbLog model, we hypothesize that this is possible for all grounded programs without cycle breaking.

## 4 Algorithm

In the section we present the main algorithms of our solver. At the core, the problem is solved by a backtracking search over the possible assignments for the distributions. When a value is assigned to a variable, we call a new propagator designed specifically for the structure of the clauses. We first present the general algorithm for a search-based solver for the PWMC problem and then the specific propagation algorithm as well as some branching heuristics.

### 4.1 General Approach

The main algorithm is shown in Algorithm 1. In essence, it is similar to other DPLL-based solvers. The computation of the count for a non-empty formula  $F$  starts by looking into a cache (line 7) to determine if the formula has already been counted. If not, then it chooses an unfixed variable (lines 8, 10) and assigns it a value. A residual formula is computed after calling a propagation algorithm (line 11) which is then divided into independent components (line 14). The components are then solved independently (line 17) and their count is stored in the cache (line 18). In order to bound the memory consumption of search based solvers, the cache has a limited number of entries. As cache cleaning techniques are not the focus of this work, we simply fully clear the cache when the limit is reached.

<sup>2</sup> [https://dtai.cs.kuleuven.be/problog/tutorial/advanced/00\\_inference.html](https://dtai.cs.kuleuven.be/problog/tutorial/advanced/00_inference.html)

■ **Algorithm 1** General PPHMC search algorithm.

---

```

1 Function PPMC( $F, \mathcal{P}$ )
  input : A Horn-CNF formula  $F$  with probabilistic variables  $\mathcal{P}$ 
  output : The projected (on  $\mathcal{P}$ ) weighted model count of  $F$ 
2    $C \leftarrow \text{newCache}()$ 
3   return PPMCr( $F, \mathcal{P}, C$ )
4
5 Function PPMCr( $F, \mathcal{P}, C$ )
  input :  $F, \mathcal{P}$  same as PPMC()
  input :  $C$  the cache of previously found counts
  output : Same as PPMC()
6  if  $\mathcal{P} = \emptyset$  then return 1
7  if  $F \in C$  then return  $C[F]$ 
8   $P \leftarrow$  a distribution of  $\mathcal{P}$  such that  $\exists v \in P \mid v$  is not fixed
9   $count \leftarrow 0$ 
10 foreach  $v \in P \mid v$  is not fixed do
    /* Assign  $v = \top$  and call the propagation algorithm. Returns the
       residual formula  $F'$  and the unconstrained probability of  $F$ 
       given  $F'$ . */
11    $(F', U_{F|F'}) \leftarrow \text{Propagate}(F, \mathcal{P}, v)$ 
12    $proba \leftarrow U_{F|F'}$ 
13   if  $F'$  is not UNSAT then
14      $Components \leftarrow$  all connected components of  $F'$ 
15     foreach  $Comp \in Components$  do
16        $\mathcal{P}' \leftarrow \mathcal{P}$  reduced to the variables in  $Comp$ 
17        $proba_{Comp} \leftarrow \text{PPMCr}(Comp, \mathcal{P}', C)$ 
18        $C[Comp] \leftarrow proba_{Comp}$ 
19        $proba \leftarrow proba * proba_{Comp}$ 
20     end
21      $count \leftarrow count + proba$ 
22   end
23 end
24 return  $count$ 

```

---

There are a few differences between our solver and other model counters that need to be pointed out. First, let us note that the variables in a distribution are mutually exclusive. Indeed, if a distribution does not have exactly one variable set to  $\top$  in an assignment, then its weight is 0 and  $\prod_i w_{P_i}(S) = 0$ , which does not contribute to the count.

A consequence of this is that, unlike classical DPLL-based counters, the branching decision is made on distributions and not variables. Indeed, since the distributions partition of the probabilistic variables (i.e., the variables on which the number of weighted models must be counted), assigning one variable to  $\top$  in each distribution means that all other probabilistic variables are set to  $\perp$ . Moreover, when fixing a variable  $v$  in the selected distribution, only the case of fixing to  $\top$  needs to be explored. The case  $v = \perp$  is explored when the other variables in the distribution are selected for branching. Notice that when there is no distribution left in  $F$ , then 1 is returned as the remaining formula is SAT (line 6). Indeed, there are no unit clauses (they are removed during the propagation) and all that remains in  $F$  are

Horn clauses with deterministic variables. By setting all remaining variables to  $\perp$  we obtain a model of this formula, as the implicant of all clauses evaluates to  $\perp$ . This would not be possible if  $F$  was not solely composed of Horn clauses. In this case, the SAT problem needs to be solved on  $F$ , which is NP-hard.

The computation of independent components is also slightly different. In traditional model counters, when the input formula  $F$  can be decomposed into multiple sub-formulas that do not share any variable, each sub-formula is solved independently and the count of  $F$  is the product of the sub-formulas' count. However, since the probabilistic variables are linked in partitions, we must add as condition that two independent sub-formulas cannot share any variable in the same distribution.

Finally, we also devised a new propagation algorithm, explained in the next section. It returns (line 11) a residual formula  $F'$  as well as what we call the *unconstrained probability of  $F$  given  $F'$* :  $U_{F|F'}$ . This probability accounts for the distributions of  $F$  that are unconstrained in  $F'$ . Indeed, if for a distribution  $P_i = \{p_i^1, \dots, p_i^m\}$  occurring in  $F$ , there are no clause in  $F'$  that contains any of the  $p_i^k \in P_i$ ,  $P_i$  will never be selected by the branching heuristic. Moreover, branching on it will not impact  $F'$ . Hence, the probability obtained by branching of  $P_i$  can be precomputed and is given by  $\sum_{p \in P_i | p \text{ is not fixed}} w(p)$ . Note that this sum may not be 1 if propagation fixed one of the  $p_i^k$  variables to  $\perp$  earlier, and  $|P_i| > 2$ . More generally, for  $l$  such distributions  $P_{u_1}, \dots, P_{u_l}$  we have  $U_F = \prod_{i=1}^l \sum_{p \in P_{u_i} | p \text{ is not fixed}} w(p)$ .

## 4.2 Propagation

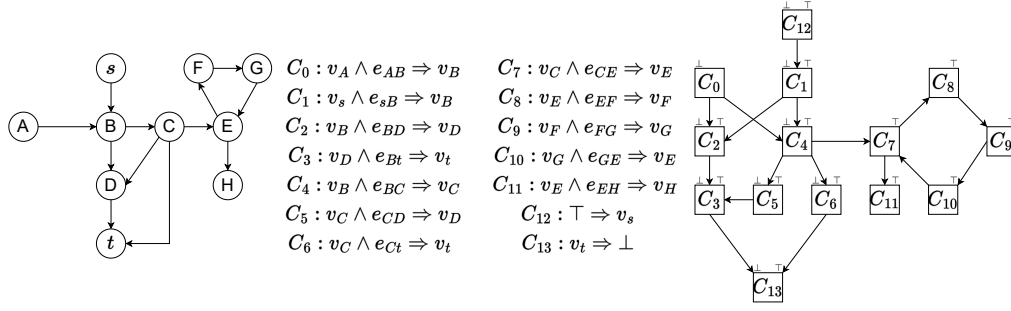
In this section, we describe the propagation algorithm used by our solver, summarized in Algorithm 2. In brief, we first apply the classical *Boolean Unit Propagation* (BUP), in which the links between probabilistic variables in partitions are also enforced, until a fixed point is reached. Then we remove from the remaining formula the clauses that do not constrain the count anymore. First, let us detail what the BUP does when a variable  $v$  is fixed in  $F$ :

- If  $v = \top$  then
  - If  $v$  is a probabilistic variable in a distribution  $P$ , apply the BUP on all  $v' \in P, v' \neq v$  with  $v' = \perp$ .
  - Every clause  $C = (I, h)$  such that  $h = v$  is removed from  $F$ . Indeed, for every assignment on its remaining variables, it evaluates to  $\top$ .
  - For every clause  $C = (I, h)$  such that  $v \in I$ , replace  $I$  by  $I' = I \setminus \{v\}$ .
- If  $v = \perp$  then
  - If  $v$  is a probabilistic variable in a distribution  $P$  and only one variable  $v'$  remains unfixed in  $P$ , apply BUP with  $v' = \top$ .
  - Every clause  $C = (I, h)$  such that  $v \in I$  is removed from  $F$  for the same reason as above.
  - The head of every clause  $C = (I, h)$  such that  $h = v$  is replaced by  $\perp$ .

There are two cases in which, after a call to the BUP, a variable is forced to take a value, and the BUP algorithm needs to be called again. First, when the last variable of an implicant is removed from it, then the head of the clause must be  $\top$ . Secondly, when a clause  $(I, h)$  has its head set to  $\perp$  and there is only one variable left in  $I$ , then this variable must be set to  $\perp$ . In Algorithm 2, the call to BUP at line 2 is executed until no such clauses can be found.

However, in the context of *projected* model counting, a key insight for our work is that further propagation can be done that is not entailed by BUP. The intuition is that the deterministic variables can be used to remove additional clauses from  $F$ . For example, if our formula includes a clause  $\neg a \vee \neg b \vee \neg c$  with  $a, b$  being probabilistic variables and  $c$  a





■ **Figure 2** On the left: an instance of a reliability problem on probabilistic networks. In the center: the associated clauses for the query  $P(\bar{R}_t^s)$ . On the right: the graph of clauses implication: there is one node per clause and a link between  $C_i$  and  $C_j$  if  $h_i \in I_j$ .

deterministic variable that does not appear in any other clauses, then setting  $c$  to  $\perp$  makes the clause evaluate to  $\top$ , regardless of the choice for  $a$  and  $b$ . Since we are only interested in finding an assignment to the deterministic variables, this assignment does not impact the final count.

Let us show how this works on the clauses in Figure 2, taken from a reliability query in a probabilistic networks, in which we want to compute  $P(\bar{R}_t^s)$ , the probability that  $s$  and  $t$  are not connected in the graph on the left. Applying BUP gives  $v_s = \top$  and  $v_t = \perp$ , which removes these variables from  $C_1, C_3$  and  $C_6$ . No further propagation can be done with BUP. However, when looking at the graph on the left, it is clear that only the nodes  $B, C$  and  $D$  impact the connectivity between  $s$  and  $t$ , but this is not detected by BPU. For instance, let us look at nodes  $A$  and the associated clause  $C_0$ , which contains the distribution  $\{e_{AB}, \bar{e}_{AB}\}$ . It can be seen that for both choices for the edge  $e_{AB}$ , setting  $v_A = \perp$  (a deterministic variable) reduces the clause  $C_0$  to  $\top$ . Since there are no clauses that have  $v_A$  in their head, setting  $v_A$  to  $\perp$  has no impact on the other variables in  $F$ . Moreover, since it is a deterministic variable, it does not impact the projected weighted model count. A similar reasoning can be made for  $H$  and  $C_{11}$  by setting  $v_H = \top$ , since  $v_h$  does not appear in any implicant in  $F$ .

The intuition behind our propagation is the following. In order for an assignment to not be a model of the input formula, it must generate a clause  $\top \Rightarrow \perp$ . Some clauses cannot contribute to such a contradiction, by setting a deterministic variable to  $\perp$  in its implicant or its head to  $\top$ . We formalize this intuition next.

First, let us define in which case we cannot know if a clause will have its head set to  $\perp$  or its implicant set to  $\top$ .

► **Definition 5** ( $\{\top, \perp\}$ -reachability). A clause  $C_i = (I_i, h_i) \in F$  is  $\perp$ -reachable if one of the two following conditions is met:

1.  $C_i$  is of the form  $I_i \Rightarrow \perp$  or  $I_i \Rightarrow p$  with  $p \in \mathcal{P}$
2. There exists a clause  $C_j = (I_j, h_j) \in F$  such that  $C_j$  is  $\perp$ -reachable and  $h_i \in I_j$ .

Similarly,  $C_i$  is  $\top$ -reachable if one of the two following conditions is met:

1. There exists no deterministic variables in  $I_i$
2. There exists a clause  $C_j = (I_j, h_j) \in F$  such that  $C_j$  is  $\top$ -reachable and  $h_j \in I_i$ .

We say that a clause is *constrained* if it is  $\perp$ -reachable and  $\top$ -reachable. The  $\{\perp, \top\}$ -reachability can be seen for the clauses in Figure 2 on the right, on the implication graph of the clauses. The implication graph of a set of Horn clauses is a graph  $G = (V, E)$  such that there is one node per clause and an edge from a clause  $C_i = (I_i, h_i)$  to  $C_j = (I_j, h_j)$  if  $h_i \in I_j$ . If a clause is  $\top$ -reachable ( $\perp$ -reachable), so are all its descendants (ancestors) in the implication graph.

■ **Algorithm 2** Propagation algorithm of Schlandals.

---

```

1 Function Propagate( $F, \mathcal{P}, v$ )
   input : A boolean formula  $F$  with probabilistic variables  $\mathcal{P}$ 
   input : A variable  $v$  set to  $\top$ 
   output: The residual formula  $F'$  and a propagation probability  $p_{prog}$ 
   /* Call the BUP procedure with the initial assignment of  $\top$  to  $v$ 
      until fix point is reached */
2  $F' \leftarrow \text{BUP}(F, v, \top)$ 
3 foreach  $C = (I, h) \in F'$  do
4   | if  $h = \perp$  or  $h \in \mathcal{P}$  then SetFReachable( $F', C$ )
5   | if  $\nexists v \in I \mid v \notin \mathcal{P}$  then SetTReachable( $F', C$ )
6 end
7 foreach  $C \in F'$  do
8   | if  $C$  is not  $\top$ -reachable or  $C$  is not  $\perp$ -reachable then
9   | |  $F' \leftarrow F' - C$ 
10  | end
11 end
12  $U_{F|F'} \leftarrow 1$ 
13 foreach distribution  $P$  such that  $P \in F \wedge P \notin F'$  do
14  |  $U_F \leftarrow U_F * \sum_{p \in P \mid p \text{ is not fixed}} w(p)$ 
15 end
16 return ( $F', U_{F|F'}$ )

```

---

We have the following theorem, which states that if a clause is not constrained, then it can be removed safely (without impacting the count) from  $F$ .

► **Theorem 6.** *Let  $F = C_1 \wedge \dots \wedge C_n$  be a formula with  $n$  Horn clauses over the variables  $\mathcal{V}$ ,  $\mathcal{P} \subseteq \mathcal{V}$  the set of probabilistic variables and  $\mathcal{D} = \mathcal{V} \setminus \mathcal{P}$  the set of deterministic variables. Let  $C_{u_1}, \dots, C_{u_k}$  be  $k$  unconstrained clauses of  $F$  with  $C_{u_i} = (I_{u_i}, h_{u_i})$ .*

*There exists a subset of  $k$  deterministic variables  $X = \{x_1, \dots, x_k\} \subseteq \mathcal{D}$  with  $x_i \in I_{u_i} \cup \{h_{u_i}\}$  and an assignment  $S_X$  on  $X$  such that*

$$\mathcal{S}_F = \mathcal{S}_{F[S_X]}$$

where  $\mathcal{S}_F$  denotes the set of models of  $F$ , projected on  $\mathcal{P}$ , and  $F[S_X]$  the formula obtained by applying the BUP algorithm on  $F$  with the assignment  $S_X$ .

We now prove this theorem and give the procedure to find the assignment on the deterministic variables.

**Proof.** Let  $G = (V, E)$  be the graph of the implications of  $F$  and  $C_i$  an unconstrained clause in  $F$ . We prove that an assignment can be found for one of the deterministic variables of  $C_i$  such that it does not impact the count of  $F$ .

First, let us assume that  $C_i$  is not  $\perp$ -reachable. We denote  $G_{C_i}^d = (V_{C_i}^d, E_{C_i}^d)$  the sub-graph of  $G$  that contains  $C_i$  and all its descendants. By definition there is no clause  $C_j \in V_{C_i}^d$  that is  $\perp$ -reachable, otherwise  $C_i$  would be  $\perp$ -reachable. Hence, all clauses in  $V_{C_i}^d$  are unconstrained. Let  $X^d = \cup_{C_j \in V_{C_i}^d} \{h_j\}$  be the set of (deterministic) heads in  $G_{C_i}^d$ . We define the assignment  $S_{X^d}$  such that  $S_{X^d}[x] = \top, \forall x \in X^d$ . This removes all the clauses in  $V_{C_i}^d$  from  $F$  without impacting the values of the other variables in the clauses.

## 15:12 Probabilistic Inference by Projected Weighted Model Counting on Horn Clauses

Next, if  $C_i$  is  $\perp$ -reachable but unconstrained, then it is not  $\top$ -reachable. Let  $G_{C_i}^p = (V_{C_i}^p, E_{C_i}^p)$  be the sub-graph of  $G$  that contains  $C_i$  and all its parents. Since  $C_i$  is  $\perp$ -reachable, so is every clause in  $V_{C_i}^p$ . Thus we have that every clause  $C_j \in V_{C_i}^p$  has at least one deterministic node  $d_j \in I_j$ , otherwise they would be  $\top$ -reachable. Let  $X^p = \cup_{C_j \in V_{C_i}^p} \{d_j\}$  be the set of such nodes. We set  $S_{X^p}$  such that  $S_{X^p}[d_j] = \perp$  for all clause  $d_j \in X^p$  which removes all clauses in  $V_{C_i}^p$  from  $F$  without constraining the other variables. ◀

■ **Algorithm 3** Procedure to mark the clauses of a formula  $F$  as  $\perp$ -reachable.

---

```

1 Function SetFReachable( $F, C$ )
  input : A boolean formula  $F$ , a clause  $C = (I, h)$  of  $F$ 
2   if  $C$  is not marked as  $\perp$ -reachable then
3     Mark  $C$  as  $\perp$ -reachable
4     foreach  $C' = (I', h') \in F \mid h' \in I$  do SetFReachable( $F, C'$ )
5   end

```

---

This additional propagation is shown in lines 3-11 of Algorithm 2. After the application of BUP until a fix point is reached, the remaining clauses are iterated over. If a clause is of the form  $C = (I, \perp)$  or has a head with an unfixed probabilistic variable, then the procedure `SetFReachable` is called, for which the code is shown in Algorithm 3. This algorithm basically traverses the implication graph of  $F$ , starting from  $C$  and marks every clause it encounters as  $\perp$ -reachable. Notice that it does not mark multiple times the same clause. Hence, the cost of marking all the  $\perp$ -reachable clauses is  $\mathcal{O}(n)$  with  $n$  the number of clauses in  $F'$ . A similar procedure is defined for the  $\top$ -reachability, but we do not include it for conciseness. After marking the clauses (lines 3-5), every unconstrained clause is removed from the formula obtained after BUP (lines 7-11). Finally, the algorithm computes the unconstrained probability  $U_{F|F'}$  (lines 12-15).

It can be noted that in some cases our propagation is similar to *Pure Literal Elimination* (PLE) on the deterministic variables. For instance, in Figure 2,  $v_A$  never appears in the head of any clause. Hence, it can be set as  $\perp$ , which is also the value found by our procedure. However, PLE cannot detect that the nodes E, F and G are not useful for the query. It should also be noted that although our propagation implies that PLE is performed, the procedure only reasons about the clauses, which makes it more efficient.

### 4.3 Branching Heuristic

At line 8 of Algorithm 1, a distribution is selected within the set of distributions for which no element is set to  $\top$ . In this section we explain the heuristics implemented in our solver to choose the distribution. We must note that our solver is not a conflict-driven clause learning (CDCL) solver, making it impossible to use the heuristics of such solvers [19, 22]. Instead, we implemented some easy to understand, fast to evaluate, but effective heuristics based on the implication graph of a formula  $F$ . We provide three heuristics related to the degree of a clause in the implication graph. These heuristics select a clause with i) the lowest in-degree ii) the lowest out-degree and iii) the maximum degree, and then a distribution in this clause. Each time they are called, they re-evaluate the score of each clause in the current sub-formula.

## 5 Results

In order to evaluate the effectiveness of our approach, we compare our solver<sup>3</sup> with `GPMC` [25] and `ProjMC` [17] on Bayesian networks and reliability queries in probabilistic networks. We chose these solvers as they were the best performing on the Projected Model Counting and Projected Weighted Model Counting tracks of the 2022 model counting competition. We did not include the `proCount` solver [11] as it is a knowledge compilation based solver. `Ganak` [22] also was not used as we found it was the worst performing on the projected tasks (as also observed in the 2022 competition). For the reliability queries, we also ran `ApproxMC` [3, 23, 24] as it is known to perform well on this problem, although it should be stressed `ApproxMC` solves an *approximate* model counting problem, while we solve an exact model counting problem. For each of the two problems we first present the methodology used to generate the instances and then present the results<sup>4</sup>. For each instance, a timeout of 600 seconds and a memory limit of 15 GB were set.

### 5.1 Bayesian Networks

The Bayesian networks come from the `bnlearn` R package [21] repository and range from small networks (fewer than 20 nodes) to large (up to hundreds of nodes). We selected all the networks in the repository for which at least one solver did not time out. For each of these networks, the queries are done on the leaves of the networks without any evidence. Hence, for a leaf  $L$  of the network that takes values  $l_1, \dots, l_n$ , we create  $n$  instances for the queries  $P(L = l_1), \dots, P(L = l_n)$ .

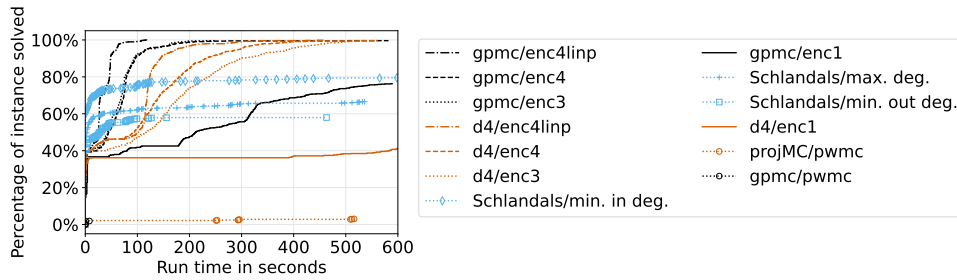
The goal of this experiment is to answer the following questions: i) how do the evaluated PWMC solvers perform using our simple encoding? ii) How does our encoding perform compared to other state-of-the-art, but more complex encodings for BNs, designed for weighted model counters? For i) we use a CNF formula in other PWMC solvers similar to our model, where we add additional clauses to enforce the distribution constraints that we have in our weighting scheme. For ii), we follow the nomenclature in [6], and we compare our encoding with `ENC1` [7], `ENC3` [4], and `ENC4` [5]. We also add the encoding recently presented in [2], which we denote `ENC4linp`. Briefly, `ENC1` is the simplest encoding: there is one indicator variable  $\lambda^x$  for each value  $x$  of node  $X$  and one parameter variable  $\theta_{\mathbf{u}}^x$  per parameter  $P(x | \mathbf{u})$  of the network. For the clauses, the indicator variables of the same node are mutually exclusive; a clause  $\lambda^{u_1} \wedge \dots \wedge \lambda^{u_n} \wedge \lambda^x \Leftrightarrow \theta_{\mathbf{u}}^x$  is created for each parameter  $P(x | \mathbf{u})$ . The main addition of `ENC3` is that, for a given CPT in the network, the same variable  $\theta_{\mathbf{u}}^x$  is reused for all the entries having the same probability. In `ENC4`, The CPTs are simplified using a modified version of Quine/McCluskey algorithm for finding prime implicants, resulting in a better decomposition of the input CNF. Finally, in `ENC4linp`, the authors propose two novelties for encoding BNs in a CNF: the use of log-encoding for the elementary assignment of a variable and one parameter variable per CPT is kept implicit. For these encodings, `GPMC` and `projMC` are run in WMC mode (which is the `d4` [16] model counter for the latter).

Figure 3 shows the percentage of instances solved within the time limit.

First, it can be seen that if we use our model in other PWMC solvers, their performance is much worse; these solvers solve a few instances.

<sup>3</sup> Available at <https://github.com/aia-uclouvain/schlandals>

<sup>4</sup> The instances can be found here [https://github.com/AlexandreDubray/pwmc\\_benchmarks](https://github.com/AlexandreDubray/pwmc_benchmarks)



■ **Figure 3** Percentage of instances solved in 600 seconds for the Bayesian Network data sets.

Compared to the ENC1 model used in traditional WMC, which is arguably the model for traditional WMC most similar to our model, we can still observe that our approach performs significantly better, solving 80% of the instances for the minimum in degree heuristic. We believe it is highly encouraging that our approach works so well for such a simple model.

When evaluating the more complex, optimized models ENC3, ENC4 and ENC4linp developed for WMC, we can see that these have increasingly better results. This confirms the benefit of the various improvements made, over the years, to the encodings. Overall, the GPMC solver is the best performing of all, even when using only the ENC3 encoding. The biggest gap between the ENC encodings is between ENC1 and ENC3. Indeed, using the same parameter variables for multiple parameters of a CPT gives a huge drop in the number of variables, especially for the hardest instances, which contain large CPTs. The benefits of the Quine/McCluskey reduction are clearly visible for the d4 solver.

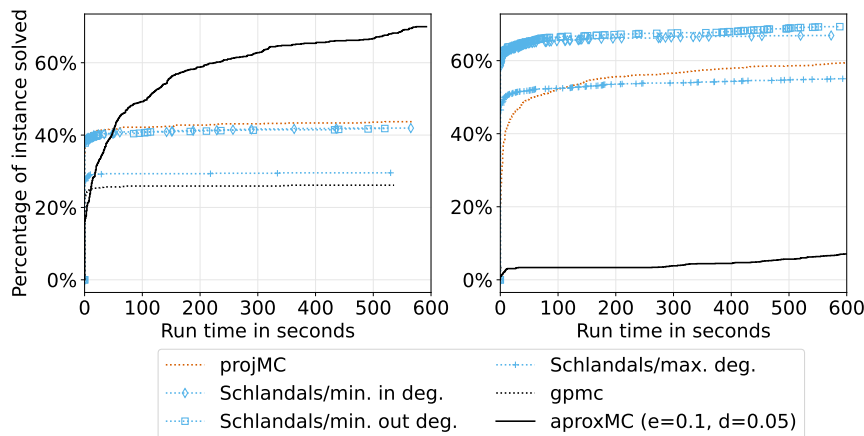
The nature of our modeling language is such that the optimizations of ENC3, ENC4 and ENC4linp cannot be directly applied to our models. Unlike the ENC encodings, we require that one variable per distribution is set to  $\top$ . However, we hypothesize that similar optimizations can also be developed for our solver in future work.

## 5.2 Connectivity in Probabilistic Networks

For this problem, the data sets used come from two sources. First, we used the power grid network of Europe and USA as extracted by the GridKit tool [18, 28]. The extracted graphs represent the electric power system in these geographical areas. In order to have various sizes of instances, both networks are divided by country (Europe) or by state (USA). Then for each subnetwork, five random pairs of nodes are selected and the probability that they are connected is computed. Notice that since the edges in the power-grid network have no orientation (the graph is undirected), we transformed it into a directed graph by replacing each edge between  $u$  and  $v$  by one edge from  $u$  to  $v$  and one edge from  $v$  to  $u$ . We assume that each edge has a probability of 0.125 to be down, as done in [12].

Secondly, we used (oriented) graphs representing water distribution systems from the WNTR Python package [14]. We considered as sources (sinks) nodes having no parents (children). Then, we compute  $P(\bar{R}_t^s)$  between each source-target pair  $(s, t)$  such that there exists a path between  $s$  and  $t$ . As for the power grid network data set, we assume a fix probability of 0.125 that each link is down.

Unlike for Bayesian Networks, our encoding and the one proposed in [12] are almost identical and can be reused for all the evaluated solvers. The only difference is that we do not create two variables per edge for the encoding passed to GPMC and projMC. Figure 4 shows the percentage of instances solved in the given amount of time for both of these data sets.



■ **Figure 4** Percentage of instances solved in 600 seconds for the power-grid networks (left) and the water networks (right).

First, let us note that **GPMC** performs the least well on both data sets. While it is able to solve up to 25% of the instances for the power grid network, it is unable to solve the largest instances of the water networks. For the latter data set, it quickly reaches the memory limit (15 GB) and stagnates until the time-out. On the other hand, it can be seen that **projMC** and **Schlandals** solve roughly the same number of instances on the power grid networks, while **Schlandals** solves 10% more instances on the water networks. On these types of data sets, the min-in degree and min-out degree heuristics work best for our solver. It can be seen that for the water networks, **Schlandals** outperforms **projMC**.

Interestingly, on the power grid data sets, our solver is still able to match the performance of **projMC** even though our propagation is not used to its full capacity. Indeed, since the original graphs are not directed, they were transformed by adding two directed edges for each undirected edge. As a result, either all clauses in a component are constrained, or no clause is constrained. Indeed, by doubling the edges in the original graph, the edges in the implication graph of the resulting CNF are also doubled. Hence, every clause which is a descendant of a  $\perp$ -reachable clause is also one of its ancestors and a similar argument can be made for  $\top$ -reachable clauses. Hence, our propagation algorithm has less pruning power. On the other hand, for the water network data set, our solver uses the full strength of our propagation and is more efficient than **projMC**.

Let us briefly comment on the performance of **approxMC** [12]. Again, it should be noted that **approxMC** is an approximate model counter that provides provides  $(\delta - \epsilon)$ -guarantees and does not consider weights on the literals. Hence, the problem it solves is quite different from the other solvers compared in this work: if the count returned by **approxMC** is  $C$ , and the exact count  $C^*$ , it ensures that  $\frac{C^*}{1+\epsilon} < C < C^*(1 + \epsilon)$  with a confidence of  $1 - \delta$  ( $\delta, \epsilon \in [0, 1]$ ). In Figure 4 the results are shown for  $\epsilon = 0.1$  and  $\delta = 0.05$ . This is an example of configuration in which the solver is quite confident in its solution, and hence the results are more comparable to that of the other solvers. It can be seen that it performs very well on the power grid network instances, but poorly on the water supply network ones. Using  $\epsilon = 0.8$  and  $\delta = 0.2$ , as reported in [12], the solver is able to solve all instances on both data sets, but it is much less confident in its solution. Overall the performance of **approxMC** heavily depends on the acceptable margin of error.

## 6 Conclusion

Weighted model counters have become an essential tool in probabilistic reasoning, but the CNF models have grown more and more complex. A step towards simplicity for some problems has been taken by the introduction of projected weighted model counting. In this work, we propose the Projected Probabilistic Horn model counting (PPHMC) problem, in which only Horn clauses are allowed, making the modeling language simpler. We have shown that with an appropriate weighting scheme it is possible to model important probabilistic problems as PPHMC problems. In particular, we provide an encoding for Bayesian networks, probabilistic networks and probabilistic logic programs. We also introduced a new tool, the Schlandals solver, specifically designed for our language. Our experiments show that our solver is competitive with state-of-the-art solvers and opens the path to further work on PPHMC.

As we have seen for the Bayesian Networks, the encoding can have a great impact on the performance. An interesting line of work would be to investigate how the optimizations developed during the past twenty years, for the encoding of Bayesian Networks into a logical formula, can be applied with our weighting schema, and how our solver can be integrated in probabilistic logic programming systems. On the other hand, a lot of work has been done to make model counters efficient. In the future, integrating such techniques (probabilistic caching, tree decomposition, symmetry breaking, advanced branching heuristics, etc.) with the specificity of PPHMC can also increase the performance of our solver. Finally, this paper has been focused on PWMC within bounded memory. However, the core of our solver can be reused in a knowledge compiler or an approximate solver.

---

## References


- 1 Rehan Abdul Aziz, Geoffrey Chu, Christian Muise, and Peter Stuckey.  $\#\exists$  SAT: Projected model counting. In *International Conference on Theory and Applications of Satisfiability Testing 2015*. Springer, 2015.
- 2 Anicet Bart, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. An improved CNF encoding scheme for probabilistic inference. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 2016.
- 3 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic sat calls. In *IJCAI*, 2016.
- 4 Mark Chavira and Adnan Darwiche. Compiling Bayesian networks with local structure. In *IJCAI*, volume 5, 2005.
- 5 Mark Chavira and Adnan Darwiche. Encoding CNFs to empower component analysis. In *Theory and Applications of Satisfiability Testing-SAT 2006: 9th International Conference, Seattle, WA, USA, August 12-15, 2006. Proceedings 9*. Springer, 2006.
- 6 Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7), 2008.
- 7 Adnan Darwiche. A logical approach to factoring belief networks. *KR*, 2, 2002.
- 8 Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7. Hyderabad, 2007.
- 9 Paulius Dilkas and Vaishak Belle. Weighted model counting with conditional weights for Bayesian networks. In *Uncertainty in Artificial Intelligence*. PMLR, 2021.
- 10 William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3), 1984.
- 11 Jeffrey M. Dudek, Vu H. N. Phan, and Moshe Y. Vardi. ProCount: Weighted Projected Model Counting with Graded Project-Join Trees. In Chu-Min Li and Filip Manyà, editors, *Theory*

- and Applications of Satisfiability Testing – SAT 2021*, Lecture Notes in Computer Science, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-80223-3\_11.
- 12 Leonardo Duenas-Osorio, Kuldeep Meel, Roger Paredes, and Moshe Vardi. Counting-based reliability estimation for power-transmission grids. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.
  - 13 Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15(3), 2015.
  - 14 Katherine A. Klise, Michael Bynum, Dylan Moriarty, and Regan Murray. A software framework for assessing the resilience of drinking water systems to disasters with an example earthquake case study. *Environmental modelling & software*, 95, 2017.
  - 15 Tuukka Korhonen and Matti Järvisalo. Integrating tree decompositions into decision heuristics of propositional model counters. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
  - 16 Jean-Marie Lagniez and Pierre Marquis. An Improved Decision-DNNF Compiler. In *IJCAI*, volume 17, 2017.
  - 17 Jean-Marie Lagniez and Pierre Marquis. A recursive algorithm for projected model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2019.
  - 18 Wided Medjroubi, Ulf Philipp Müller, Malte Scharf, Carsten Matke, and David Kleinhans. Open data in power grid modelling: New approaches towards transparent grid models. *Energy Reports*, 3, 2017.
  - 19 Tian Sang, Paul Beame, and Henry Kautz. Heuristics for fast exact model counting. In *Theory and Applications of Satisfiability Testing: 8th International Conference, SAT 2005, St Andrews, UK, June 19-23, 2005. Proceedings 8*. Springer, 2005.
  - 20 Tian Sang, Paul Beame, and Henry Kautz. Solving Bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 1. AAAI Press, 2005.
  - 21 Marco Scutari. Learning Bayesian networks with the bnlearn R package. *arXiv preprint arXiv:0908.3817*, 2009. arXiv:0908.3817.
  - 22 Shubham Sharma, Subhjit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A Scalable Probabilistic Exact Model Counter. In *IJCAI*, volume 19, 2019.
  - 23 Mate Soos, Stephan Gocht, and Kuldeep S. Meel. Tinted, detached, and lazy cnf-xor solving and its applications to counting and sampling. In *International Conference on Computer Aided Verification*, 2020.
  - 24 Mate Soos and Kuldeep S. Meel. Bird: engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In *AAAI*, 2019.
  - 25 Ryosuke Suzuki, Kenji Hashimoto, and Masahiko Sakai. Improvement of projected model-counting solver with component decomposition using SAT solving in components. Technical report, JSAI Technical Report, SIG-FPAI-506-07, 2017.
  - 26 Marc Thurley. sharpSAT-counting models with advanced component caching and implicit BCP. *SAT*, 4121, 2006.
  - 27 Jonas Vlasselaer, Angelika Kimmig, Anton Dries, Wannes Meert, and Luc De Raedt. Knowledge compilation and weighted model counting for inference in probabilistic logic programs. In *Proceedings of the First Workshop on Beyond NP*. AAAI Press, 2016.
  - 28 Bart Wiegmans. Gridkit: European And North-American Extracts, March 2016. doi:10.5281/ZENODO.47317.






# A CP Approach for the Liner Shipping Network Design Problem

Yousra El Ghazi ✉ 

Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Djamal Habet ✉ 

Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Cyril Terrioux ✉ 

Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

---

## Abstract

The liner shipping network design problem consists, for a shipowner, in determining, on the one hand, which maritime lines (in the form of rotations serving a set of ports) to open, and, on the other hand, the assignment of ships (container ships) with the adapted sizes for the different lines to carry all the container flows. In this paper, we propose a modeling of this problem using constraint programming. Then, we present a preliminary study of its solving using a state-of-the-art solver, namely the OR-Tools CP-SAT solver.

**2012 ACM Subject Classification** Computing methodologies → Artificial intelligence

**Keywords and phrases** Constraint optimization problem, modeling, solving, industrial application

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.16

**Supplementary Material Dataset:** <https://pageperso.lis-lab.fr/cyril.terrioux/LSNDP/instances.zip>

**Funding** This work has been funded by Bpifrance under the PIA project Transformation Numérique du Transport Maritime (TNTM).

**Acknowledgements** The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

## 1 Introduction

Nowadays, maritime transport plays a major role in world trade. According to the International Maritime Organization (IMO), more than 80% of international trade is carried out by sea. The transport of containerized commodities constitutes the major part of this trade. It relies on more than 5,000 container ships that serve more than 500 ports worldwide. In this context, many combinatorial optimization problems [6, 9, 28] may arise with non-negligible economic and ecological impacts given their scale.

In this paper, we focus on the Liner Shipping Network Design Problem (LSNDP [9]). A shipping line, also called a service, is defined by a cyclic route (called a rotation) that visits a given set of ports in a given order and at regular times (see, for example, Figure 1). Generally, each port is thus visited by a vessel of the line at a weekly or biweekly frequency. All the vessels on a line are assumed to be homogeneous in terms of their main features (loading capacity, speed, fuel consumption, engines, ...). Operating a weekly line with a rotation lasting  $k$  weeks requires  $k$  vessels of the same type. Given a set of ports, a fleet of container ships, and a container flow (defined by a set of triples consisting of the original port of the commodities, their destination port, and the number of containers they represent), the LSNDP problem consists, for a shipowner, in determining, on the one hand, which shipping lines to open, and, on the other hand, which ships to operate on each line in order to carry as



© Yousra El Ghazi, Djamal Habet, and Cyril Terrioux;  
licensed under Creative Commons License CC-BY 4.0

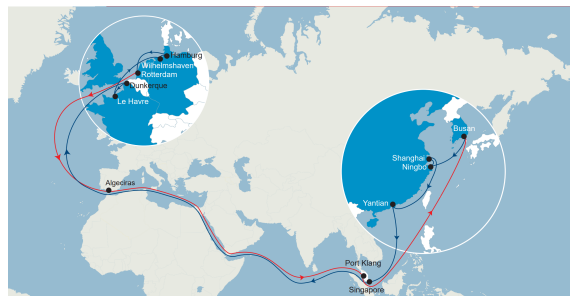
29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 16; pp. 16:1–16:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Example of a line connecting Asia and Europe.

many commodities as possible while ensuring a weekly frequency of visits to each port and optimizing costs. It is classified as NP-hard [7]. To give an idea of its difficulty, we can note that, taken separately, each of its two subproblems is already an NP-hard problem [7, 13]. Moreover, from a practical viewpoint, its solving by exact methods is currently limited to instances with a dozen ports at most. Although recent, this problem has been the subject of many works, especially in the last ten years. Most of them are from operational research. Note that, in the literature, different variants of the LSNDP problem are studied according to the assumptions and properties taken into account (transit time, transshipment, constant or variable speed from one rotation to another or from one leg (i.e. a trip between two consecutive ports in the rotation) to another, type of service, possibility of refusing some commodities, ...).

Different approaches have been considered, often based on integer (mixed) linear programming (e.g. [18, 19, 21, 22, 23, 27, 30]). They are mainly based on two types of formulations. The first type of formulation is service-oriented. The set of possible services is calculated upstream and provided as input to the model. The latter is then limited to selecting the services to be kept among the candidates. The main disadvantage of this type of formulation is that the number of possible services grows exponentially with the number of ports, which limits its practical interest in the context of a solving performed with complete methods. On the other hand, it can be interesting in the context of incomplete methods, because one can then consider only a subset of the possible services. In practice, the solution proposed in [2] and based on a tabu method coupled with column generation allowed handling instances up to 120 ports. The second type of formulation is based on the selection of the arcs of the graph representing the possible links between each pair of ports. A service is then defined by the arcs that compose it, and the same arc can be used to define several services. From a practical viewpoint, such modeling coupled with complete methods [18, 23, 22] allows handling instances with up to a dozen ports [9].

Other approaches (e.g. [1, 2]) are based on two-step solving. Since the LSNDP problem consists of two subproblems, they process each subproblem separately. For example, the approaches presented in [2, 5, 13, 12] solve, in the first step, the problem of creating services and, in the second step, consider the vessel assignment and the management of the commodity flow based on the services found by the first step. In [17], the first phase is devoted to the management of the flow, while the second defines the services. Generally, the solving is done in several passes, the first phase then benefiting from some feedbacks from the second phase of the previous pass. Of course, this type of approach corresponds to incomplete methods. In practice, these approaches provide satisfactory results for instances with up to 120 ports [2]. They are often based on matheuristics (e.g. [5, 13, 12]) or the Variable Neighborhood Search algorithm (VNS) such as [17].

Beyond that, there are many related problems to the LSNDP problem. For example, the Vehicle Routing Problem (VRP [15]) and its variants have strong similarities with the LSNDP problem. In particular, the routes are circuits and, for some variants, the vehicle load or transit times can be taken into account. In maritime transport, the Liner Shipping Fleet Repositioning Problem (LSFRP [28]) consists in moving container ships from one service to another while taking into account the commodities to be transported, the empty containers to be relocated and maximizing the difference between the revenues and the costs generated. Among the approaches studied to solve this problem, we can underline the interest in using constraint programming (CP) put forward in [14].

While the VRP and LSFRP problems (and other shipping-related problems such as [16, 24]) have been studied from a CP perspective, this does not seem to be the case for the LSNDP problem. In this paper, we propose a model to handle a relatively general version of the LSNDP problem. Our model considers variable speeds from one trip to another and takes into account transshipments and transit times. Although developed in partnership with one of the world's leading container shipping companies, the model presented takes into account a relatively general version of the LSNDP problem. It can, of course, be adapted to specific needs, taking advantage of the flexibility of constraint programming. One of the aims of this work is to study the interest of a CP-based approach to modeling and solving such problems.

This paper is organized as follows. Section 2 introduces the notions necessary to understand the paper. Then, in Section 3, we propose a CP model for the LSNDP problem. Finally, we present some experimental results, in Section 4, before discussing related work in Section 5 and concluding in Section 6.

## 2 Preliminary

### 2.1 Constraint Programming

An instance  $P$  of the Constraint Optimization Problem (COP) can be defined as a 4-tuple  $(X, D, C, f)$  where  $X = \{x_1, \dots, x_n\}$  is the set of *variables*,  $D = \{D_{x_1}, \dots, D_{x_n}\}$  is the set of domains, the domain  $D_{x_i}$  being related to the variable  $x_i$ ,  $C = \{c_1, \dots, c_e\}$  represents the set of the *constraints* which define the interactions between the variables and describe the allowed combinations of values and  $f$  specifies the criterion to be optimized. Solving a COP instance  $P = (X, D, C, f)$  amounts to finding an assignment of all variables of  $X$  satisfying all constraints of  $C$  and optimizing the criterion given by  $f$ . This is an NP-hard problem.

One of the advantages of constraint programming lies in the existence of specialized constraints (the *global constraints*) which will make easier the modeling of problems, but also, their solving thanks to their dedicated filtering algorithms. In the following, we will exploit the following global constraints (where  $\odot$  denotes a relational operator among  $\leq$ ,  $<$ ,  $=$ ,  $\neq$ ,  $>$  or  $\geq$ ):

- **Alldiff-except** $(Y, v)$  [3, 10] which ensures that the values of the variables of  $Y$  are pairwise distinct, except in the case where they are equal to  $v$ ,
- **Circuit** $(Y, \ell)$  [4] which imposes that the values of the variables of  $Y$  form a circuit of length  $\ell$  (in the sense of graph theory), each variable  $y_i$  having for value  $i$  if it does not take part in the circuit, and  $j$  (with  $i \neq j$ ) if  $j$  is the successor of  $i$  in the circuit,
- **Count** $(Y, V) \odot k$  [4, 8] which ensures that the number of variables of  $Y$  whose value belongs to  $V$  satisfies the condition imposed by the relation  $\odot$  with respect to  $k$ ,
- **Elt** $(Y, i) = k$  [29] which ensures that the  $i$ th value of  $Y$  (using a 0-based indexing) is equal to  $k$  ( $Y$  can be here an ordered set of variables or values),

- $\text{Elt}_m(Y, i, j) = k$  [4, 3] which ensures the same property as  $\text{Elt}$ , but, for an ordered set  $Y$  of variables or values organized in the form of a two-dimensional matrix,
- $\text{Maximum}(Y) = k$  which ensures that the greatest value of  $Y$  is equal to  $k$  ( $Y$  can here be a set of variables or expressions),
- $\text{Sum}(Y, \Lambda) \odot k$  which imposes that the sum of the values of  $Y$  weighted by the coefficients of  $\Lambda$  satisfies the condition imposed by the relation  $\odot$  with respect to  $k$ . In the following, this constraint will be represented in the more explicit form  $\sum_i \lambda_i \cdot y_i \odot k$ .

## 2.2 The Liner Shipping Network Design Problem

Liner shipping involves the use of standardized vessels that will reliably move cargo between ports according to a pre-determined route and schedule. It is often compared to scheduled passenger service, such as a train or bus service because it operates on a fixed schedule and provides regular and predictable service for shippers and receivers of commodities. A shipping line, also called a service, is defined by a cyclic route (called a rotation) that serves a set of ports in a specific order and on a regular schedule. Figure 1 describes the example of a line connecting Asia and Europe.

In this paper, we consider only the transportation of commodities in their containerized form, as this mode of transportation constitutes the bulk of freight transportation in terms of quantity and value. Thus, a customer wishing to move commodities from a port of loading (POL) to a port of destination (POD) needs to place them inside one or more containers. Containers have the advantage, for the shipowner, that their dimensions are standardized, thus facilitating their handling and placement on board of specialized vessels such as container ships. There are mainly two sizes of containers: 20-foot containers (about 6.1 m) and 40-foot containers (12.2 m) with a height of 8.6 feet (2.6 m) and a width of 8 feet (2.4 m). The majority of containers transported are of one of these two sizes. Also, the storage space of the vessels is divided into 40-foot unit spaces on which it is possible to stack both 40-foot and 20-foot units. The Twenty-foot Equivalent Unit (TEU) is the unit generally used to count a number of containers. For example, a 40-foot container counts as 2 TEUs.

From the carrier's viewpoint, each commodity  $k$  is seen as a quantity  $q(k)$  of containers (expressed in TEUs) to be transported from the port of origin  $pol(k)$  to the port of destination  $pod(k)$  in exchange for a revenue  $rev(k)$  per TEU (expressed in dollars). This revenue may be zero in the case of empty containers. Some commodities may have a maximal transit time  $tt_{max}(k)$  that must be respected. This time is the maximum time allowed for their transport. Generally, such commodities are transported within the framework of premium offers proposed by the carriers. It should be noted that a batch of containers sent by a customer from one port to another cannot be divided into several sub-batches. Finally, cargo can be transported from its port of origin to its port of destination via the successive use of different lines. The operation of unloading a commodity from one line and loading it on another is called “*transshipment*”. It may require the commodities to be stored for several days in the transshipment port until the vessel of the next line arrives and loads them on board. This can result in costs (see transshipment costs below) and longer travel times.

Concerning vessels, container ships are grouped by type of vessel with identical or similar features. Thus, each class  $v$  is characterized by its *capacity*  $\kappa(v)$  (i.e. the maximum number of containers (in TEU) that can be transported), its *daily charter rate*  $tc(v)$  (corresponding to the daily cost of using the vessel), its *interval of possible speeds*  $[\nu_{min}(v), \nu_{max}(v)]$  (in knots), its *hourly consumption*  $cons(v, \nu)$  of fuel for the main engine (in tons per hour), for each *type of fuel*  $fuel(v)$  and each possible speed  $\nu$ . Regarding consumption, other parameters

that could have an impact such as wind strength, sea currents, draft or load on board are ignored. These parameters can be variable in time and difficult to anticipate, as the lines are generally defined on a yearly scale.

Each port  $p$  also has its own features, namely its *productivity*  $prod(p, v)$  (i.e., the number of containers loaded or unloaded per hour for vessels of type  $v$ ), its waiting time  $wt(p, v)$  (time at anchor before entering the port), its maneuvering time  $man^{in}(p, v)$  and  $man^{out}(p, v)$  to enter and leave the port respectively, its call charges  $pc(p, v)$  (in dollars), and its transshipment cost  $ts(p)$  (in dollars). The times are given in hours and depend on the type  $v$  of vessels, as do the call charges. A canal  $c$  (e.g. the Suez or Panama Canal) is characterized by a waiting time  $wt(c, v)$ , a traversal time  $trav(c)$  (in hours), and a traversal cost  $pc(c, v)$  (in dollars).

The number of vessels operating on a service is determined by the length of the rotation and the frequency of departures. Indeed, a rotation must guarantee a regular frequency of visits to each port it serves. This rotation frequency is generally weekly or biweekly. For a weekly frequency, the duration of the rotation must be a multiple of seven days. The number of vessels deployed per rotation must then be equal to the number of weeks in that rotation. For example, the line shown in Figure 1 has a duration of 91 days or 13 weeks. It is therefore operated with 13 vessels.

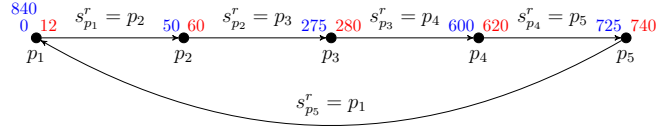
The liner shipping network design problem (LSNDP) can be defined as follows: Given a set of *ports*, a set of *vessels* divided by type (each type  $v$  having  $nb(v)$  vessels) and a set of *commodities* to be transported, define a set of rotations having a weekly frequency and determine the vessels operating them to transport the commodities while respecting, if necessary, the transit times and maximizing the profit. The profit is defined as the difference between the revenues generated by the commodities transported and all the costs generated by this transport (fuel costs, vessel operating costs, port call and canal costs, transshipment costs, ...). To calculate fuel costs, for each type of fuel  $f$ , we have the price  $fp(f)$  per ton of fuel (expressed in dollars).

While the primary purpose of this problem is to design maritime transportation networks, it can also be used to assist in decision-making. For example, it can be used to simulate situations such as traffic jams to enter certain ports and determine whether or not it is relevant to adapt existing rotations. It can also be used to consider changes in the flow of containers to be transported, to evaluate the interest in taking market share in certain commodity flows or to anticipate the construction of new ships.

## 3 Model

### 3.1 Modeling Choices

In our model, we adopt the usual assumptions of the literature. In particular, we assume that all container ships of a given type have identical features and that the frequency of services is weekly. Furthermore, we choose to treat canals (such as the Suez and Panama canals) in the same way as ports. The time it takes to cross a canal replaces the time it takes to load/unload a ship in a port. As a result, the notion of rotation now also takes canals into account. Since a rotation can pass several times through the same canal, but not through the same port, we consider, in our model, two instances of each canal so that a canal can be used both on the “outbound” and on the “return”. For example, we can see that the line represented in Figure 1 passes twice through the Suez Canal, once on the “outbound” (blue route) and once on the “return” (red route). Note that creating more than two instances of the same canal is of little interest because a solution passing more than twice through the same canal has little chance of being optimal.



■ **Figure 2** A rotation involving five ports.

$$\text{Circuit}(\{s_p^r \mid p \in \mathcal{P} \cup \mathcal{C}\}, l_r) \quad r \in \mathcal{R} \quad (\text{R.1})$$

$$v_r > 0 \Rightarrow \sum_{p \in \mathcal{P}} (s_p^r \neq p) \geq 3 \quad r \in \mathcal{R} \quad (\text{R.2})$$

$$v_r > 0 \iff l_r \geq 3 \quad r \in \mathcal{R} \quad (\text{R.3})$$

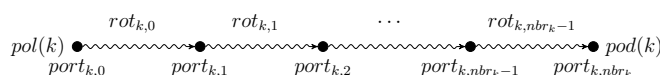
■ **Figure 3** Constraints related to rotations and routes.

Our model takes, as inputs, all the information about the ports and canals, the flow of commodities, the types of vessels, and the distances between pairs of ports/canals. It also relies on the maximum number  $r_{max}$  of rotations to be defined, the time horizon  $h_{max}$  (in hours), that is the maximum duration to achieve a rotation, and the maximum number  $ts_{max}$  of transshipments allowed per commodity. One of the particularities of this model is that the main operations will be time-stamped to handle rotation or transit times of the commodities as precisely as possible. Moreover, speeds can be different from one leg to another. Finally, it takes into account the possibility of refusing to transport a commodity in the network if it is not profitable or impossible.

Thereafter, given the large number of variables, we define the variables progressively when needed. The set of ports is denoted  $\mathcal{P} = \{0, 1, \dots, |\mathcal{P}| - 1\}$ , the set of canals  $\mathcal{C} = \{c, c + |\mathcal{C}_0| \text{ s.t. } c \in \mathcal{C}_0\}$  (with  $\mathcal{C}_0 = \{|\mathcal{P}|, \dots, |\mathcal{P}| + |\mathcal{C}_0| - 1\}$  the set of canals before duplication), the set of type of vessels available  $\mathcal{V} = \{1, 2, \dots, |\mathcal{V}|\}$  and that of the commodities  $\mathcal{K} = \{1, 2, \dots, |\mathcal{K}|\}$ . We note respectively  $\mathcal{I}$  and  $\mathcal{I}^+$  the index sets  $[0, ts_{max}]$  and  $[0, ts_{max} + 1]$ . Let  $\mathcal{R} = \{1, \dots, r_{max}\}$  be the set of the indices of the possible rotations.

### 3.2 Definition of Rotations

Our model does not necessarily use all possible  $r_{max}$  rotations. Therefore, we consider a variable  $v_r$  per rotation  $r$ . Its value is an integer between 1 and  $|\mathcal{V}|$  representing the type of vessels exploited if the rotation is used, 0 otherwise. Each rotation  $r$  must correspond to a circuit. To define such circuits, we introduce a variable  $s_p^r$  per port/canal  $p$  and rotation  $r$ . Its value is  $p$  if the port/canal  $p$  is not involved in the circuit, the successor of port/canal  $p$  otherwise. Figure 2 illustrates this for a rotation involving five ports. We also introduce a variable  $l_r$  specifying the length of the circuit associated with rotation  $r$ . Thus, for each rotation  $r$ , the existence of a circuit can be guaranteed by constraint (R.1) (see Figure 3). Note that this constraint avoids the existence of subtours, a property that is not generally easy to guarantee. For instance, in MIP, avoiding subtours requires adding non-linear constraints that must then be linearized. Then, thanks to constraint (R.2), a circuit must involve at least three ports (this is a business rule generally desired by carriers), and so, cannot involve only canals. Finally, constraint (R.3) ensures that rotation  $r$  is used if and only if the associated circuit has a length at least equal to three.



■ **Figure 4** Transport of a commodity  $k$  from its origin port  $pol(k)$  to its destination port  $pod(k)$ .

### 3.3 Cargo Flow

Our model allows for the possibility of not carrying a commodity  $k$  if it is not possible or not profitable. To do this, we define a Boolean variable  $\alpha_k$  that is true if commodity  $k$  is accepted in the network, false otherwise. Taking charge of commodity  $k$  means that it is loaded at the original port  $pol(k)$  and unloaded at the destination one  $pod(k)$ , possibly passing through intermediate ports. In our model, we consider only the intermediate ports where the commodity will be transhipped as shown in Figure 4. Also, we introduce a variable  $rot_{k,i}$  per commodity  $k$  and step  $i$  to represent the  $i$ th rotation used to transport the commodity  $k$ .  $rot_{k,i}$  has the value  $r$  if commodity  $k$  is carried thanks to rotation  $r$  during the  $i$ th step,  $-1$  if this step is not needed. Similarly, the variable  $port_{k,i}$  represents the port where the commodity  $k$  enters its  $i$ th rotation. By so doing, commodity  $k$  enters in its  $i$ th rotation at port  $port_{k,i}$  and leaves it at port  $port_{k,i+1}$  (i.e. the port in which it starts its next rotation if  $port_{k,i+1}$  differs from  $pod(k)$ ). These variables have the value of the corresponding port  $p$  ( $p \in \mathcal{P}$ ) if the  $i$ th rotation is used,  $-1$  otherwise. More precisely, the domain of  $port_{k,i}$  is  $\{-1, pol(k)\}$  if  $i = 0$ ,  $\{-1, pod(k)\}$  if  $i = ts_{max} + 1$ ,  $\{-1\} \cup \mathcal{P} - \{pol(k)\}$  otherwise. For each commodity  $k$ , a variable  $nbr_k$  specifies the number of rotations used (between 0 and  $ts_{max} + 1$ ).

We can now define the associated constraints (see Figure 5). First, constraint (F.1) specifies that commodity  $k$  is accepted in the network if and only if there is at least one rotation that carries it. Of course, the port of departure of an accepted commodity  $k$  is necessarily its port of origin  $pol(k)$  (constraint (F.2)). The last port used is necessarily the destination port  $pod(k)$ . To ensure this, we introduce a variable  $pod_k$  per commodity  $k$  that can take two values: either  $pod(k)$  if commodity  $k$  is accepted, or  $-1$  otherwise. Constraint (F.3) guarantees that  $pod_k$  has the relevant value depending on the value of  $\alpha_k$  while constraint (F.4) ensures that the last used port is consistent with  $pod_k$ . Naturally, no port or rotation can be used beyond the destination port (constraints (F.5)–(F.7)). Moreover, a commodity cannot transit several times through the same port or the same rotation, what is ensured by constraints (F.8) and (F.9). To facilitate the expression of some constraints about the path followed by the commodities, we introduce Boolean variables  $from_{k,p}^r$  (resp.  $to_{k,p}^r$ ) which are true if commodity  $k$  enters (resp. leaves) the rotation  $r$  at port  $p$ , false otherwise. These variables are directly related to the previous ones as seen in constraints (F.10) and (F.11). These variables are also used to link the cargo flow to the definition of routes. Indeed, if commodity  $k$  is (un)loaded at a port  $p$  for a rotation  $r$ , it implies that the port  $p$  is used in this rotation (constraints (F.12) and (F.13)). Conversely, if a port  $p$  is used in rotation  $r$ , then there is at least one commodity that is (un)loaded in that port for that rotation (see constraints (F.14) and (F.15)). Finally, if a rotation is not used, no commodities can transit through it, and vice versa (constraint (F.16)). Note that this constraint is redundant, but in most cases, it allows finding some conflicts earlier.

### 3.4 Properties of Rotations and Vessels

In some MIP models in the literature (e.g. [18, 9]), each service is associated with a predefined type of vessel. While this choice facilitates the consideration of the specificities of each type of vessel, it leads to handling many rotations, few of which will be used in the end. Moreover,



$$\alpha_k = 1 \iff nbr_k > 0 \quad k \in \mathcal{K} \quad (\text{F.1})$$

$$\alpha_k = 1 \iff port_{k,1} = pol(k) \quad k \in \mathcal{K} \quad (\text{F.2})$$

$$\alpha_k = 1 \iff pod_k = pod(k) \quad k \in \mathcal{K} \quad (\text{F.3})$$

$$\text{Elt}(\{port_{k,i+1} \mid i \in [1, ts_{max}]\}, nbr_k) = pod_k \quad k \in \mathcal{K} \quad (\text{F.4})$$

$$i \geq nbr_k \iff rot_{k,i} = -1 \quad k \in \mathcal{K}, i \in \mathcal{I} \quad (\text{F.5})$$

$$nbr_k < i \Rightarrow port_{k,i} = -1 \quad k \in \mathcal{K}, i \in \mathcal{I}^+ \quad (\text{F.6})$$

$$port_{k,i} = -1 \Rightarrow nbr_k \leq i \quad k \in \mathcal{K}, i \in \mathcal{I}^+ \quad (\text{F.7})$$

$$\text{Alldiff-exception}(\{port_{k,i} \mid i \in \mathcal{I}^+\}, -1) \quad k \in \mathcal{K} \quad (\text{F.8})$$

$$\text{Alldiff-exception}(\{rot_{k,i} \mid i \in \mathcal{I}\}, -1) \quad k \in \mathcal{K} \quad (\text{F.9})$$

$$from_{k,p}^r = \sum_{i \in \mathcal{I}} (port_{k,i} = p) \cdot (rot_{k,i} = r) \quad k \in \mathcal{K}, p \in \mathcal{P}, r \in \mathcal{R} \quad (\text{F.10})$$

$$to_{k,p}^r = \sum_{i \in \mathcal{I}} (port_{k,i+1} = p) \cdot (rot_{k,i} = r) \quad k \in \mathcal{K}, p \in \mathcal{P}, r \in \mathcal{R} \quad (\text{F.11})$$

$$from_{k,p}^r = 1 \Rightarrow s_p^r \neq p \quad k \in \mathcal{K}, p \in \mathcal{P}, r \in \mathcal{R} \quad (\text{F.12})$$

$$to_{k,p}^r = 1 \Rightarrow s_p^r \neq p \quad k \in \mathcal{K}, p \in \mathcal{P}, r \in \mathcal{R} \quad (\text{F.13})$$

$$s_p^r \neq p \Rightarrow \text{Count}(\{port_{k,i} \mid k \in \mathcal{K}, i \in \mathcal{I}^+\}, \{p\}) \geq 1 \quad p \in \mathcal{P}, r \in \mathcal{R} \quad (\text{F.14})$$

$$s_p^r \neq p \Rightarrow \text{Count}(\{rot_{k,i} \mid k \in \mathcal{K}, i \in \mathcal{I}\}, \{r\}) \geq 1 \quad p \in \mathcal{P}, r \in \mathcal{R} \quad (\text{F.15})$$

$$v_r = 0 \iff \text{Count}(\{rot_{k,i} \mid k \in \mathcal{K}, i \in \mathcal{I}^+\}, \{r\}) = 0 \quad r \in \mathcal{R} \quad (\text{F.16})$$

■ **Figure 5** Constraints related to cargo flow.

trying a new vessel type for a rotation requires the solver to change the assignment of a lot of variables. In our model, we choose to let the solver decide on the type of vessels associated with each rotation. Therefore, it is necessary to ensure that the type of vessels chosen for a rotation matches the features of the rotation. This requires the introduction of a certain number of variables whose values will then be fixed using `Elt` constraints. The variable  $\kappa_r$  represents the maximum capacity (expressed in TEUs) of commodities that can be transported via rotation  $r$ . Its value is 0 if the rotation is not used, the capacity of the type of vessel used otherwise. The variables  $\nu_{min}^r$  and  $\nu_{max}^r$  specify the minimum and maximum speeds of the rotation  $r$  (0 if the rotation is not used). The variable  $fp_r$  expresses the price per ton of fuel for rotation  $r$  (0 if the rotation is not operated). For each rotation  $r$ , we post the constraints (P.1)–(P.4) (see Figure 6). Likewise, some information (call costs, waiting time, ...) related to ports or canals also depends on the type of vessels associated with rotation  $r$ . For each rotation  $r$ , we then introduce the variables  $wt_{p,r}$ ,  $man_{p,r}^{in}$  and  $man_{p,r}^{out}$  which specify respectively the waiting time of port/canal  $p$  and the maneuvering time to enter and leave port  $p$ . The cost of calling at port  $p$  for rotation  $r$  is represented by the variable  $pc_{p,r}$  while the productivity for port  $p$  and rotation  $r$  is expressed by the variable  $prod_{p,r}$ . Constraints (P.5)–(P.9) ensure the consistency of these features. Finally, we consider the variable  $tc_r$  which, for each rotation  $r$ , specifies the daily cost of using the vessels associated with the rotation and its related constraint (P.10).

$$\begin{aligned} \text{Elt}(\{0\} \cup \{\kappa(v) \mid v \in \mathcal{V}\}, v_r) &= \kappa_r & r \in \mathcal{R} & \quad (\text{P.1}) \\ \text{Elt}(\{0\} \cup \{\nu_{\min}(v) \mid v \in \mathcal{V}\}, v_r) &= \nu_{\min}^r & r \in \mathcal{R} & \quad (\text{P.2}) \\ \text{Elt}(\{0\} \cup \{\nu_{\max}(v) \mid v \in \mathcal{V}\}, v_r) &= \nu_{\max}^r & r \in \mathcal{R} & \quad (\text{P.3}) \\ \text{Elt}(\{0\} \cup \{fp(\text{fuel}(v)) \mid v \in \mathcal{V}\}, v_r) &= fp_r & r \in \mathcal{R} & \quad (\text{P.4}) \\ \text{Elt}(\{0\} \cup \{wt(p, v) \mid v \in \mathcal{V}\}, v_r) &= wt_{p,r} & r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} & \quad (\text{P.5}) \\ \text{Elt}(\{0\} \cup \{man^{in}(p, v) \mid v \in \mathcal{V}\}, v_r) &= man_{p,r}^{in} & r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} & \quad (\text{P.6}) \\ \text{Elt}(\{0\} \cup \{man^{out}(p, v) \mid v \in \mathcal{V}\}, v_r) &= man_{p,r}^{out} & r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} & \quad (\text{P.7}) \\ \text{Elt}(\{0\} \cup \{pc(p, v) \mid v \in \mathcal{V}\}, v_r) &= pc_{p,r} & r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} & \quad (\text{P.8}) \\ \text{Elt}(\{1\} \cup \{prod(p, v) \mid v \in \mathcal{V}\}, v_r) &= prod_{p,r} & r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} & \quad (\text{P.9}) \\ \text{Elt}(\{0\} \cup \{tc(v) \mid v \in \mathcal{V}\}, v_r) &= tc_r & r \in \mathcal{R} & \quad (\text{P.10}) \end{aligned}$$

■ **Figure 6** Constraint related to properties of rotations and vessels.

$$\begin{aligned} from_{k,p}^r = 1 &\Rightarrow leave_{k,p}^r = 1 & k \in \mathcal{K}, r \in \mathcal{R}, p \in \mathcal{P} & \quad (\text{L.1}) \\ to_{k,p}^r = 1 &\Rightarrow leave_{k,p}^r = 0 & k \in \mathcal{K}, r \in \mathcal{R}, p \in \mathcal{P} & \quad (\text{L.2}) \\ (s_p^r = p' \wedge from_{k,p'}^r = 0 \wedge to_{k,p'}^r = 0) &\Rightarrow leave_{k,p}^r = leave_{k,p'}^r & k \in \mathcal{K}, r \in \mathcal{R}, p, p' \in \mathcal{P} & \quad (\text{L.3}) \\ s_p^r = p &\Rightarrow leave_{k,p}^r = 0 & k \in \mathcal{K}, r \in \mathcal{R}, p \in \mathcal{P} & \quad (\text{L.4}) \\ \sum_{k \in \mathcal{K}} q(k) \cdot leave_{k,p}^r &\leq \kappa_r & r \in \mathcal{R}, p \in \mathcal{P} & \quad (\text{L.5}) \end{aligned}$$

■ **Figure 7** Constraints related to loads.

### 3.5 Vessel Load

We need to ensure that vessels do not leave each port loaded beyond their maximum capacity. This requires knowing, for each rotation, which commodities it carries at the exit of each port. To do this, we use a Boolean variable  $leave_{k,p}^r$  per commodity  $k$ , port  $p$ , and rotation  $r$ . This variable is true if commodity  $k$  leaves port  $p$  via rotation  $r$ , false otherwise. Constraints (L.1) and (L.2) (see Figure 7) deal with the cases when the commodities are respectively loaded in rotation  $r$  and unloaded from rotation  $r$  while constraint (L.3) guarantees the transitivity all along the trip. Finally, the constraint (L.4) corresponds to the case where a port  $p$  does not appear in rotation  $r$ . Constraint (L.5) then allows ensuring that, for each port  $p$ , the load, when leaving the port, does not exceed the maximum capacity  $\kappa_r$  of rotation  $r$ .

### 3.6 Timestamp and Transit Times

#### 3.6.1 Duration of Port Operations and Canal Traversal

To express the duration of loading/unloading operations in a port or the duration of traversing a canal, we introduce a variable  $t_p^r$  per port/canal  $p$  and rotation  $r$ . In the case of a canal the value of this variable is defined as equal to the duration of the traversal if the canal is used, 0 otherwise (see constraint (T.1) in Figure 8). For a port  $p$ , two cases are possible. If the port

$$t_p^r = trav(p) \cdot (s_p^r \neq p) \quad r \in \mathcal{R}, p \in \mathcal{C} \quad (\text{T.1})$$

$$s_p^r = p \iff t_p^r = 0 \quad r \in \mathcal{R}, p \in \mathcal{P} \quad (\text{T.2})$$

$$teu_{p,r} = \sum_{k \in \mathcal{K}} (from_{k,p}^r + to_{k,p}^r) \cdot q(k) \quad r \in \mathcal{R}, p \in \mathcal{P} \quad (\text{T.3})$$

$$s_p^r \neq p \Rightarrow t_p^r = \left\lceil \frac{\mu \cdot teu_{p,r}}{prod_{p,r}} \right\rceil \quad r \in \mathcal{R}, p \in \mathcal{P} \quad (\text{T.4})$$

$$v_r = 0 \iff dep_r = -1 \quad r \in \mathcal{R} \quad (\text{T.5})$$

$$v_r > 0 \Rightarrow \text{Maximum}(\{p \cdot (s_p^r \neq p) \mid p \in \mathcal{P}\}) = dep_r \quad r \in \mathcal{R} \quad (\text{T.6})$$

$$dep_r = p \Rightarrow time_{p,r}^{in} = 0 \quad r \in \mathcal{R}, p \in \mathcal{P} \quad (\text{T.7})$$

$$time_{p,r}^{out} = time_{p,r}^{in} + t_p^r \quad r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} \quad (\text{T.8})$$

$$s_{p'}^r = p' \Rightarrow st_p^r = \left\lceil \frac{\delta(p,p')}{\nu_p^r} \right\rceil \quad r \in \mathcal{R}, p, p' \in \mathcal{P} \cup \mathcal{C} \quad (\text{T.9})$$

$$(v_r > 0 \wedge s_p^r \neq p) \Rightarrow \nu_p^r \geq \nu_{min}^r \quad r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} \quad (\text{T.10})$$

$$v_r > 0 \Rightarrow \nu_p^r \leq \nu_{max}^r \quad r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} \quad (\text{T.11})$$

$$s_p^r = p \iff st_p^r = 0 \quad r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} \quad (\text{T.12})$$

$$s_p^r = p \Rightarrow time_{p,r}^{in} = 0 \quad r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} \quad (\text{T.13})$$

$$s_p^r = p \iff \nu_p^r = 0 \quad r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} \quad (\text{T.14})$$

$$(s_p^r = p' \wedge p' \neq dep_r) \Rightarrow time_{p',r}^{in} = time_{p,r}^{out} + man_{p,r}^{out} + st_p^r + wt_{p',r} + man_{p',r}^{in} \quad r \in \mathcal{R}, p, p' \in \mathcal{P} \cup \mathcal{C} \quad (\text{T.15})$$

$$(from_{k,p}^r \wedge to_{k,p'}^r) \Rightarrow (time_{p,r}^{in} < time_{p',r}^{in} \vee time_{p',r}^{in} < time_{p,r}^{in} < time_{p',r}^{in} < time_{p',r}^{in} + T_r) \quad r \in \mathcal{R}, p, p' \in \mathcal{P} \cup \mathcal{C}, k \in \mathcal{K} \quad (\text{T.16})$$

■ **Figure 8** Constraints related to timestamps.

is not used in the rotation  $r$ , the variable  $t_p^r$  is 0 (see constraint (T.2)). Otherwise, its value depends on the number of TEUs loaded and unloaded in the port  $p$  for the rotation  $r$ . So, we consider the variable  $teu_{p,r}$  which indicates the number of TEUs loaded and unloaded at port  $p$  for rotation  $r$ . Its value can be defined thanks to constraint (T.3). We can express the duration of the operations thanks to constraint (T.4). A crane movement allows moving a container whatever its size. To take into account the existence of 20-foot and 40-foot containers among the commodities to be handled, the parameter  $\mu$  makes it possible to calculate the number of containers to be handled and thus the number of crane movements necessary from the number of containers expressed in TEUs.

### 3.6.2 Call Timestamps

In order to establish the schedule for each call, it is necessary to designate a port as the departure port in each rotation. To do this, we consider a variable  $dep_r$  per rotation  $r$  which has, for value, a port  $p$  if the rotation is used,  $-1$  otherwise. The choice of the starting port being purely arbitrary, we choose the one with the largest index. This can be achieved thanks to constraints (T.5) and (T.6). Then, in our model, we consider two key moments: the moment when the vessel arrives at the berth (resp. enters the canal) and the moment when it leaves the berth (resp. leaves the canal). For each rotation  $r$  and each port/canal  $p$ , these two moments are represented respectively by the variables  $time_{p,r}^{in}$  and  $time_{p,r}^{out}$  which take their values in  $[0, h_{max}]$ . For each rotation, we consider that time 0 coincides with the time of arrival at the berth in the departure port thanks to constraint (T.7). The time

of leaving a port or a canal depends only on the time of arrival and the duration of the operations in the port or the traversal of the canal (constraint (T.8)). Then, to determine the arrival time at a port or canal as a function of the exit time from the port/canal ahead of it in the rotation, we need to define the travel time as a function of the vessel's speed. The variables  $st_p^r$  and  $\nu_p^r$  represent respectively the travel time from the port/canal  $p$  to its successor (if any) in rotation  $r$  and the speed (expressed in knots) used on this leg. The two variables are correlated by constraint (T.9). Of course, the speeds used must be consistent with the capabilities of the vessels operating the rotation (constraints (T.10) and (T.11)). We can now define the time of arrival at the port/canal  $p'$  from its predecessor  $p$  in rotation  $r$  thanks to constraint (T.15). Note that for canals, we consider that the variables  $man_{p,r}^{in}$  and  $man_{p,r}^{in}$  are 0. This allows us to avoid defining the previous constraint according to the different possibilities of port/canal successions. In the case where a port  $p$  is not operated in a rotation  $r$ , we set the values of the variables  $st_p^r$ ,  $\nu_p^r$  and  $time_{p,r}^{in}$  to the value 0 (constraints (T.12)–(T.14)).

Finally, if a commodity  $k$  is loaded in rotation  $r$  at port  $p$  and unloaded at port  $p'$ , this imposes that the arrival at port  $p$  takes place before the arrival at port  $p'$  if the trip between  $p$  and  $p'$  does not pass through the departure port of rotation  $r$ . If this path passes through the departure port, then the arrival at port  $p'$  will occur in the next rotation, and the arrival at port  $p$  is between the two visits to port  $p'$ . For example, if we consider the rotation in Figure 2 (which lasts 840 hours) and a commodity sent from port  $p_2$  to port  $p_4$ , a vessel operating this rotation enters port  $p_2$  at hour 50 (in blue) and arrives in port  $p_4$  at hour 600. In this case, we have  $time_{p_2,r}^{in} < time_{p_4,r}^{in}$ . On the other hand, if we consider a commodity going from  $p_4$  to  $p_3$ , the vessel enters port  $p_3$  at hour 275 before visiting  $p_4$ . This commodity will then be delivered only at the next passage of the vessel at time 1,115. We then have  $time_{p_3,r}^{in} < time_{p_4,r}^{in} < time_{p_3,r}^{in} + T_r$ . This is ensured by constraint (T.16).

### 3.6.3 Transit Times

To accurately consider the transit time of commodities, we need to know the key moments in their transportation, namely when they are loaded on board a rotation or unloaded. To simplify the model, we consider that a commodity is loaded on board a rotation when the rotation leaves the port and that it is unloaded when the rotation arrives at the port. These two times are represented by the variables  $ctime_{i,k}^{in}$  and  $ctime_{i,k}^{out}$  respectively. Constraints (T.17) and (T.18) (see Figure 9) ensure the correspondence between the key times of the rotations and the ones of the commodities.

The time spent by the commodity  $k$  in its  $i$ th rotation is represented by the variable  $\delta_{i,k}$ . It corresponds naturally to the difference between the exit time and the entry time. However, we must take into account the particular case where the journey passes through the port of departure. In this case, the commodities are unloaded at the next rotation. For example, if we consider the previous example, a commodity sent from port  $p_2$  to port  $p_4$  leaves port  $p_2$  at hour 60 (in red) and arrives in port  $p_4$  at hour 600 (in blue). This gives a travel time of 540 hours. On the other hand, a commodity shipped from port  $p_4$  to port  $p_3$  leaves port  $p_4$  at time 620 and arrives at port  $p_3$  at time 1,115 and thus takes 495 hours to reach its destination. Constraints (T.19) and (T.20) deal respectively with the first and second cases.

In the case where a transshipment takes place, the time that the commodities spend on the quay between the two rotations must be taken into account. Given the weekly frequency of the rotations, this time can be of the order of a week at most. To consider it more precisely, we introduce a variable  $\Delta_{i,k}$  per commodity  $k$  and  $i$ th rotation used. The value of this variable is related to the weekly frequency of rotations. For example, consider a commodity

$$\text{Elt}_m(\{time_{p,r}^{out} \mid p \in \mathcal{P}, r \in R\}, port_{k,i}, rot_{k,i}) = ctime_{i,k}^{in} \quad k \in \mathcal{K}, i \in \mathcal{I} \quad (\text{T.17})$$

$$\text{Elt}_m(\{time_{p,r}^{in} \mid p \in \mathcal{P}, r \in R\}, port_{k,i+1}, rot_{k,i}) = ctime_{i,k}^{out} \quad k \in \mathcal{K}, i \in \mathcal{I} \quad (\text{T.18})$$

$$ctime_{i,k}^{in} \leq ctime_{i,k}^{out} \Rightarrow \delta_{i,k} = ctime_{i,k}^{out} - ctime_{i,k}^{in} \quad k \in \mathcal{K}, i \in \mathcal{I}^+ \quad (\text{T.19})$$

$$(rot_{k,i} = r \wedge ctime_{i,k}^{in} > ctime_{i,k}^{out}) \Rightarrow \delta_{i,k} = ctime_{i,k}^{out} - ctime_{i,k}^{in} + T_r \quad k \in \mathcal{K}, i \in \mathcal{I}^+ \quad (\text{T.20})$$

$$i + 1 < nbr_k \Rightarrow \Delta_{i,k} = (ctime_{i+1,k}^{in} - ctime_{i,k}^{out}) \% 168 \quad k \in \mathcal{K}, i \in \mathcal{I} \quad (\text{T.21})$$

$$i + 1 \geq nbr_k \Rightarrow \Delta_{i,k} = 0 \quad k \in \mathcal{K}, i \in \mathcal{I} \quad (\text{T.22})$$

$$\sum_{i \in \mathcal{I}^+} \delta_{i,k} + \sum_{i \in \mathcal{I}} \Delta_{i,k} \leq tt_{max}(k) \quad k \in \mathcal{K} \quad (\text{T.23})$$

■ **Figure 9** Constraints related to transit times.

$k$  that arrives at a port  $p$  at hour 200 (according to its  $ctime_{i,k}^{out}$  value) on a rotation  $r$  and leaves it at hour 2,000 (according to its  $ctime_{i+1,k}^{in}$  value) via a rotation  $r'$ . The weekly frequency of the rotations  $r$  and  $r'$  implies that, in practice, commodity  $k$  leaves the port at hour 320. Indeed, a vessel of rotation  $r'$  leaves the port at hours 152 (i.e. 2,000 modulo  $(7 \times 24)$ ), 320, 488, ... If the arrival in the port is later than the departure from the port (according to the values  $ctime_{i,k}^{out}$  and  $ctime_{i+1,k}^{in}$ ), it means that the commodity must wait for the vessel of the next week and so we have to add 168 hours to the considered difference. Constraint (T.21) takes into account these two cases. The  $i$ th rotations that are not used for commodity  $k$  have a  $\Delta_{i,k}$  variable whose value is zero (constraint (T.22)). The transit time of commodity  $k$  can then be guaranteed by constraint (T.23). Note that not all commodities have a transit time constraint. Also, if commodity  $k$  does not have a maximum transit time constraint, the variables and constraints presented here are not considered for it.

### 3.7 Vessel Availability

Since the frequency of the rotations is weekly, each port is visited by one vessel operating the rotation each week. The number of vessels needed is therefore the total time of the rotation divided by the duration of one week. If the variables  $n_r$  and  $T_r$  represent respectively the number of vessels used by the rotation  $r$  and the total time of the rotation  $r$ , we can impose constraint (A.1) (see Figure 10). The time of the rotation is, of course, zero if the rotation is not used (constraint (A.2)). Otherwise, since each rotation starts at time 0, the total time is the arrival time at the departure port from the last port of the rotation (constraint (A.3)). Finally, we guarantee that, for each type of vessel, the number of vessels used does not exceed the number of available vessels thanks to constraint (A.4).

### 3.8 Objective Function

Briefly, the objective function is the difference between the revenues generated by accepting commodities into the network and the total costs of transporting them (fuel, vessel operations, port calls, ...). The fuel cost depends on the fuel price and the fuel consumption of each trip made. For this latter, we consider a variable  $cons_p^r$  per port and rotation that specifies the amount of fuel consumed per hour by the rotation  $r$  for the trip made between the port/canal  $p$  and its successor. In the absence of successors, the variable  $cons_p^r$  has, of course, the value

$$n_r = \left\lceil \frac{T_r}{7 \times 24} \right\rceil \quad r \in \mathcal{R} \quad (\text{A.1})$$

$$v_r = 0 \iff T_r = 0 \quad r \in \mathcal{R} \quad (\text{A.2})$$

$$dep_r = p' \wedge s_p^r = p' \Rightarrow T_r = time_{p,r}^{out} + man_{p,r}^{out} + st_p^r + wt_{p',r} + man_{p',r}^{in} \quad r \in \mathcal{R}, p, p' \in \mathcal{P} \quad (\text{A.3})$$

$$\sum_{r \in \mathcal{R}} n_r \cdot (v_r = v) \leq nb(v) \quad v \in \mathcal{V} \quad (\text{A.4})$$

■ **Figure 10** Constraints related to vessel availability.

$$\text{El}_m(\{cons(v, \nu) \mid v \in \{0\} \cup \mathcal{V}, \nu \in \{0\} \cup [\nu_{min}(v), \nu_{max}(v)]\}, v_r, \nu_p^r) = cons_p^r \quad r \in \mathcal{R}, p \in \mathcal{P} \cup \mathcal{C} \quad (\text{O.1})$$

$$teu_p^{ts} = \sum_{k \in \mathcal{K} \mid p \neq pod(k)} to_{k,p}^r \cdot q(k) \quad p \in \mathcal{P} \quad (\text{O.2})$$

■ **Figure 11** Constraints related to the objective function.

0. The quantity consumed here depends only on the type of vessel used and the speed. In constraint (O.1) (see Figure 11), we assume that  $cons(v, \nu)$  is 0 if  $v$  or  $\nu$  is 0.

The costs associated with transshipment depend on the port and the quantity of commodities transshipped. So we need to represent the quantity of commodities transshipped at each port. To do this, we introduce a variable  $teu_p^{ts}$  per port. The commodities  $k$  transshipped at port  $p$  are those that are unloaded at port  $p$  (i.e., those for which  $to_{k,p}^r$  is 1) and for which port  $p$  is not their destination port (see constraint (O.2)). We can now express our objective function based on revenues (R), fuel costs (C), canal and port call costs (E), vessel operating costs (X), and transshipment costs (T):

$$\begin{aligned} \max \quad & \sum_{k \in \mathcal{K}} rev(k) \cdot q(k) \cdot \alpha_k & (R) \\ - \quad & \sum_{r \in \mathcal{R}} \sum_{p \in \mathcal{P} \cup \mathcal{C}} fp_r \cdot cons_p^r \cdot st_p^r & (C) \\ - \quad & \sum_{r \in \mathcal{R}} \sum_{p \in \mathcal{P} \cup \mathcal{C}} pc_{pr} \cdot (s_p^r \neq p) & (E) \\ - \quad & 7 \sum_{r \in \mathcal{R}} tc_r \cdot n_r & (X) \\ - \quad & \sum_{p \in \mathcal{P}} ts(p) \cdot teu_p^{ts} & (T) \end{aligned}$$

### 3.9 Additional Constraints

Given the size of the search space, it may be desirable to avoid certain symmetries as much as possible. Starting each rotation at time 0 (see constraint (T.7)) allows for avoiding any translation on the time axis. However, other symmetries may exist. For example, the rotations are interchangeable. To avoid this, we can ensure that the first rotations are used in priority and these rotations are sorted in decreasing order of their duration thanks to constraint (S.1) (see Figure 12). One of the main practical difficulties of the LSNDP problem lies in the enumeration of the different possible circuits. For a circuit of length

$$T_1 \geq T_2 \geq \dots \geq T_r \quad (\text{S.1})$$

$$\sum_{r \in \mathcal{R}} (s_p^r \neq p) \leq \max(|\mathcal{K}_p^{pol}|, |\mathcal{K}_p^{pod}|) \quad p \in \mathcal{P}_2 \quad (\text{S.2})$$

$$v_r = v \Rightarrow s_p^r = p \quad r \in \mathcal{R}, p \in \mathcal{P}_3 \quad (\text{S.3})$$

$$teu_p^{ts} = 0 \quad p \in \mathcal{P}_4 \quad (\text{S.4})$$

■ **Figure 12** Possible additional constraints where  $\mathcal{P}_i$  denotes the set of ports impacted by the constraint (S.i).

$\ell$ , since it is possible to go from one port to any other, the solver may have to consider a non-negligible part of the  $\ell!$  possible permutations. Since, in addition, several rotations are usually considered simultaneously, this can quickly become very time-consuming. To reduce the number of rotations and thus of circuits to consider, we introduce constraint (S.2). Let  $\mathcal{K}_p^{pol}$  and  $\mathcal{K}_p^{pod}$  denote respectively the set of commodities for which  $p$  is the origin port and one for which  $p$  is the destination port. This constraint ensures that the number of rotations that uses a port  $p$  does not exceed the maximum between  $|\mathcal{K}_p^{pol}|$  and  $|\mathcal{K}_p^{pod}|$ . In a way, it eliminates some solutions in which the call in the port would only be used for transshipments. Generally, calling a port for only transshipments is not wished by shipping companies, except for some particular ports (e.g. hubs). Thanks to the flexibility of CP, we can add this constraint depending on the needs of the shipowner.

On the other hand, some ports cannot handle certain types of vessels. For example, the port of Dutch Harbor in Alaska is not deep enough. It can therefore only handle small container ships. Thus, if  $v$  vessels cannot berth at port  $p$ , we impose constraint (S.3) for each rotation  $r$ . Similarly, some ports do not have enough space to store containers. It is therefore impossible to carry out transshipments there. For such ports, we can then exploit constraint (S.4) to prohibit any transshipment.

## 4 Experiments

### 4.1 Experimental Protocol and Implementation Details

The LINER-LIB benchmark<sup>1</sup> [7] is the reference for experiments on the LSNDP problem. It consists of seven instances with 12 to 197 ports, thus allowing the evaluation of both complete and incomplete methods. In order to have instances of a reasonable and varied size, we have produced sub-instances from instances of the LINER-LIB benchmark. To do this, from an instance, we select  $n$  ports in the following way. The first selected port is the one that handles the most commodities. The next  $n - 1$  ports are the ones that exchange the most commodities with the already selected ports. For our test set, we considered the smallest instance (**Baltic**) of the LINER-LIB set and 40 instances produced from the instances **Baltic**, **EuropeAsia**, **Mediterranean** and **WAF**. The number of ports varies from 3 to respectively 11, 10, 10 and 17. Moreover, as the LINER-LIB benchmark does not take into account productivity, waiting or maneuvering times, we generate randomly these values. Note that this partial random generation introduces no bias, since these values have a negligible impact on the solving efficiency. For the following experiments, the number of

<sup>1</sup> <https://github.com/blof/LINERLIB/>

rotations  $r_{max}$  is fixed at 4 and that of transshipments at 1 per commodity (a higher value not being desired by the experts) while the maximum duration  $h_{max}$  is set to 12 weeks. The chosen values of  $r_{max}$  and  $h_{max}$  seem reasonable to us taking into account the commodities to be transported and the distances to be covered. In practice, the optimal solutions we found require less time and fewer rotations than our choice of parameters allows. For  $\mu$ , we take the value 0.54 given by the experts. The instances we consider are available at <https://pageperso.lis-lab.fr/cyril.terrioux/LSNDP/instances.zip>.

The presented model is implemented in the OR-Tools CP-SAT solver (version 9.6.2534 [26]) via its Python interface. This choice is first guided by the solver efficiency, since the OR-Tools CP-SAT solver won several gold medals during the past MiniZinc Challenges [25]. Moreover, lazy clause generation [20] provides good results for the LSFRP problem [14]. Finally, another advantage is the possibility of exploiting a certain form of parallelism. Hence, for the solving, we run from 1 to 16 threads. When a single thread is run, it corresponds to the CP-SAT solver. Except for the number of threads, all the parameters are the default ones. The experiments are being conducted on servers with Intel Xeon Gold 5218R processors running at 2.1 GHz and 192 GB of memory with a time limit of two hours. When exploiting several threads, each instance is solved 10 times and the reported runtime is the average time. The solving step involving  $t$  threads is denoted  $\times t$ . We apply it to our model  $M$ , but also, to two derived versions denoted  $M-1,2$  and  $M-2$ . Model  $M-1,2$  does not consider constraints (S.1) and (S.2) while model  $M-2$  uses constraints (S.1), but not constraints (S.2).

## 4.2 Results

First, we compare our model with its two derived versions from the efficiency viewpoint (see Table 1<sup>2</sup> and Figure 13). Clearly, model  $M$  is the most efficient. Indeed, the addition of constraints (S.1) and (S.2) allows us to solve optimally more instances (24 instances for model  $M$  against 14 for models  $M-1,2$  and  $M-2$  when using a single thread) while reducing significantly the runtime. As there often exists an arc between each pair of ports/canals, the circuit constraint admits a huge number of allowed tuples. In practice, the number of allowed tuples studied by the solver is mainly restricted by the load and transit time constraints or the objective function. In the latter case, as the objective function considers all the rotations, it may take some time for the solver to realize that a rotation is not suitable. So finding an optimal solution may require exploring a huge number of feasible solutions. Constraints (S.1) and (S.2) allow us to reduce this number significantly, as we can see in our results. Then, as shown in Table 1, exploiting several threads allows improving the efficiency, but mostly by reducing the runtime. For instance, using 16 threads instead of a single one leads to reducing the runtime by a factor of 5 on average and up to 20 at best.

Figure 13 indicates that the runtime increases exponentially with the number of ports. However, other parameters affect the runtime like the number of commodities to be processed or the type of instances. For example, our model performs well on Baltic and WAF instances, which corresponds to feeder instances (i.e. a collection of small services that ensures the transport of commodities between some main ports and satellite ones). In contrast, it turns out to be less efficient for instances like EuropeAsia ones that connect the more important ports of two commercial areas. Then, our model finds interesting solutions even if it does not accept all the commodities or visit all the ports (see Table 2). One explanation is related to

<sup>2</sup> See Appendix A for the other instances.



## 16:16 A CP Approach for the Liner Shipping Network Design Problem

■ **Table 1** Runtimes in seconds for some representative instances (a runtime different from 7,200 s corresponds to an instance solved optimally).

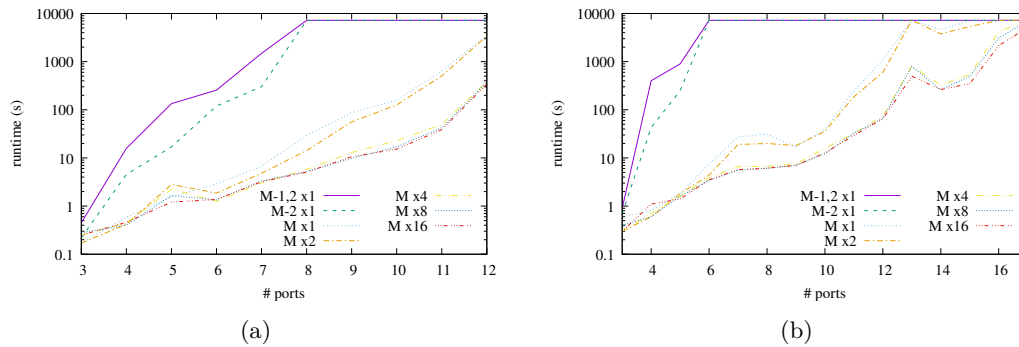
Instance	$M-1,2 \times 1$	$M-2 \times 1$	$M \times 1$	$M \times 2$	$M \times 4$	$M \times 8$	$M \times 16$
Baltic	7,200	7,200	3,502	3,385	386	372	343
Baltic_sub7	1,483	301	6.7	4.8	3.0	3.3	3.2
EuropeAsia_sub7	7,200	7,200	7,200	7,200	7,200	7,200	7,200
Mediterranean_sub7	7,200	7,200	7,200	7,200	7,200	7,001	6,234
WAF_sub7	7,200	7,200	27.4	19.0	6.6	5.6	5.7
WAF_sub17	7,200	7,200	7,200	7,200	7,200	7,021	5,117

■ **Table 2** Information of some instances and solutions (the value in k\$ of the best solution found, the number of visited ports, accepted commodities and used rotations).

Instance	Instance		Solution				
	Name	$ \mathcal{P} $	$ \mathcal{K} $	Cost	#ports	#comm.	#rot
Baltic	Baltic	12	22	4,752	10	16	3
Baltic_sub7	Baltic_sub7	7	12	2,508	6	9	2
EuropeAsia_sub7	EuropeAsia_sub7	7	42	4,228	6	23	3
Mediterranean_sub7	Mediterranean_sub7	7	26	225	6	15	2
WAF_sub7	WAF_sub7	7	12	5,823	7	12	3
WAF_sub17	WAF_sub17	17	32	11,952	10	16	4

the way the LINER-LIB benchmark was built (namely by aggregating data from different shipowners without ensuring that the considered fleet can handle all the commodities).

Finally, our approach manages to optimally solve some instances of up to 17 ports. While the literature reports solving of up to a dozen ports, it is difficult to compare accurately with existing exact methods: implementations are generally not available and each work treats the problem with a different point of view, in particular regarding the working hypotheses or the cost function to optimize (see Section 5).



■ **Figure 13** Runtime (in s) for Baltic (a) and WAF (b) instances w.r.t. the number of ports.

## 5 Related Work

The Liner Shipping Fleet Repositioning Problem (LSFRP [28]) aims to adapt the network in order to take into account some evolution of the customer needs (e.g. seasonality, port congestion, increase or decrease of the demands, ...). Moving container ships from one service to another while considering commodity transport is a complex and expensive task

for shipping companies. In this context, the CP approach presented in [14] turns out to be more efficient than MIP ones. In the case of LSFRP, services are already defined while LSNDP aims to design them. Moreover, the number of vessels and commodities to handle may be reduced for LSFRP. Finally, an LSFRP instance corresponds to a one-shot task while an LSNDP one leads to a schedule for several weeks or months.

LSNDP is close to Vehicle Routing Problem (VRP [15]) and its variants like the pickup and delivery problem (PDP [11]). Indeed, in both cases, vehicles carry commodities from one location to another. The first difference is that generally, for VRP and PDP, a commodity is carried by a single vehicle whereas, for LSNDP, the transport can be achieved by several vessels operating different rotations thanks to transshipments. Taking into account transshipments makes the problem more difficult. This requires additional variables and constraints, while significantly increasing the number of possible routes for commodities (and therefore the number of feasible solutions to study). Moreover, in general, VRP and PDP aim to transport all the commodities while, for LSNDP, some commodities may be rejected. Finally, the objective function is often more complex for LSNDP than for VRP and PDP. For instance, the variety of considered costs (e.g. call cost, transshipment cost, fuel cost, charter rate, ...) is more important.

Regarding the exact solving of LSNDP, unlike our model, the models proposed in [18, 23, 22] use a constant speed for each leg and do not handle transit time constraints. Those of [18, 23] cannot reject a commodity. In contrast, [18] takes into account the empty container repositioning while [23, 22] consider the transshipment costs. The objective functions consist in minimizing costs [18, 23] or maximizing the profit [22]. Regarding the type of service, the three models consider a more general form than ours. For instance, they can exploit butterfly services (i.e. services that can call several times in the same port). However, such an extension could be taken into account in our model by duplicating ports as we do for canals and relaxing some constraints like constraints (F.9). Note that the experimentations achieved in [22] rely on the LINER-LIB benchmark, but the proposed approach does not succeed in solving optimally the Baltic instance.

## 6 Conclusions and Perspectives

In this paper, we have proposed a first CP model to solve the LSNDP problem. The first practical results are very encouraging with optimally solved instances with up to 17 ports and show the interest in a CP approach. In the future, this model will have to be extended to better handle some kinds of instances and take into account other forms of services (e.g. butterflies) or the constraints imposed by the International Maritime Organization (IMO) concerning the gas emissions of ships (e.g. related to carbon intensity indicator). Another extension would be to differentiate containers by type (full, empty, reefers, ...). In particular, the repositioning of empty containers is an important issue, while the transport of reefers is highly profitable and raises specific questions. Moreover, to facilitate scaling up, the use of incomplete methods will need to be explored more deeply.

---

### References

- 1 R. Agarwal and Ö. Ergun. Ship Scheduling and Network Design for Cargo Routing in Liner Shipping. *Transportation Science*, 42(2):175–196, 2008. doi:10.1287/trsc.1070.0205.
- 2 J. F. Álvarez. Joint routing and deployment of a fleet of container vessels. *Maritime Economics & Logistics*, 11(2):186–208, 2009. doi:10.1057/me1.2009.5.

- 3 N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global constraint catalog. Technical Report T2012:03, TASC-SICS-LINA, 2014.
- 4 N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, 1994.
- 5 B. D. Brouer, G. Desaulniers, C. V. Karsten, and D. Pisinger. A Matheruristic for the Liner Shipping Network Design Problem with Transit Time Restrictions. In *Computational Logistics - 6th International Conference, ICCL*, volume 9335 of *Lecture Notes in Computer Science*, pages 195–208. Springer, 2015. doi:10.1007/978-3-319-24264-4\_14.
- 6 B. D. Brouer, C. V. Karsten, and D. Pisinger. Optimization in liner shipping. *Annals of Operations Research*, 271(1):205–236, 2018. doi:10.1007/s10479-018-3023-8.
- 7 B. D. Brouer, J. F. Álvarez, C. E. M. Plum, D. Pisinger, and M. M. Sigurd. A base integer programming model and benchmark suite for liner-shipment network design. *Transportation Science*, 48(2):281–312, 2014. doi:10.1287/trsc.2013.0471.
- 8 M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Programming Languages: Implementations, Logics, and Programs (PLILP)*, volume 1292 of *Lecture Notes in Computer Science*, pages 191–206, 1997. doi:10.1007/BFb0033845.
- 9 M. Christiansen, E. O. Hellsten, D. Pisinger, D. Sacramento, and C. Vilhelmsen. Liner shipping network design. *European Journal of Operational Research*, 286, 2020. doi:10.1016/j.ejor.2019.09.057.
- 10 R. Cymer. Dulmage-Mendelsohn Canonical Decomposition as a generic pruning technique. *Constraints*, 17(3):234–272, 2012.
- 11 Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991. doi:10.1016/0377-2217(91)90319-Q.
- 12 C. V. Karsten, B. D. Brouer, G. Desaulniers, and D. Pisinger. Time constrained liner shipping network design. *Transportation Research. Part E: Logistics and Transportation Review*, 105:152–162, 2017. doi:10.1016/j.tre.2016.03.010.
- 13 C. V. Karsten, D. Pisinger, S. Røpke, and B. D. Brouer. The time constrained multi-commodity network flow problem and its application to liner shipping network design. *Transportation Research Part E: Logistics and Transportation Review*, 76:122–138, 2015. doi:10.1016/j.tre.2015.01.005.
- 14 E. Kelareva, K. Tierney, and P. Kilby. CP methods for scheduling and routing with time-dependent task costs. *EURO Journal on Computational Optimization*, 2(3):147–194, 2014. doi:10.1007/s13675-014-0022-7.
- 15 P. Kilby and P. Shaw. Vehicle routing. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 801–836. Elsevier, 2006. doi:10.1016/S1574-6526(06)80027-1.
- 16 D. Kizilay, P. Van Hentenryck, and D. Türsel Eliiyi. Constraint programming models for integrated container terminal operations. *European Journal of Operational Research*, 286(3):945–962, 2020. doi:10.1016/j.ejor.2020.04.025.
- 17 A. Krosggaard, D. Pisinger, and J. Thorsen. A flow-first route-next heuristic for liner shipping network design. *Networks*, 72(3):358–381, 2018. doi:10.1002/net.21819.
- 18 Q. Meng and S. Wang. Liner shipping service network design with empty container repositioning. *Transportation Research Part E: Logistics and Transportation Review*, 47(5):695–708, 2011. doi:10.1016/j.tre.2011.02.004.
- 19 Q. Meng and S. Wang. Optimal operating strategy for a long-haul liner service route. *European Journal of Operational Research*, 215(1):105–114, 2011. doi:10.1016/j.ejor.2011.05.057.
- 20 O. Ohrimenko, P. J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints An International Journal*, 14(3):357–391, 2009. doi:10.1007/s10601-008-9064-x.
- 21 C. E. M. Plum, D. Pisinger, J. J. Salazar-González, and M. M. Sigurd. Single liner shipping service design. *Computers & Operations Research*, 45:1–6, 2014. doi:10.1016/j.cor.2013.11.018.

- 22 C. E. M. Plum, D. Pisinger, and M. M. Sigurd. A service flow model for the liner shipping network design problem. *European Journal of Operational Research*, 235(2):378–386, 2014. doi:10.1016/j.ejor.2013.10.057.
- 23 L. B. Reinhardt and D. Pisinger. A branch and cut algorithm for the container shipping network design problem. *Flexible Services and Manufacturing Journal*, 24(3):349–374, 2012. doi:10.1007/s10696-011-9105-4.
- 24 D. Sacramento, C. Solnon, and D. Pisinger. Constraint Programming and Local Search Heuristic: a Matheuristic Approach for Routing and Scheduling Feeder Vessels in Multi-terminal Ports. *Annals of Operations Research*, 1(4), 2020. doi:10.1007/s43069-020-00036-x.
- 25 P. J. Stuckey, T. Feydy, A. Schutt, G. Tack, and J. Fischer. The MiniZinc Challenge 2008-2013. *AI Magazine*, 35(2):55–60, 2014.
- 26 OR-Tools team. OR-Tools CP-SAT solver, 2023. URL: [https://developers.google.com/optimization/cp/cp\\_solver](https://developers.google.com/optimization/cp/cp_solver).
- 27 K. Thun, H. Andersson, and M. Christiansen. Analyzing complex service structures in liner shipping network design. *Flexible Services and Manufacturing Journal*, 29(3-4):535–552, 2017. doi:10.1007/s10696-016-9262-6.
- 28 K. Tierney. *Optimizing Liner Shipping Fleet Repositioning Plans*, volume 57 of *Operations research / computer science interfaces series*. Springer, 2015. doi:10.1007/978-3-319-17665-9.
- 29 P. Van Hentenryck and J.-P. Carillon. Generality versus Specificity: An Experience with AI and OR Techniques. In *Proceedings of the National Conference on Artificial Intelligence*, pages 660–664, 1988.
- 30 S. Wang and Q. Meng. Liner shipping network design with deadlines. *Computers & Operations Research*, 41:140–149, 2014. doi:10.1016/j.cor.2013.08.014.

## **A** Detailed Results

Tables 3 and 4 provide the same information as Tables 1 and 2 but for all the instances we consider.

■ **Table 3** Runtimes in seconds for all the considered instances (a runtime different from 7,200 s corresponds to an instance solved optimally).

Instance	$M-1,2 \times 1$	$M-2 \times 1$	$M \times 1$	$M \times 2$	$M \times 4$	$M \times 8$	$M \times 16$
Baltic_sub3	0.4	0.2	0.2	0.2	0.2	0.3	0.3
Baltic_sub4	15.8	4.5	0.6	0.4	0.4	0.4	0.5
Baltic_sub5	134	16.9	1.6	2.8	2.3	1.6	1.2
Baltic_sub6	256	120	2.9	1.9	1.3	1.4	1.4
Baltic_sub7	1,483	301	6.7	4.8	3.0	3.3	3.2
Baltic_sub8	7,200	7,200	29.2	14.2	5.8	5.2	5.1
Baltic_sub9	7,200	7,200	87.7	55.8	12.9	9.8	10.6
Baltic_sub10	7,200	7,200	158	126	22.9	16.9	15.2
Baltic_sub11	7,200	7,200	637	504	49.6	42.6	38.7
Baltic	7,200	7,200	3,502	3,385	386	372	343
EuropeAsia_sub3	2.5	1.0	0.9	1.0	1.2	1.3	1.2
EuropeAsia_sub4	157	244	134	100	37.3	18.1	10.5
EuropeAsia_sub5	7,200	7,200	7,200	3,736	4,101	4,459	2,356
EuropeAsia_sub6	7,200	7,200	7,200	7,200	7,200	7,200	7,200
EuropeAsia_sub7	7,200	7,200	7,200	7,200	7,200	7,200	7,200
EuropeAsia_sub8	7,200	7,200	7,200	7,200	7,200	7,200	7,200
EuropeAsia_sub9	7,200	7,200	7,200	7,200	7,200	7,200	7,200
EuropeAsia_sub10	7,200	7,200	7,200	7,200	7,200	7,200	7,200
Mediterranean_sub3	0.4	0.2	0.3	0.3	0.3	0.3	0.3
Mediterranean_sub4	2.8	0.8	0.8	0.8	0.8	0.9	1.0
Mediterranean_sub5	85.0	28.9	26.5	22.4	5.2	3.8	3.8
Mediterranean_sub6	621	229	281	307	30.6	25.2	21.5
Mediterranean_sub7	7,200	7,200	7,200	7,200	7,200	7,001	6,234
Mediterranean_sub8	7,200	7,200	7,200	7,200	7,200	7,200	7,200
Mediterranean_sub9	7,200	7,200	7,200	7,200	7,200	7,200	7,200
Mediterranean_sub10	7,200	7,200	7,200	7,200	7,200	7,200	7,200
WAF_sub3	0.9	0.7	0.6	0.3	0.3	0.4	0.3
WAF_sub4	401	42.8	0.8	0.6	0.7	0.6	1.1
WAF_sub5	896	248	1.8	1.8	1.6	1.6	1.4
WAF_sub6	7,200	7,200	7.5	4.4	3.9	3.4	3.5
WAF_sub7	7,200	7,200	27.4	19.0	6.6	5.6	5.7
WAF_sub8	7,200	7,200	31.3	19.9	6.7	6.1	6.1
WAF_sub9	7,200	7,200	17.0	18.0	7.5	7.0	7.0
WAF_sub10	7,200	7,200	39.4	35.7	16.0	12.3	12.7
WAF_sub11	7,200	7,200	249	187	32.2	32.6	29.3
WAF_sub12	7,200	7,200	1,093	600	73.4	66.7	64.9
WAF_sub13	7,200	7,200	7,200	7,200	815	792	499
WAF_sub14	7,200	7,200	4,662	3,779	323	262	263
WAF_sub15	7,200	7,200	7,034	5,365	542	494	349
WAF_sub16	7,200	7,200	7,200	7,200	4,182	3,106	2,154
WAF_sub17	7,200	7,200	7,200	7,200	7,200	7,021	5,117

■ **Table 4** Some information about instances and solutions. For solutions, we provide the value (in k\$) of the best solution found, the number of visited ports, one of accepted commodities and one of used rotations.


Instance Name	$\mathcal{P}$	$\mathcal{K}$	Solution			
			Cost	#ports	#comm.	#rot
Baltic_sub3	3	4	1,876	3	4	1
Baltic_sub4	4	6	1,895	3	4	1
Baltic_sub5	5	8	2,074	5	8	2
Baltic_sub6	6	10	2,074	5	8	2
Baltic_sub7	7	12	2,508	6	9	2
Baltic_sub8	8	14	3,322	7	10	3
Baltic_sub9	9	16	3,733	8	11	3
Baltic_sub10	10	18	4,187	9	13	3
Baltic_sub11	11	20	4,345	10	15	3
Baltic	12	22	4,752	10	16	3
EuropeAsia_sub3	3	6	616	3	4	1
EuropeAsia_sub4	4	12	616	3	4	1
EuropeAsia_sub5	5	20	1,463	4	8	1
EuropeAsia_sub6	6	30	1,463	4	8	1
EuropeAsia_sub7	7	42	4,228	6	23	3
EuropeAsia_sub8	8	56	4,425	6	19	2
EuropeAsia_sub9	9	72	4,425	6	19	2
EuropeAsia_sub10	10	89	2,935	7	24	2
Mediterranean_sub3	3	5	177	3	5	1
Mediterranean_sub4	4	9	177	3	5	1
Mediterranean_sub5	5	14	196	4	8	1
Mediterranean_sub6	6	20	196	4	8	1
Mediterranean_sub7	7	26	225	6	15	2
Mediterranean_sub8	8	34	302	8	30	2
Mediterranean_sub9	9	43	509	8	30	2
Mediterranean_sub10	10	53	605	9	38	2
WAF_sub3	3	4	1,293	3	4	1
WAF_sub4	4	6	1,308	4	6	3
WAF_sub5	5	8	2,329	5	8	3
WAF_sub6	6	10	2,911	6	10	3
WAF_sub7	7	12	5,823	7	12	3
WAF_sub8	8	14	5,823	7	12	3
WAF_sub9	9	16	5,823	7	12	3
WAF_sub10	10	18	7,543	8	14	3
WAF_sub11	11	20	8,831	9	15	4
WAF_sub12	12	22	9,764	10	16	4
WAF_sub13	13	24	11,282	11	18	4
WAF_sub14	14	26	11,952	10	16	4
WAF_sub15	15	28	11,952	10	16	4
WAF_sub16	16	30	11,952	10	16	4
WAF_sub17	17	32	11,952	10	16	4




# Optimization Models for Pickup-And-Delivery Problems with Reconfigurable Capacities

Arnoosh Golestanian ✉ 

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Giovanni Lo Bianco ✉ 

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Chengyu Tao ✉ 

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

J. Christopher Beck ✉ 

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

---

## Abstract

When a transportation service accommodates both people and goods, operators sometimes opt for vehicles that can be dynamically reconfigured for different demands. Motivated by air service in remote communities in Canada's north, we define a pickup-and-delivery problem in which aircraft can add or remove seats during a multi-stop trip to accommodate varying demands. Given the demand for people and cargo as well as a seat inventory at each location, the problem consists in finding a tour that picks up and delivers all demand while potentially reconfiguring the vehicle capacity at each location by adding or removing seats. We develop a total of six models using three different approaches: constraint programming, mixed integer programming, and domain-independent dynamic programming. Our numerical experiments indicate that domain-independent dynamic programming is able to substantially outperform the other technologies on both solution quality and run-time on a set of randomly generated instances spanning the size of real problems in northern Canada.

**2012 ACM Subject Classification** Computing methodologies → Modeling methodologies

**Keywords and phrases** Pickup and delivery, Dial-a-ride problem, Optimization

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.17

**Funding** This research was supported by the Natural Sciences and Engineering Research Council of Canada.

## 1 Introduction

Pickup-and-delivery problems involve using vehicles to transport goods and/or passengers from a set of origins to a set of destinations on a given transportation network [1]. A typical pickup-and-delivery problem such as the Pickup and Delivery Traveling Salesperson Problem (PD-TSP) includes a one or more vehicles, requests with different pickup and delivery locations, and an objective to find a minimum-cost tour (or set of routes) that visit(s) each pickup location before its corresponding delivery location [4]. There has been substantial research literature on pickup and delivery problems over the past several years (e.g., [19, 21]) motivated, in part, by global efforts to reduce transportation-related carbon emissions [16]. Many variations of such problems have been proposed and studied in the operations research literature. For example, some problems include handling costs when an item is loaded or unloaded depending on the position of the item in the vehicle [24] and some include subsets of requests that cannot be in a vehicle at the same time [5].

In this paper, we propose and study a novel variation of PD-TSP: requests can include both goods (cargo) and passengers and the vehicle has a capacity that can be adjusted en-route depending on the request and equipment stored at locations in the network. The



© Arnoosh Golestanian, Giovanni Lo Bianco, Chengyu Tao, and J. Christopher Beck; licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 17; pp. 17:1–17:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



problem is motivated by a real transportation problem faced by air services in northern Canada. Since many communities in this region are reachable only by air during some parts of the year, their access to basic human needs such as fresh food and healthcare services is limited. The need for air transportation combined with the relatively small populations and lack of resources led northern air services to adopt the practice of transporting both cargo and passengers on the same flights. The vehicles are aircraft with removable seats, allowing staff to either remove passenger seats and store them at airports to transport more cargo or add additional seats, previously stored at airports, to carry more passengers. The problem, which we call the Pickup-and-Delivery with Seat Replacement Problem (PD-SRP), therefore requires finding the shortest tour delivering all goods and passengers from their origins to destinations without exceeding aircraft capacity but allowing seats to be removed from or added to aircraft at each location, subject to seat availability and total aircraft capacity.

To solve the PD-SRP, we developed three types of optimization models: one Constraint Programming (CP) model, three Mixed Integer Programming (MIP) models, and two Domain-Independent Dynamic Programming (DIDP) [14] models. We compare their performance on randomly generated instances based on the size of the problem in Canada's north, demonstrating that both of the DIDP models outperform the CP and MIP models in terms of the number of instances solved and proved optimal, solution quality, and solve time.

## 2 Related Works

Reconfigurable capacity is a general term in the transportation literature, typically indicating that vehicle capacity can be changed at some cost and/or limited by some constraints [22, 23]. Other terms such as multi-compartment vehicle or multi-purpose vehicle are used to convey a similar meaning [20, 8]. We review the vehicle routing and dial-a-ride problems literature for studies that considered adjustable vehicles.

*Vehicle Routing Problems (VRP):* The Vehicle Routing Problem and its many variations have been studied extensively over the past 50 years [18]. The idea of adjusting the vehicle to handle different types of demand has been studied in multi-compartment vehicle routing problems [20]. For example, Henke et al. [9] studied how to split the capacity of a truck into different compartments to maintain the separation of different colors of recycled glass. Similarly, for grocery distribution, different temperature-sensitive products can be transported on the same truck with multiple compartments [11]. In both of these problems, a vehicle's capacity configuration is fixed for its entire route and cannot be modified during the trip.

*Dial-a-Ride Problems (DARP):* In the Dial-a-Ride Problem a transportation request takes the form of pickup and delivery location pair and the service provider utilizes its fleet of vehicles to fulfill the requests while minimizing a cost function that typically includes some travel distance component [10]. Some variants include a reconfigurable vehicle capacity to serve the needs of different users: those who use seats or those who use wheelchairs [23]. Some of the vehicle seats can be folded and stored inside the vehicle to make room for passengers in wheelchairs. Unlike this problem, the seats of the vehicle in the PD-SRP cannot be stored on the aircraft without occupying cargo capacity and are instead detached and stored at the airports.

Hatzenbühler et al. [8] studied a multi-purpose pickup and delivery problem that can deliver passengers or cargo by exchanging the module of the vehicle at a depot or special service site. Each vehicle includes a removable module and a fixed platform such that changing modules modifies the ability of the vehicle from only carrying cargo to only carrying passengers and vice versa. Compared to problems with conventional solo-purpose vehicles,

requests can be served with a fewer vehicles but at the expense of adding extra service sites and visits. We can view the core multi-purpose pickup and delivery problem as a special case of PD-SRP where the seat exchange decisions must be all-or-none: either all seats are removed to maximize cargo space or all seats are installed to maximize passenger capacity.

### 3 Problem Definition

In PD-SRP, we are given  $n$  requests, each potentially requiring the transportation of cargo and passenger demands. Let  $V = P \cup D$  where  $P = \{v_1, \dots, v_n\}$  is the set of pickup locations and  $D = \{v_{n+1}, \dots, v_{2n}\}$  is the set of delivery locations. We assumed that cargo is shipped in unit-sized boxes, each having the same weight and volume. Although, in reality cargo is shipped in various shapes and weights, incorporating four-dimensional packing (i.e., combining volume and weight) would substantially complicate the problem. Therefore, similar to approximations done in practice by airlines (e.g., standard weight per passenger), we opted for this simplification.

Each request  $i$  includes picking up  $\hat{q}_i$  boxes of cargo and  $\hat{\pi}_i$  passengers from location  $v_i$  and delivering them to location  $v_{n+i}$ . Thus, the demand of the corresponding delivery location has an equal magnitude negative value (i.e.,  $-\hat{q}_i = \hat{q}_{i+n}$ ,  $-\hat{\pi}_i = \hat{\pi}_{i+n}$ ,  $\forall i \in P$ ). Note that this representation can model more complex patterns (e.g., requests that share pickup or delivery locations but not both) by copying locations for each unique pickup-delivery pair.

When an aircraft is at its maximum seat capacity, it has  $\hat{S}$  seats and can carry  $\hat{C}$  boxes of cargo. By removing a seat,  $L$  boxes of cargo capacity are added to the aircraft. Therefore, the maximum cargo capacity when removing all the seats is  $K = \hat{S}L + \hat{C}$ . Each location  $i$  starts with  $S_i^0$  stored seats and therefore the aircraft can add at most  $\min(\hat{S}, S_i^0)$  seats or remove at most  $\hat{S}$  seats when visiting location  $i$ . There is no maximum number of seats that can be stored at a given location.

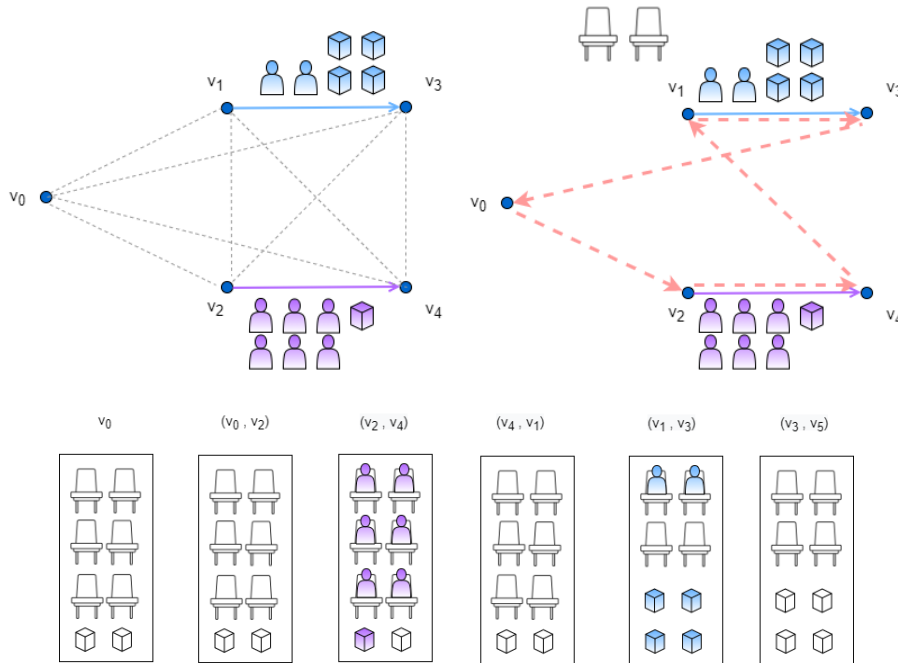
In order to represent the problem as a path, two nodes are assigned to the depot:  $v_0$  is the start node and  $v_{2n+1}$  is the end node. For modeling purposes we define sets  $V_{N+1} = V \cup \{v_{2n+1}\}$ ,  $V_0 = V \cup \{v_0\}$  and  $V_{0,N+1} = V \cup \{v_0, v_{2n+1}\}$ . Therefore, the problem is defined on graph  $G = (V_{0,N}, A)$  where  $A = \{(i, j) | i, j \in V_{0,N+1}, i \neq j\}$  with each arc having an associated distance,  $d_{ij}$ . The vehicle is initially at the depot  $v_0$  with a cargo and passenger capacity of  $\tilde{C}_0$  and  $\tilde{S}_0$  where  $\tilde{C}_0 = K - L\tilde{S}_0$  and  $\tilde{S}_0 \leq \hat{S}$ , respectively, and must finish the trip at depot  $v_{2n+1}$ . We assume that the start and end nodes are not the pickup or delivery location of any requests. Again, this assumption is not limiting as such requests can be represented by adding extra nodes at the same location as the start and end nodes.

In PD-SRP we aim to minimize the travel distance while deciding how many seats to add or remove at each location to fulfill all the requests while respecting capacities. The PD-SRP is NP-hard because if we fix the seat decisions and set all the demands to zero, the problem can be reduced to TSP which is known to be NP-hard [13].

An instance of this problem can be seen in Figure 1. The optimal tour is shown in pink, and the seat icon near each vertex represents the number of seats stored at the corresponding base. The optimal tour for this instance is  $(v_0, v_2, v_4, v_1, v_3, v_5)$  with two seats left at  $v_1$ .

### 4 Methods

We develop six models for the PD-SRP using constraint programming (CP), mixed integer programming (MIP), and domain independent dynamic programming (DIDP). One of the MIP models solves a restricted version of the PD-SRP and is used to warm-start the CP model and the two other MIP models. In this section, we describe each of the models in detail.



■ **Figure 1** Example of an PD-SRP instance with 2 requests. The optimal tour is shown by dotted pink edges. In the aircraft configuration, white seats and boxes show the current passenger and cargo capacity, respectively. The colored seats and boxes show the corresponding cargo and passenger requests that are picked up.

#### 4.1 A Constraint Programming Model (CP)

Our CP model equates distance and time and, thus, uses a one-machine scheduling approach where jobs correspond to the visits and the setup times between two consecutive jobs correspond to the distance between two locations. The model uses  $|V_{0,2n+1}|$  interval variables  $x_i$  that represent visits to each location, and a sequence variable,  $\pi$ , that constrains interval variables to form a sequence with an extra end node representing the return to the depot. The size of the interval variable is 0 because there is no service time associated with the visits. For every location  $i \in \{0, \dots, 2n\}$ , variable  $s_i$  is introduced to represent the number of seats that are added or removed. The formulation of the CP model is presented in Figure 2. Note that CP model is written in CP Optimizer language.

The objective function is the minimization of the total distance traveled by the aircraft.  $\text{EndOf}(x_{2n+1})$  corresponds to the end-point of the last interval variable in the sequence variable  $\pi$ : the time (i.e., total distance travelled) when the aircraft returns to the depot. Constraint (1a) ensures that each pair of consecutive interval variables is scheduled with a transition time equal to at least the required travel distance between the two corresponding locations. Constraint (1b) enforces that the pickup location of each request is visited before the delivery location. Constraint (1c) specifies that the aircraft begins and ends at the start and end depot locations.

We used three cumulative functions to represent the following values that are potentially changed by each interval variable (aircraft visits): available cargo space, number of empty seats, and the total number of seats. In particular, cumulative functions (1d) and (1f) are used to represent the available passenger and cargo space as the trip proceeds.  $H$  represents the number of empty seats in the aircraft (i.e., the available passenger space) and  $C$  represents

$$\begin{aligned}
\min \text{EndOf}(x_{2n+1}) & \tag{CP} \\
\text{s.t. NoOverlap}(\pi, \{d_{i,j} : (i,j) \in A\}) & \tag{1a} \\
\text{EndBeforeStart}(x_i, x_{n+i}) & \forall i \in \{1, \dots, n\} \tag{1b} \\
\text{First}(\pi, x_0), \text{Last}(\pi, x_{2n+1}) & \tag{1c} \\
C = \text{StepAt}(x_0, \tilde{C}_0) + \sum_{i=0}^{2n} \text{StepAtStart}(x_i, -\hat{q}_i - L \cdot s_i) & \tag{1d} \\
C \geq 0 & \tag{1e} \\
H = \text{StepAt}(x_0, \tilde{S}_0) + \sum_{i=0}^{2n} \text{StepAtStart}(x_i, -\hat{\pi}_i + s_i) & \tag{1f} \\
H \geq 0 & \tag{1g} \\
S = \text{StepAt}(x_0, \tilde{S}_0) + \sum_{i=0}^{2n} \text{StepAtStart}(x_i, s_i) & \tag{1h} \\
S \leq \hat{S} & \tag{1i} \\
x_i : \text{intervalVar}(0) & \forall i \in V_{0,2n+1} \tag{1j} \\
s_i : \text{integerVar}(-\hat{S}, \min(\hat{S}, S_i^0)) & \forall i \in V_0 \tag{1k} \\
\pi : \text{sequenceVar}(x_0, \dots, x_{2n+1}) & \tag{1l}
\end{aligned}$$

■ **Figure 2** The CP Model for the PD-SRP.

the available cargo space. Before the trip starts,  $K = H + C$  and, if there are  $\hat{S}_0$  seats in the aircraft at the start,  $H = L \cdot \hat{S}_0$ . The expression  $\text{StepAtStart}(var, impact)$  specifies the change (increment or decrement) to the cumulative function at the start of an interval variable. The available cargo space  $C$  will decrease as cargo and seats are picked up, therefore we use  $\text{StepAtStart}(x_i, -\hat{q}_i - L \cdot s_i)$  to represent the changes to available cargo space at each location  $i \in \{1, \dots, 2n\}$ . The available passenger space will decrease when cargo is picked up, while increasing when adding seats as represented by  $\text{StepAtStart}(x_i, -\hat{\pi}_i + s_i)$  at each location  $i \in \{0, \dots, 2n\}$ . The cumulative function  $S$  is introduced in constraint (1h) to describe the change of the total number of seats in the aircraft.  $S$  will change with the number of seats being added or removed as represented by  $s_i$ . Constraint (1i) restricts the total number of seats by the maximum seat capacity  $\hat{S}$ . In constraint (1k), the domain of  $s_i$  is  $[-\hat{S}, S_i^0]$  reflecting the range of the number of seats that the aircraft can remove or add at location  $i$ .

It should be noted that every interval variable contributes to the cumulative constraint, which means that these bounds are maintained throughout the sequence. Therefore, we do not need to have a separate cumulative function for every location of the tour.

## 4.2 Mixed Integer Programming Models

In this section, we describe three MIP models motivated by existing models for pickup and delivery problems. The first two models exactly represent the PD-SRP and therefore admit optimal solutions. The final model is a restriction of the PD-SRP problem that can be used to quickly find a feasible solution and, therefore, an upper bound for the PD-SRP. In our experiments, we investigate the use of this restricted model to warm-start the CP model and two other MIP models.

$$\begin{aligned}
 \min \quad & \sum_{i \in V_0} \sum_{j \in V_{N+1}, i \neq j} d_{ij} x_{ij} && (\text{MIIIP}_{loc}) \\
 & \sum_{j \in V_{N+1}} x_{ij} = 1 && i \in V_0 \quad (2a) \\
 & \sum_{j \in V_0, j \neq i} x_{ji} - \sum_{j \in V_{N+1}, i \neq j} x_{ij} = 0 && i \in V \quad (2b) \\
 & \tau_i + x_{ij} - |V|(1 - x_{ij}) \leq \tau_j && i \in V_0, j \in V_{N+1}, i \neq j \quad (2c) \\
 & 1 \leq \tau_i \leq |V| && i \in V \quad (2d) \\
 & \tau_i + 1 \leq \tau_{n+i} && i \in P \quad (2e) \\
 & y_i + \pi_i \leq \hat{S} && i \in V_0 \quad (2f) \\
 & y_i + \pi_i + s_i \leq \hat{S} && i \in V_0 \quad (2g) \\
 & \pi_i + \hat{\pi}_i \leq \hat{S} && i \in V_0 \quad (2h) \\
 & u_i + q_i + Ly_i + L\pi_i \leq K && i \in V_0 \quad (2i) \\
 & u_j \leq u_i - Ls_i - \hat{q}_i x_{ij} + (2K)(1 - x_{ij}) && i \in V, j \in V_{N+1}, i \neq j \quad (2j) \\
 & u_j \geq u_i - Ls_i - \hat{q}_i x_{ij} - (2K)(1 - x_{ij}) && i \in V, j \in V_{N+1}, i \neq j \quad (2k) \\
 & y_j \leq y_i + s_i - \hat{\pi}_i x_{ij} + 2\hat{S}(1 - x_{ij}) && i \in V, j \in V_{N+1}, i \neq j \quad (2l) \\
 & y_j \geq y_i + s_i - \hat{\pi}_i x_{ij} - 2\hat{S}(1 - x_{ij}) && i \in V, j \in V_{N+1}, i \neq j \quad (2m) \\
 & \pi_j \geq \pi_i + \hat{\pi}_i x_{ij} - \hat{S}(1 - x_{ij}) && i \in V, j \in V_{N+1}, i \neq j \quad (2n) \\
 & q_j \geq q_i + \hat{q}_i x_{ij} - K(1 - x_{ij}) && i \in V, j \in V_{N+1}, i \neq j \quad (2o) \\
 & -y_i \leq s_i \leq \min(\hat{S}, S_i^0) && i \in V \quad (2p) \\
 & \tau_0 = \pi_0 = q_0 = 0, u_0 = \tilde{C}_0, y_0 = \tilde{S}_0 && (2q) \\
 & x_{ij} \in \{0, 1\} && i \in V_0, j \in V_{N+1}, i \neq j \quad (2r) \\
 & u_i, y_i, \pi_i, q_i, \tau_i \in \mathbb{R}^{0+}, s_i \in \mathbb{R} && i \in V_{0, N+1} \quad (2s)
 \end{aligned}$$

■ **Figure 3** The  $\text{MIIIP}_{loc}$  Model for the PD-SRP.

#### 4.2.1 Two-indexed Location-Based MIP ( $\text{MIIIP}_{loc}$ )

We propose a two-indexed location-based MIP model for PD-SRP ( $\text{MIIIP}_{loc}$ ) based on a model for an existing pickup and delivery variant [7]. In  $\text{MIIIP}_{loc}$ ,  $x_{ij}$  is a binary variable that is 1 if arc  $(i, j) \in A$  is traveled and is 0 otherwise. Non-negative continuous variables  $\tau_i$ ,  $u_i$ , and  $y_i$  represent the distance, available cargo space, and empty seats, respectively, on arrival at vertex  $i \in V_{0, N+1}$ . As above, let variable  $s_i$  be the number of seats that are added ( $s_i > 0$ ) or removed ( $s_i < 0$ ) at location  $i$ . Finally, let  $\pi_i$  and  $q_i$  be the number of passengers and boxes of cargo on the aircraft on arrival at vertex  $i \in V_{0, N+1}$ , respectively. The  $\text{MIIIP}_{loc}$  model is shown in Figure 3.

The objective function minimizes the total distance traveled. Constraint (2a) ensures that each customer is visited exactly once while constraint (2b) forces an arrival and departure at each non-depot vertex. Constraints (2c) and (2d) prevent the formation of the subtours, using Miller-Tucker-Zemlin (MTZ) constraints [17]. Constraint (2e) forces the aircraft to visit the pickup location of each commodity before the delivery location. Constraints (2f) and (2g) respectively ensure that the total number of seats before and after adding or removing

new seats does not exceed the passenger capacity. Similarly, constraint (2h) ensures that the total number of passengers on the aircraft after fulfilling the demand of vertex  $i$  does not exceed the passenger capacity. Constraint (2i) enforces the relationship between  $u_i$  and  $y_i$ . Note that the left hand side of the constraint restricts the picked-up cargo and passengers to not exceed the aircraft capacity. Constraints (2j) and (2k) define the upper bound and lower bound on the available cargo space, respectively. Similarly, constraints (2l) and (2m) set the upper and lower bounds on the number of empty seats. Constraint (2n) ensures that the passenger demand is met at each location, while constraint (2o) does the same thing for the cargo demand. Constraint (2p) restricts the number of the seats that can be added based on their availability. The lower bound on the number of removed seats, when  $s_i < 0$ , is always the number of seats on the aircraft at the arrival of location  $i$ . Lastly, constraints (2q) - (2s) specify binary and continuous variable domains.

#### 4.2.2 Three-indexed Rank-Based MIP ( $\text{MIIP}_{rank}$ )

The three-indexed ranked-based MIP model for PD-SRP ( $\text{MIIP}_{rank}$ ) is adapted from a model for the multi-commodity pickup and delivery traveling salesperson problem [3]. In  $\text{MIIP}_{rank}$ ,  $z_{i,j}^t$  is a binary variable indicating that aircraft goes directly from location  $i$  to location  $j$  and location  $i$  is at position  $t$  of the tour, for  $i, j \in V_{0,N+1}, i \neq j, t \in \{0, \dots, 2n+1\}$ . Binary variable  $y_{i,t}$  is 1 if location  $i$  is visited at position  $t$  of the tour,  $i \in V_{0,N+1}, t \in \{0, \dots, 2n+1\}$  and 0 otherwise. Variable  $s_t$  is the number of seats added or removed at the  $t$ 'th position of the tour, for  $t \in \{0, \dots, 2n+1\}$ , with a negative value corresponding to the number of seats removed. Variables  $w_t$  and  $u_t$  represent the empty seats and available cargo space on arrival at  $t$ 'th position of the tour, for  $t \in \{0, \dots, 2n+1\}$ . Finally, let  $\pi_t$  and  $q_t$  be the number of passengers and boxes of cargo on arrival at  $t$ 'th position of the tour. The  $\text{MIIP}_{rank}$  is presented in Figure 4.

The objective function minimizes the total travel distance. Constraints (3a) and (3b) ensure that tour positions are assigned to exactly one location and that each location is visited exactly once, respectively. Constraint (3c) calculates the number of empty seats just before visit  $t$ , where  $\sum_{i=1}^n y_{i,t-1} \hat{\pi}_i$  is the number of passengers picked up at position  $t-1$  of the tour. Similarly, constraint (3d) calculates the available cargo space just before visit  $t$ , where  $\sum_{i=1}^n y_{i,t} \hat{q}_i$  is the amount of cargo picked up at position  $t$  of the tour. Constraint (3e) states that each commodity is picked up before it is delivered. Constraint (3f) enforces the relationship between  $w_t$  and  $u_t$ . The left hand side of the constraint enforces that the picked-up cargo and passengers do not exceed the available aircraft capacity. Constraint (3g) ensures that there is always  $\hat{C}$  space available for cargo on the aircraft. From (3f) and (3g) we can conclude that  $\pi_t + w_t \leq \hat{S}$ : the total number of seats does not exceed the passenger capacity. Constraint (3h) ensures the feasibility of the number of seats to be added or removed. Constraints (3i) and (3j) calculate the number of passengers and boxes of cargo at each position of the tour, respectively. Constraints (3k) and (3l) enforce the relationship between  $y$  and  $z$  variables and, together with (3e) and (3m), prevent the formation of subtours in a MTZ fashion. Lastly, constraints (3n) - (3q) specify the domains of the variables.

#### 4.2.3 Upper bound MIP Model ( $\text{MIIP}_{UB}$ )

Our preliminary experiments suggested that the CP and MIP models presented above struggled to find good feasible solutions. We, therefore, investigate the use of a third MIP model, designed to quickly find an upper bound on the PD-SRP by solving a restriction of the full problem. Such a model provides a heuristic solution as well as a potential warm-start solution for the complete models.

$$\begin{aligned}
 \min \quad & \sum_{t=0}^{2n} \sum_{i \in V_{0,N+1}} \sum_{j \in V_{0,N+1}, j \neq i} d_{i,j} z_{i,j}^t && (\text{MIP}_{rank}) \\
 \sum_{i \in V_0} y_{i,t} = 1 & && t \in \{0, \dots, 2n\} && (3a) \\
 \sum_{t=1}^{2n} y_{i,t} = 1 & && i \in V && (3b) \\
 w_t = w_{t-1} + s_{t-1} - \sum_{i=1}^n y_{i,t-1} \hat{\pi}_i & && t \in \{1, \dots, 2n+1\} && (3c) \\
 u_t = u_{t-1} - Ls_{t-1} - \sum_{i=1}^n y_{i,t-1} \hat{q}_i & && t \in \{1, \dots, 2n+1\} && (3d) \\
 \sum_{t=1}^n ty_{i,t} - \sum_{t=1}^n ty_{n+i,t} \leq -1 & && i \in P && (3e) \\
 q_t + u_t + Lw_t + L\pi_t \leq K & && t \in \{0, \dots, 2n+1\} && (3f) \\
 q_t + u_t \geq \hat{C} & && t \in \{0, \dots, 2n+1\} && (3g) \\
 -w_t \leq s_t \leq \min(\hat{S}, \sum_{i=1}^n S_i^0 y_{i,t}) & && t \in \{0, \dots, 2n+1\} && (3h) \\
 \pi_t = \pi_{t-1} + \sum_{i=1}^n y_{i,t-1} \hat{\pi}_i & && t \in \{1, \dots, 2n+1\} && (3i) \\
 q_t = q_{t-1} + \sum_{i=1}^n y_{i,t-1} \hat{q}_i & && t \in \{1, \dots, 2n+1\} && (3j) \\
 y_{i,t} - \sum_{j=0}^n z_{i,j}^t = 0 & && i \in V_0, t \in \{0, \dots, 2n\} && (3k) \\
 y_{j,t} - \sum_{i=0}^n z_{i,j}^{t-1} = 0 & && j \in V_{0,N+1}, t \in \{1, \dots, 2n+1\} && (3l) \\
 y_{0,0} = y_{2n+1,2n+1} = 1, y_{0,t} = 0 & && t \in \{1, \dots, 2n\} && (3m) \\
 s_t \leq \hat{S}, w_t \leq \hat{S}, u_t \leq K & && t \in \{0, \dots, 2n+1\} && (3n) \\
 u_0 = \tilde{C}_0, w_0 = \tilde{S}_0, \pi_0 = q_0 = 0 & && && (3o) \\
 y_{i,t} \in \{0, 1\}, z_{i,j}^t \in \{0, 1\} & && i, j \in V_{0,N+1}, t \in \{0, \dots, 2n+1\} && (3p) \\
 u_t, w_t, \pi_t, q_t \in \mathbb{R}^{0+}, s_t \in \mathbb{R} & && t \in \{0, \dots, 2n+1\} && (3q)
 \end{aligned}$$

■ **Figure 4** A Three-Indexed MIP Model for the PD-SRP.

$$\begin{aligned}
\min \quad & \sum_{i \in P_0} \sum_{j \in P_0, i \neq j} c_{i,j} x_{i,j} && (\text{MIP}_{UB}) \\
\text{s.t.} \quad & \sum_{j \in P_0, i \neq j} x_{i,j} = 1 && \forall i \in P \quad (4a) \\
& \sum_{i \in P_{N+1}, i \neq j} x_{j,i} - \sum_{i \in P_0, i \neq j} x_{i,j} = 0 && \forall j \in P \quad (4b) \\
& t_i + x_{i,j} - |P|(1 - x_{i,j}) \leq t_j && \forall i \in P_{N+1}, j \in P_{N+1}, i \neq j \quad (4c) \\
& s_j \leq s_i + (S_{i+n}^0 + S_j^0)x_{i,j} + |\hat{S}|(1 - x_{i,j}) && \forall i \in P_0, j \in P_{N+1}, i \neq j \quad (4d) \\
& Ls_i + \hat{q}_i \leq K && \forall i \in P_{N+1} \quad (4e) \\
& 1 \leq t_i \leq |P| && \forall i \in P \quad (4f) \\
& s_i \leq \hat{S} && \forall i \in P \quad (4g) \\
& s_i \geq \hat{\pi}_i && \forall i \in P \quad (4h) \\
& t_0 = 0 && (4i) \\
& \tilde{S}_0 \leq s_0 \leq \tilde{S}_0 + S_0^0 && (4j) \\
& x_{i,j} \in \{0, 1\} && \forall i \in P \quad (4k) \\
& t_i \in \mathbb{N}, s_i \in \mathbb{N} && \forall i \in P_{0,N+1} \quad (4l)
\end{aligned}$$

■ **Figure 5** The Upper Bound MIP model for a restriction of PD-SRP.

The upper bound model is obtained by over-constraining the original problem to require that a request must be delivered immediately after being picked up. The nodes in this problem include the start depot  $v_0$ , the end depot  $v_{N+1}$ , and all the pickup nodes  $P = \{v_1, \dots, v_n\}$ . For modeling purposes we define sets  $P_{N+1} = P \cup \{v_{N+1}\}$ ,  $P_0 = P \cup \{v_0\}$  and  $P_{0,N+1} = P \cup \{v_0, v_{N+1}\}$ . The delivery nodes are not explicitly included because each origin-to-destination trip takes place immediately after the visit to the pickup node with the total distance increased by both the travel to the pickup node and the travel between the pickup node and the delivery node.

The  $\text{MIP}_{UB}$  model is presented in Figure 5. Let  $x_{i,j}$  be a binary variable indicating that the aircraft goes from the delivery location of the request  $i$  to the pickup location of request  $j$ . Let  $s_i$  be the number of seats in the aircraft right after visiting location  $i$ . Finally, let  $t_i$  be the position of location  $i$  on the tour. The solution returned by this model is likely to be sub-optimal for the PD-SRP.

The objective function minimizes the total distance traveled. The coefficient  $c_{i,j}$  represents the total distance starting from the delivery location of request  $i$ , visiting the pickup location of request  $j$ , and then travelling to the delivery locations of request  $j$ . Request 0 is to travel from the depot to the pickup location of the first request. The delivery and pickup locations of request 0 are nodes  $v_0$  and  $v_{2n+1}$ , respectively.

Constraints (4a) and (4b) ensure that each node is visited exactly once. Constraints (4c), (4f), and (4i) prevent the formation of subtours. Constraint (4d) describes seat changes when the aircraft visits a node and constraint (4e) requires that the space taken up by the seats in the aircraft must be less than or equal to the remaining space after picking up the cargo of the current request. Constraint (4g) restricts the number of seats to never surpasses the maximum number of seats allowed in the aircraft and constraint (4h) ensures that the number of seats in the aircraft never drops below the number of passengers to be picked up. Constraints (4j) - (4l) specify the domains of the variables.



### 4.3 Domain-Independent Dynamic Programming Models

Domain-Independent Dynamic Programming (DIDP) is a recently proposed methodology for solving combinatorial optimization problems by formulating the problem as state-based dynamic program (DP) and using a generic solver to solve it [14]. DP models are declaratively formulated in Dynamic Programming Description Language (DyPDL), a solver-independent modeling formalism for DP that is inspired by AI planning. In DyPDL, a model consists of the following:

- *state variables*: variables that take on numeric, set, or set-element values that define the states in the search space of the problem
- *target state*: the problem state for which the optimal value is to be computed by the recursive formulation
- *constants*: state-independent values
- *transitions*: decisions in the DP that move between states
- *base cases*: a set of conditions that define states that terminate the recursion
- *state constraints*: conditions that must be satisfied by all states
- *dual bound*: an optional lower (upper) bound on the objective function for minimization (maximization) problems.

We developed two DIDP models for the PD-SRP.

#### 4.3.1 A Two-transition DIDP Model ( $\text{DIDP}_{2T}$ )

Our first DIDP model has two types of transition: one to represent adding or removing seats and picking up or delivering cargo and passengers and a second to model moving the aircraft to a different location. In the model, a state is a tuple  $\langle U, i, q, \pi, s, \alpha \rangle$ , which represents the set of unvisited vertices,  $U$ , the current location,  $i$ , the cargo load,  $q$ , the number of passengers,  $\pi$ , the number of seats,  $s$ , and a flag representing which type of transition to apply,  $\alpha$ . We set  $\alpha = 1$  if we have finished pickup/delivery at a location to indicate that the next transition should be to move the aircraft. Otherwise,  $\alpha = 0$ .

The  $\text{DIDP}_{2T}$  model is defined in Figure 6. We focus first on Eqs. (5c) and (5d), which respectively define the possible seat changes and possible next locations at a location  $i$ .

Suppose that the number of seats at the current location  $i$  is increased by  $\delta$ . Since there are  $S_i^0$  seats stored at each location initially, when the aircraft has  $s$  seats, at  $i$  we can add at most  $\min\{S_i^0, \hat{S} - s\}$  seats and remove at most  $s$  seats. For simplicity we will denote  $\hat{S}_i = \min\{S_i^0, \hat{S} - s\}$ . Therefore,  $\delta \in [-s, \hat{S}_i]$ . Let numeric constants  $w_i$  and  $u_i$  be the net change of cargo and passengers at location  $i$ , respectively. The cargo will be increased by  $w_i$ , so the current cargo will become  $q + w_i \leq K - (s + \delta)L$ , the current space for cargo. Similarly, the number of passengers will be  $\pi + u_i \leq s + \delta$ . Lastly,  $\delta$  must only take integer values. With these conditions, Eq. (5c) specifies the values of  $\delta$ .

Consider visiting the next location,  $j$ , from current location  $i$ . To be a valid location to visit next,  $j$  must be unvisited ( $j \in U$ ), it must be connected by an edge in the graph to  $i$  ( $(i, j) \in A$ ), and it must be either a pickup location ( $j \notin D$ ) or its corresponding pickup location must have already been visited. If we let  $p_j$  be the pickup location for the request whose delivery location is  $j$ , then this final condition is:  $p_j \notin U$ . Eq. (5d) represents the candidate locations to visit next after current location  $i$ .

The objective function specifies the state for which the optimal cost needs to be computed: the state where all pickup and delivery nodes are unvisited, the current location is the start depot ( $v_0$ ), the cargo and passenger loads are 0, the aircraft has  $\tilde{S}_0$  seats, and the next

$$\begin{aligned}
& \text{compute } Z(V, 0, 0, 0, \tilde{S}_0, 0) && (\mathbb{DIDP}_{2T}) \\
Z(U, i, q, \pi, s, \alpha) &= \begin{cases} d_{i, 2n+1} & \text{if } U = \emptyset, \alpha = 1 \\ \min_{\delta \in T(q, \pi, s, i)} Z(U, i, q + w_i, \pi + u_i, s + \delta, 1) & \text{if } U \neq \emptyset, \alpha = 0 \\ \min_{j \in R(U, i)} d_{i, j} Z(U \setminus \{j\}, j, q, \pi, s, 0) & \text{if } U \neq \emptyset, \alpha = 1 \end{cases} && (5a) \\
Z(U, i, q, \pi, s, \alpha) &\geq 0 && (5b) \\
T(i, q, \pi, s) &= \left\{ \delta \in [-s, \hat{S}_i] \mid q + w_i \leq K - (s + \delta)L \wedge \pi + u_i \leq s + \delta, \delta \in \mathbb{Z} \right\} && (5c) \\
R(U, i) &= \{j \in U \mid (i, j) \in A \wedge (j \notin D \vee p_j \notin U)\}. && (5d)
\end{aligned}$$

■ **Figure 6** The Two-transition DIDP Model ( $\mathbb{DIDP}_{2T}$ ) for PD-SRP.

$$\begin{aligned}
& \text{compute } Z(V, 0, 0, 0, \tilde{S}_0) && (\mathbb{DIDP}_{1T}) \\
Z(U, i, q, \pi, s) &= \begin{cases} d_{i, 2n+1} & \text{if } U = \emptyset \wedge \exists \delta \in T(i, q, \pi, s) \\ \min_{(\delta, j) \in T(i, q, \pi, s) \times R(U, i)} d_{i, j} + Z(U \setminus \{j\}, j, q + w_i, \pi + u_i, s + \delta) & \text{if } U \neq \emptyset \end{cases} && (6a) \\
Z(U, i, q, \pi, s) &\geq 0 && (6b) \\
& \text{Eq. (5c), Eq. (5d).}
\end{aligned}$$

■ **Figure 7** The One-transition DIDP Model ( $\mathbb{DIDP}_{1T}$ ) for PD-SRP.

transition should be to move the aircraft ( $\alpha = 0$ ). In Eq. (5a), the first line computes the cost to return to the depot from node  $i$ , the second line describes the cost of adding or removing  $\delta$  seats at node  $i$ , and the third line describes the cost of visiting node  $j$  from  $i$ . Note that when the aircraft is moved, the state variable  $\alpha$  is set to 0 and if the decision regarding seats is made in this transition,  $\alpha$  is set to 1. Constraint (5b) is a dual bound for the DIDP model which is optional but may be exploited by the solver.

### 4.3.2 A One-transition DIDP Model ( $\mathbb{DIDP}_{1T}$ )

We present the  $\mathbb{DIDP}_{1T}$  model in Figure 7. In this model, instead of two types of transitions, we define one type that performs the pickup/delivery and seat exchange at a location and then moves the aircraft to a new location. A state is the same as in  $\mathbb{DIDP}_{1T}$  with the exception of the  $\alpha$  flag which is no longer necessary:  $\langle U, i, q, \pi, s \rangle$ . As a transition first picks up or delivers cargo, passengers, and seats at the current location and then moves the aircraft to the next location, each transition corresponds to selecting  $(\delta, j)$ :  $\delta$  is the number of picked up seats and  $j$  is the next location to visit. The set of possible decisions at each state is therefore  $T(i, q, \pi, s) \times R(U, i)$  as defined in the second line of Eq. (6a).

The objective function of  $\mathbb{DIDP}_{1T}$  defines the state for which the optimal cost is to be calculated. It is identical to the target state in  $\mathbb{DIDP}_{2T}$  with the removal of  $\alpha$ . In Eq. (6a), the first line describes the cost of returning to the depot from node  $i$ , and the second line describes the cost of visiting node  $j$  from  $i$ . Note that the first line checks if there exists some  $\delta$  such that the capacity constraints on the cargo and the passengers are satisfied. If there is no such  $\delta$ , we assume  $Z(\emptyset, i, q, \pi, s) = \infty$ .

### 4.3.3 Model Sizes and Solver

In a DIDP model, we need to define all transitions that are applicable in a state. In  $\mathbb{DIDP}_{2T}$ ,  $\delta$  can take an integer in  $[-\hat{S}, \hat{S}]$  depending on a state, so there are  $2\hat{S} + 1$  candidates. We have  $|V_{N+1}|$  locations to visit. Thus,  $\mathbb{DIDP}_{2T}$  requires  $2\hat{S} + 1 + |V_{N+1}|$  transitions to be defined in total. In contrast,  $\mathbb{DIDP}_{1T}$  needs to define  $(2\hat{S} + 1)|V_{N+1}|$  transitions but does not have state variable  $\alpha$ . An alternative perspective is that the two DIDP models make different trade-offs between the maximum branching factor and solution length.  $\mathbb{DIDP}_{1T}$  has a branching factor of at most  $(2\hat{S} + 1)|V_{N+1}|$  at each state and a solution path length of  $|V_{N+1}|$ .  $\mathbb{DIDP}_{2T}$  has a maximum branching factor that alternates between  $2\hat{S} + 1$  and  $|V_{N+1}|$  and a solution length of  $2|V_{N+1}|$ . The performance of a solver is affected by the number of state variables, the branching factor, and the solution length.

We solve the DIDP models with a complete anytime beam search (CABS) solver [25, 15]. CABS is an anytime algorithm meaning that seeks to quickly find a feasible solution and then to improve it in the remaining run-time. CABS employs beam search: a heuristic search algorithm that maintains a fixed number,  $b$  (beam width), of best states when exploring the search space. In CABS, beam search is performed iteratively with increasing the beam width until a stopping condition is met. Due to the iteratively increasing beam width, CABS is a complete algorithm [25].

## 5 Numerical Evaluation

### 5.1 Experimental Setup

We have developed six different models, i.e., CP,  $\text{MIP}_{loc}$ ,  $\text{MIP}_{rank}$ ,  $\text{MIP}_{UB}$ ,  $\mathbb{DIDP}_{1T}$ ,  $\mathbb{DIDP}_{2T}$ . For the experiment, we use  $\text{MIP}_{UB}$  to warm-start the MIP and CP models, producing three additional approaches:  $\text{MIP}_{loc\_W}$ ,  $\text{MIP}_{rank\_W}$  and  $\text{CP}_W$ .

To implement and solve the models we used Python v3.8.0 and the corresponding Python interfaces to the solvers: Gurobi Optimizer 10.0.1 and gurobipy for MIP, CP Optimizer 22.1.0.0 and DOCplex for CP, and didppy 0.3.3 for DIDP.<sup>1</sup> Each run has a time limit of 600s. The machine used to run the experiment has Intel(R) Core(TM) i7-9700 8 core CPU @ 3.00GHz, 12MB cache, and memory of 31G.

The models are tested on randomly generated instances with sizes 4, 6, 8, 10, 12, 15, and 20 with 10 instances per size. The size of each instance is the number of requests, which is half of the number of locations. We generate problem instances randomly, approximately reflecting real-world problem size, aircraft capacity and configurations, and stored seats at each location. We fix the maximum number of seats in the aircraft  $\hat{S} = 6$ , the cargo-to-seat ratio  $L = 100$ , and the cargo capacity on a full-seat aircraft  $\hat{C} = 200$ . The number of seats in the aircraft start configuration,  $\tilde{S}_0$ , is selected uniformly from  $\{0, \dots, 6\}$  and the cargo capacity in the start configuration is  $\tilde{C}_0 = 800 - 100\tilde{S}_0$ . Similarly, the number of seats available at location  $i$ ,  $S_i^0$ , is set uniformly from  $\{0, \dots, 6\}$ , independently for each location. The  $(x, y)$  coordinates of every location are uniformly generated from  $\{0, \dots, 100\}^2$ .

We generate the passenger and cargo demand to ensure the existence of capacity-feasible solutions. For each request  $i \in \{1, \dots, n\}$ , there is a demand of  $\hat{\pi}_i$  passengers and demand of  $\hat{q}_i$  kg cargo (i.e.,  $\hat{q}_i/L$  units of cargo). We first define the total number of passengers and units of cargo as  $\hat{K} = \hat{q}_i/L + \hat{\pi}_i$ , and  $\hat{K}$  is uniformly generated from  $\{1, \dots, \hat{S} + \hat{C}/L = 8\}$ . The passenger request  $\hat{\pi}_i$  is then selected uniformly from  $\{0, \dots, \min(\hat{S}, \hat{K})\}$ . Consequently, the cargo request is  $\hat{q}_i = L(\hat{K} - \hat{\pi}_i)$ .

<sup>1</sup> <https://didp.ai>

To compare the models, we used the number of instances solved and proved optimal, the PAR10 score time [12] (i.e., mean run-time with 10 times the time limit used if no optimal solution was proved), and mean relative error (MRE).

MRE compares the solution quality returned by each model. For an optimization problem let  $obj_{t,m,i}$  be the objective value of the best solution achieved by time  $t$  of model  $m$  for instance  $i$  and let  $obj_i^*$  be the best-known objective value for that instance considering all the models. For the set of instances,  $\mathcal{I}$ , the relative error and mean relative error are computed in Eqs. (7) and (8). If a model did not find a feasible solution by a given time, the  $\text{MIP}_{UB}$  value is used to calculate a non-infinite measure.

$$RE(t, m, i) = \frac{obj_{t,m,i} - obj_i^*}{obj_i^*} \quad (7)$$

$$MRE(t, m) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} RE(t, m, i) \quad (8)$$

## 5.2 Results

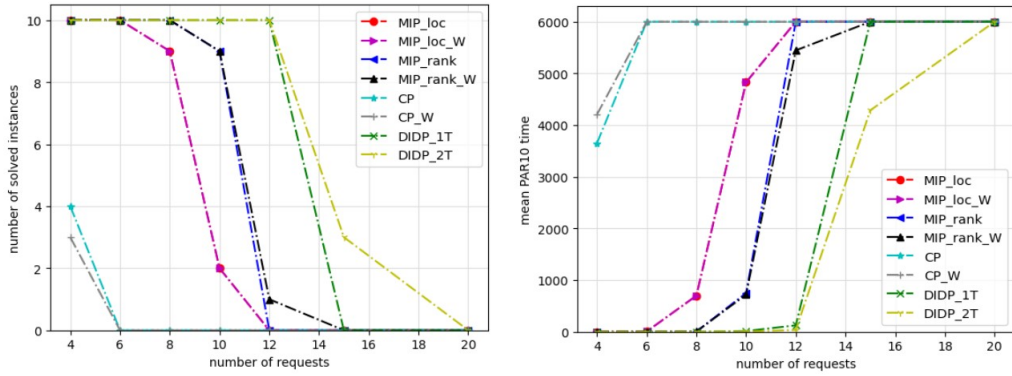
Figures 8a and 8b show the number of solved instances (i.e., proved infeasible or optimal) and mean PAR10 times for all the models. We do not include  $\text{MIP}_{UB}$  as it is incomplete, however for each model, its run-time is less than 0.02s. The run times for  $\text{MIP}_{loc\_W}$ ,  $\text{MIP}_{rank\_W}$ , and  $\text{CP}_W$  models, include the warm-start time.

The DIDP models solved all of the instances with 12 or fewer requests, with  $\text{DIDP}_{2T}$  performing slightly better than  $\text{DIDP}_{1T}$  for instances of size 15 as it could solve three instances compared to none for  $\text{DIDP}_{1T}$ . Neither CP nor  $\text{CP}_W$  were able to solve any instances of size larger than 6 while the MIP models scaled up to size 10 or 12. There was one instance of size 4 that  $\text{CP}_W$  could not prove optimality, but CP could.

In terms of solution time, the DIDP models were the fastest and CP models were the slowest. For the MIP models,  $\text{MIP}_{rank\_W}$  performed slightly better than  $\text{MIP}_{rank}$  in terms of both the number of solved instances and mean solution time. For one instance of size 12,  $\text{MIP}_{rank\_W}$  proved optimality where  $\text{MIP}_{rank}$  could not.

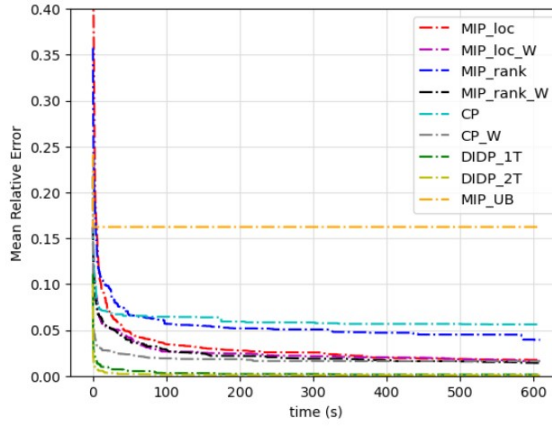
The MRE graph is shown in Figure 8c.  $\text{DIDP}_{2T}$  returns the best solutions and finds those best solutions within a few 10s of seconds. Up to 300s,  $\text{CP}_W$  outperformed  $\text{MIP}_{rank\_W}$ ,  $\text{MIP}_{loc\_W}$ ,  $\text{MIP}_{loc}$  but after that point, their solution qualities are very similar. The solution qualities returned by CP are the worst after 100 seconds. However, the use of  $\text{MIP}_{UB}$  as a warm start substantially improves CP quality especially for short run times. The performance of the  $\text{MIP}_{loc}$  and  $\text{MIP}_{loc\_W}$  models was very similar, however, the  $\text{MIP}_{loc\_W}$  model returned slightly better solution qualities than  $\text{MIP}_{loc}$ , especially before 200s. As we expected, the solutions found by the incomplete  $\text{MIP}_{UB}$  are substantially worse than other models.

Overall, DIDP models performed better than MIP and CP, and in particular  $\text{DIDP}_{2T}$  performed best in terms of the number of solved instances and average time to solve the instances. We hypothesize that DIDP outperforms other models due to the combination of tight capacity constraints and the precedence constraints induced by the pickup-and-delivery structure. DIDP uses these constraints to prune many transitions and, thus, reduce the search space. This result is consistent with previously observed behavior of DIDP on constrained routing problems [15] and suggests an opportunity for research to understand model characteristics that correlate with strong DIDP performance compared to other optimization approaches.



(a) Number of instances solved to optimality.

(b) Mean PAR10 time to solve instances.



(c) Comparison of MRE of all the models.

■ Figure 8 Performance of MIP, CP and DIDP models.

## 6 Discussion

The contributions of this work are the introduction of a novel pickup-and-delivery problem inspired by air services in northern Canada, the creation and evaluation of six optimization models in three different frameworks, and the further demonstration that the recently proposed domain-independent dynamic programming approach can out-perform incumbent techniques in a model-and-solve paradigm.

While DP models are inherently state-based, the DIDP formalism provides a novel avenue for constraint-based problem solving with connections to early ideas in CP (e.g., [6]). The DIDP models for PD-SRP are unusual as DP models due to the extensive, constraint-based, limitations on transitions (i.e., Eqs. (5c) and (5d)). While such limitations are key to strong DP performance, they are typically procedurally implemented in a problem-specific DP search algorithm. In DIDP, in contrast, constraint reasoning is used to prune transitions based on the values of state variables rather than pruning variable domains based on partial assignments. We believe that understanding this difference and developing constraint-based reasoning for this context is a fruitful research direction for CP.

Our study has a number of limitations and opportunities for further research:

- In the definition of PD-SRP, we discretized cargo into identical boxes with one size dimension (i.e., weight). In reality, cargo can take many forms from boxes of different

sizes and weights to baggage in various forms. Minimally, the volume of cargo needs to be represented. More generally, the problem should address the four-dimensional (i.e., volume plus weight) packing of heterogeneous cargo.

- We made the assumption that passengers do not have travel time restrictions. However, as a potential avenue for future research it would be interesting to incorporate additional constraints regarding how long a single passenger can be stowed in the aircraft or how long they can wait to be picked up.
- As is common in OR literature on transportation problems, our objective function is the minimization of the travel distance. A more realistic objective would represent aspects such as time and fuel consumption as well as handling and storage costs for seats.
- Most airlines run regular services with defined timetables and routings. Preliminary work indicates that determining seat exchanges is an easy problem when routes are decided. If this result bears out, there are two implications. First, we may have tools to deal with harder aspects of the real world problem including multiple aircraft, uncertain and dynamically changing demand (e.g., due to extreme weather in Canada's north), and strategic decisions about timetable creation, seat inventory, and aircraft capacities. Second, even with the version of PD-SRP presented here, we may be able to scale by exploiting the “easy” seat exchange part of the problem through Benders decomposition [2].
- Although, in this study, our focus was to design simple models that can be used “off the shelf”, it is interesting to investigate sophisticated custom-constraint CP models in the future development of this work to see if they outperform the currently developed MIP and CP models.

## 7 Conclusion

This paper studied a novel pickup and delivery transportation problem with reconfigurable capacities, a problem inspired by air service in northern Canada. We defined the problem formally and developed six models in three different modeling formalisms: constraint programming, mixed integer programming, and domain-independent dynamic programming. We compared the performance of the models on a set of randomly generated instances. MIP and CP models were solved with commercial solvers, the DIDP model was solved using the recently developed domain-independent dynamic programming solver [15].

Our results show that domain-independent dynamic programming models are the fastest in both finding high-quality feasible solutions to problem instances and in solving them to optimality. For large instances, when the number of requests is greater than 15, even DIDP models were not able to solve the instances to the optimality. Although in general, MIP models were faster to find feasible solutions than CP, for short run times, CP found better solutions than both of the MIP models.

Our future work will study generalizations of the problem by considering multiple aircraft and more realistic representation of cargo size and aircraft capacity. We have also embarked on a study of the decomposition of the problem both to better fit the real-world use case where routes are often predefined and to exploit the computational advances of the mathematical structure of the decomposition.

---

## References

- 1 Maria Battarra, Jean-François Cordeau, and Manuel Iori. Chapter 6: pickup-and-delivery problems for goods transportation. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 161–191. SIAM, 2014.

## 17:16 Optimization Models for PDPs with Reconfigurable Capacities

- 2 J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.
- 3 Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.
- 4 Irina Dumitrescu, Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121:269–305, 2010.
- 5 Pablo Factorovich, Isabel Méndez-Díaz, and Paula Zabala. Pickup and delivery problem with incompatibility constraints. *Computers & Operations Research*, 113:104805, 2020.
- 6 M. S. Fox, N. Sadeh, and C. Baykan. Constrained heuristic search. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 309–316, 1989.
- 7 Maria Gabriela S Furtado, Pedro Munari, and Reinaldo Morabito. Pickup and delivery problem with time windows: a new compact two-index formulation. *Operations Research Letters*, 45(4):334–341, 2017.
- 8 Jonas Hatzenbühler, Erik Jenelius, Győző Gidófalvi, and Oded Cats. Multi-purpose pickup and delivery problem for combined passenger and freight transport. *arXiv preprint arXiv:2210.05700*, 2022.
- 9 Tino Henke, M Grazia Speranza, and Gerhard Wäscher. The multi-compartment vehicle routing problem with flexible compartment sizes. *European Journal of Operational Research*, 246(3):730–743, 2015.
- 10 Sin C Ho, Wai Yuen Szeto, Yong-Hong Kuo, Janny MY Leung, Matthew Petering, and Terence WH Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421, 2018.
- 11 Alexander Hübner and Manuel Ostermeier. A multi-compartment vehicle routing problem with loading and unloading costs. *Transportation Science*, 53(1):282–300, 2019.
- 12 Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *International conference on principles and practice of constraint programming*, pages 454–469. Springer, 2011.
- 13 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.
- 14 Ryo Kuroiwa and J. Christopher Beck. Domain-independent dynamic programming: Generic state space search for combinatorial optimization. *Proceedings of the International Conference on Automated Planning and Scheduling*, 33(1):236–244, July 2023.
- 15 Ryo Kuroiwa and J. Christopher Beck. Solving domain-independent dynamic programming problems with anytime heuristic search. *Proceedings of the International Conference on Automated Planning and Scheduling*, 33:245–253, July 2023.
- 16 Jin Li, Qihui Lu, and Peihua Fu. Carbon footprint management of road freight transport under the carbon emission trading mechanism. *Mathematical Problems in Engineering*, 2015, 2015.
- 17 Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.
- 18 Andrea Mor and Maria Grazia Speranza. Vehicle routing problems over time: a survey. *Annals of Operations Research*, 314(1):255–275, 2022.
- 19 Salma Naccache, Jean-François Côté, and Leandro C Coelho. The multi-pickup and delivery problem with time windows. *European Journal of Operational Research*, 269(1):353–362, 2018.

- 20 Manuel Ostermeier, Tino Henke, Alexander Hübner, and Gerhard Wäscher. Multi-compartment vehicle routing problems: State-of-the-art, modeling framework and future directions. *European Journal of Operational Research*, 292(3):799–817, 2021.
- 21 Sophie N Parragh, Karl F Doerner, and Richard F Hartl. A survey on pickup and delivery models part ii: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58:81–117, 2006.
- 22 Yuan Qu and Jonathan F Bard. The heterogeneous pickup and delivery problem with configurable vehicle capacity. *Transportation Research Part C: Emerging Technologies*, 32:1–20, 2013.
- 23 Oscar Tellez, Samuel Vercraene, Fabien Lehuédé, Olivier Péton, and Thibaud Monteiro. The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity. *Transportation Research Part C: Emerging Technologies*, 91:99–123, 2018.
- 24 Marjolein Veenstra, Kees Jan Roodbergen, Iris FA Vis, and Leandro C Coelho. The pickup and delivery traveling salesman problem with handling costs. *European Journal of Operational Research*, 257(1):118–132, 2017.
- 25 Weixiong Zhang. Complete anytime beam search. In *AAAI/IAAI*, pages 425–430, 1998.





# Preprocessing in SAT-Based Multi-Objective Combinatorial Optimization

Christoph Jabs   




HIIT, Department of Computer Science, University of Helsinki, Finland

Jeremias Berg   

HIIT, Department of Computer Science, University of Helsinki, Finland

Hannes Ihalainen  

HIIT, Department of Computer Science, University of Helsinki, Finland

Matti Järvisalo   

HIIT, Department of Computer Science, University of Helsinki, Finland

---

## Abstract

Building on Boolean satisfiability (SAT) and maximum satisfiability (MaxSAT) solving algorithms, several approaches to computing Pareto-optimal MaxSAT solutions under multiple objectives have been recently proposed. However, preprocessing in (Max)SAT-based multi-objective optimization remains so-far unexplored. Generalizing clause redundancy to the multi-objective setting, we establish provably-correct liftings of MaxSAT preprocessing techniques for multi-objective MaxSAT in terms of computing Pareto-optimal solutions. We also establish preservation of Pareto-MCSes – the multi-objective lifting of minimal correction sets tightly connected to optimal MaxSAT solutions – as a distinguishing feature between different redundancy notions in the multi-objective setting. Furthermore, we provide a first empirical evaluation of the effect of preprocessing on instance sizes and multi-objective MaxSAT solvers.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial optimization; Theory of computation → Constraint and logic programming

**Keywords and phrases** maximum satisfiability, multi-objective combinatorial optimization, preprocessing, redundancy

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.18

**Supplementary Material** *Software (Source Code)*: <https://bitbucket.org/coreo-group/mo-prepro> archived at `swh:1:dir:57e426c8966cdb2e20db6631aaa2f0cdd5d4cfd4`

**Funding** Work financially supported by Academy of Finland under grants 322869, 342145 and 356046.

**Acknowledgements** The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

## 1 Introduction

Boolean satisfiability (SAT) solving [7] is arguably a noticeable success story of constraint programming. The impact of SAT solvers goes beyond merely deciding satisfiability. Incremental use of SAT solvers [13] today enables efficiently solving, e.g., hard optimization problems via maximum satisfiability (MaxSAT) [1]. While MaxSAT allows for finding optimal solutions in terms of a single objective function, practical applications have motivated various algorithmic advances and non-trivial generalizations of MaxSAT solving techniques to optimization under multiple objectives [43, 40, 10, 25, 20, 11]. These algorithms allow for computing one or several of the so-called Pareto-optimal solutions of multi-objective MaxSAT instances, i.e., solutions in which no objective can be improved without negatively affecting the value of another objective.



© Christoph Jabs, Jeremias Berg, Hannes Ihalainen, and Matti Järvisalo;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 18; pp. 18:1–18:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preprocessing has become a central part of the SAT solving pipeline [8], pruning the instance through applying complex combinations of different inference and simplification rules based on fundamental notions of (clause) redundancy. Motivated by its success in SAT, preprocessing in MaxSAT solving, through both extensions of SAT-based simplification techniques [3], and novel MaxSAT-specific techniques [5, 23, 39], is becoming increasingly popular and better understood, especially through recent work generalizing fundamental notions of redundancy in SAT [29, 28, 21, 22] to MaxSAT [24]. The MaxSAT liftings of redundancy notions allow for uniformly establishing the formal correctness of a wide range of MaxSAT preprocessing techniques [24, 4].

The advances in SAT and MaxSAT preprocessing, together with the recent advances in extending the reach of SAT-based approaches to multi-objective combinatorial optimization, call for studying fundamental and practical aspects of preprocessing in multi-objective settings. So-far preprocessing for (Max)SAT-based multi-objective optimization remains unexplored, with several open research questions. Developing correct liftings of MaxSAT preprocessing techniques to multi-objective settings, where Pareto-optimal solutions are sought for, calls for redundancy notions in order to uniformly capture the correctness of such liftings. In analogy to work analysing the power of different redundancy notions in SAT and more recently in MaxSAT, understanding the relationship between different redundancy notions in the multi-objective setting is also fundamentally relevant. From a more practical perspective, the effect of preprocessing for multi-objective problems in terms of simplifications achieved and solver runtimes has also not been thoroughly explored.

We make contributions to each of these questions. We provide redundancy notions for the multi-objective setting based on the notions of reconstructible and literal-reconstructible clauses, allowing for establishing the correctness of a large number of preprocessing techniques for multi-objective MaxSAT in terms of computing Pareto-optimal solutions. Additionally, we identify the preservation of Pareto-MCSEs (the multi-objective lifting of minimal correction sets tightly connected to Pareto-optimal solutions [42]) as a distinguishing feature between the two proposed redundancy notions. We also consider liftings of MaxSAT preprocessing techniques which alter in a controlled way the objective functions at hand and thereby cannot be directly captured by the clause redundancy notions. Putting these preprocessing techniques lifted to the multi-objective setting into practice, we provide a first preprocessor implementation for multi-objective MaxSAT, and perform a first empirical evaluation of the effect of preprocessing both in terms of instance size reductions achieved and runtimes of recently proposed approaches to multi-objective MaxSAT solving.

## 2 Multi-Objective MaxSAT

For a Boolean variable  $x$  there are two literals,  $x$  and  $\neg x$ . A clause  $C$  is a set (or disjunction) of literals and a (CNF) formula  $F$  a set (or conjunction) of clauses. A (truth) assignment  $\tau$  assigns variables to truth values 0 (false) or 1 (true). Assignments are extended to literals  $l$ , clauses  $C$ , and formulas  $F$ , in the standard way:  $\tau(\neg l) = 1 - \tau(l)$ ,  $\tau(C) = \max\{\tau(l) \mid l \in C\}$ , and  $\tau(F) = \min\{\tau(C) \mid C \in F\}$ , defining semantics for CNF formulas. An assignment  $\tau$  is a solution to a CNF formula  $F$  if  $\tau(F) = 1$ ;  $\tau$  is complete for  $F$  if  $\tau$  assigns a value to all variables in  $F$ , and otherwise partial for  $F$ . With slight abuse of notation, an assignment  $\tau$  can be viewed as the set of the literals it assigns to 1. Then  $\tau(x) = 1$  ( $\tau(x) = 0$ ) is shorthand for  $x \in \tau$  ( $\neg x \in \tau$ ),  $\neg C$  for  $\{\neg l \mid l \in C\}$ , and  $\tau \supset \neg C$  means that  $\tau$  falsifies a clause  $C$ .

We focus on the following natural generalization of the maximum satisfiability (MaxSAT) problem to multi-objective combinatorial optimization [44, 17, 15]. An instance  $\mathcal{I} = (F, O)$  of multi-objective MaxSAT (MO-MaxSAT) consists of a CNF formula  $F$ , the clauses of which

need to be satisfied by any solution to the instance, and a tuple  $O = (O_1, \dots, O_p)$  of  $p$  linear objective functions with positive coefficients over literals (or equivalently, pseudo-Boolean expressions) under minimization. We denote the set of literals appearing in  $O_i$  by  $\mathcal{B}_i(\mathcal{I})$  and the set of literals appearing in at least one of the objectives by  $\mathcal{B}(\mathcal{I}) = \bigcup_{i=1}^p \mathcal{B}_i(\mathcal{I})$ . Furthermore, we denote by  $c_i(l)$  the coefficient of literal  $l$  in  $O_i$ . If  $l$  does not appear in  $O_i$ , then  $c_i(l) = 0$ .

The cost  $O(\tau) = (O_1(\tau), \dots, O_p(\tau))$  of a solution  $\tau$  to  $\mathcal{I}$  (i.e., a solution to  $F$ ) is obtained by evaluating each objective under  $\tau$ . If  $\tau$  is not a solution of  $F$ , we let  $O(\tau) = (\infty, \dots, \infty)$ . As a central notion of optimality in the multi-objective setting in general, we focus on Pareto-optimality, which is based on the following domination relation between solutions.

► **Definition 1** ((Weak) Domination). *Consider two solutions  $\tau_1$  and  $\tau_2$  with costs  $O(\tau_1) = (O_1(\tau_1), \dots, O_p(\tau_1))$  and  $O(\tau_2) = (O_1(\tau_2), \dots, O_p(\tau_2))$ . The solution  $\tau_1$  weakly dominates  $\tau_2$  (denoted  $\tau_1 \preceq \tau_2$ ) if  $O_i(\tau_1) \leq O_i(\tau_2)$  holds for all  $i = 1, \dots, p$ . If additionally  $O_i(\tau_1) < O_i(\tau_2)$  for some  $i$ , then  $\tau_1$  dominates  $\tau_2$  (denoted  $\tau_1 \prec \tau_2$ ).*

Intuitively, the solution  $\tau_1$  weakly dominates another solution  $\tau_2$  if it is not worse in any objective. We use  $\tau_1 \not\preceq \tau_2$  to denote that  $\tau_1$  does not weakly dominate  $\tau_2$ . Note that domination is not a total order on solutions, i.e.,  $\tau_1 \not\preceq \tau_2$  does not generally imply  $\tau_2 \prec \tau_1$  or  $\tau_2 \preceq \tau_1$ .

A partial assignment  $\tau^p$  dominates another (partial) assignment  $\delta^p$  if for every extension  $\delta \supset \delta^p$  there is an extension  $\tau \supset \tau^p$  that dominates  $\delta$ . A solution  $\tau$  to an MO-MaxSAT instance  $\mathcal{I}$  is Pareto-optimal<sup>1</sup> if  $\tau$  is not dominated by any other solution to  $\mathcal{I}$ .

The notion of the non-dominated set of an MO-MaxSAT instance characterizes the solutions of interest in terms of their (non-dominated) costs.

► **Definition 2** (Non-dominated set). *The non-dominated set  $\text{non-dominated}(\mathcal{I}) = \{O(\tau) \mid \tau \text{ is Pareto-optimal}\}$  of an MO-MaxSAT instance  $\mathcal{I} = (F, O)$  consists of the costs of the Pareto-optimal solutions of  $\mathcal{I}$ .*

Practical algorithm for computing the non-dominated set of a given MO-MaxSAT instance also provide for each cost  $o \in \text{non-dominated}(\mathcal{I})$  a Pareto-optimal solution having cost  $o$ . It is worth noting that for an  $o \in \text{non-dominated}(\mathcal{I})$  there may be more than one Pareto-optimal solution with cost  $o$  and that for a single-objective MaxSAT instance  $\mathcal{I}$  the set  $\text{non-dominated}(\mathcal{I})$  consists of the optimal (minimum) cost of  $\mathcal{I}$ .

### 3 Clause Redundancy in MO-MaxSAT

Preprocessing an MO-MaxSAT instance  $\mathcal{I}$  refers to the iterative application of a set of preprocessing techniques (inference/simplification rules) on  $\mathcal{I}$ , resulting in a preprocessed instance  $\mathcal{P}(\mathcal{I})$  for which  $\text{non-dominated}(\mathcal{I}) = \text{non-dominated}(\mathcal{P}(\mathcal{I}))$ . In other words, correctness of preprocessing requires that the non-dominated set of the original  $\mathcal{I}$  does not change under the preprocessing techniques applied.

As fundamental notions for capturing, establishing the correctness of, and analysing the strengths of different MO-MaxSAT preprocessing techniques, we propose several (clause) redundancy properties in MO-MaxSAT. These properties can be viewed as multi-objective counterparts of earlier proposed redundancy notions in SAT [29, 28, 21, 22] and most recently

<sup>1</sup> Sometimes in the literature also referred to as *efficient* or *non-inferior* [14].

in MaxSAT [24], with similar motivations. In contrast to SAT (where clause redundancy notions are required to preserve satisfiability) and similarly as in MaxSAT, clause redundancy notions are required to preserve optimal costs. Compared to MaxSAT, however, the multiple objectives and Pareto optimization require additional care.

For an MO-MaxSAT instance  $\mathcal{I} = (F, O)$  and a clause  $C$ ,  $\mathcal{I} \wedge C = (F \wedge C, O)$  is the instance obtained by adding  $C$  to  $\mathcal{I}$ . We begin with a general notion of redundancy for the problem of computing the non-dominated set in MO-MaxSAT.

► **Definition 3** (Redundant clauses). *A clause  $C$  is redundant wrt an MO-MaxSAT instance  $\mathcal{I}$  if  $\text{non-dominated}(\mathcal{I}) = \text{non-dominated}(\mathcal{I} \wedge C)$ .*

Note that this definition does not require that all Pareto-optimal solutions should be preserved.

We propose two refined redundancy notions which turn out to differ in strength and thereby in terms of the preprocessing techniques they capture. The notions are based on the following alternative characterization of redundancy that essentially states that a clause  $C$  is redundant if every solution that falsifies it is weakly dominated by some solution that satisfies  $C$ .

► **Proposition 4.** *A clause  $C$  is redundant wrt an instance  $\mathcal{I} = (F, O)$  iff, for any solution  $\tau \supset \neg C$  to  $\mathcal{I}$  that falsifies  $C$ , there is a witnessing assignment (or simply witness)  $\omega^\tau$  for which  $\omega^\tau(C) = 1$  and  $\omega^\tau \preceq \tau$ .*

**Proof.** We prove each of the directions separately.

*$C$  is redundant  $\Rightarrow$  a witness exists:* Consider a solution  $\tau \supset \neg C$  to  $\mathcal{I}$ . Then there exists a Pareto-optimal solution  $\delta \preceq \tau$  (we can pick  $\delta = \tau$  if  $\tau$  is Pareto-optimal). Since  $\text{non-dominated}(\mathcal{I}) = \text{non-dominated}(\mathcal{I} \wedge C)$  (as  $C$  is redundant), there is a solution  $\omega^\delta$  to  $\mathcal{I} \wedge C$  with  $O(\tau) = O(\omega^\delta)$ . Such  $\omega^\delta$  satisfies  $C$  and weakly dominates  $\delta$ . Thus, it also weakly dominates  $\tau$ , fulfilling the requirements of the proposition.

*A witness exists  $\Rightarrow C$  is redundant:* To show that  $C$  is redundant according to Definition 3 we show that  $\text{non-dominated}(\mathcal{I} \wedge C) = \text{non-dominated}(\mathcal{I})$ . For the direction  $\text{non-dominated}(\mathcal{I} \wedge C) \subset \text{non-dominated}(\mathcal{I})$ , note that every Pareto-optimal solution  $\tau$  to  $\mathcal{I} \wedge C$  is also a solution to  $\mathcal{I}$ . Furthermore,  $\tau$  is also Pareto-optimal wrt  $\mathcal{I}$ . If this was not the case, by the assumption the solution  $\delta$  dominating  $\tau$  wrt  $\mathcal{I}$  would have a witness  $\omega^\delta$  dominating  $\tau$  wrt  $\mathcal{I} \wedge C$ . Since therefore every Pareto-optimal solution to  $\mathcal{I} \wedge C$  is also Pareto-optimal wrt  $\mathcal{I}$ , it follows that  $\text{non-dominated}(\mathcal{I} \wedge C) \subset \text{non-dominated}(\mathcal{I})$ .

For the other direction consider an element  $o \in \text{non-dominated}(\mathcal{I})$  and let  $\tau$  be a Pareto-optimal solution to  $\mathcal{I}$  for which  $O(\tau) = o$ . For the interesting case, assume  $\tau \supset \neg C$ , i.e., that it falsifies  $C$ . Then by the assumption  $\tau$  is weakly dominated by some witness  $\omega^\tau$  that satisfies  $C$ . Now  $O(\tau) = O(\omega^\tau) = o$  (as otherwise  $\tau$  would not be Pareto-optimal) demonstrating that  $o \in \text{non-dominated}(\mathcal{I} \wedge C)$  and thus that  $C$  is redundant. ◀

The (weakly) dominating witness  $\omega^\tau$  guaranteed by Proposition 4 for any redundant clause  $C$  might differ depending on the specific solution  $\tau$  that falsifies  $C$ . The redundancy notions of *reconstructible* and *literal-reconstructible* clauses we propose next are based on placing stronger requirements on this witness.

► **Definition 5** (Reconstructible clauses). *A clause  $C$  is reconstructible on the (partial) assignment  $\omega$  wrt an MO-MaxSAT instance  $\mathcal{I}$  if (i)  $\omega(C) = 1$ , and (ii)  $(\tau \setminus \neg\omega) \cup \omega \preceq \tau$  for every solution  $\tau \supset \neg C$  to  $\mathcal{I}$ .*

In words, a clause  $C$  is reconstructible wrt an MO-MaxSAT instance  $\mathcal{I}$  if there is a *single witnessing assignment*  $\omega$  that satisfies  $C$  and weakly dominates *all* solutions  $\tau$  that do not. Moreover, enforcing the partial assignment  $\omega$  in any such solution  $\tau$  allows for efficiently obtaining a solution to  $\mathcal{I}$  that satisfies  $C$  and weakly dominates  $\tau$ . For the corner case, note that if there are no solutions to  $\mathcal{I}$  that falsify  $C$ , then  $C$  is reconstructible on any witness.

The fact that reconstructible clauses are redundant follows directly from Proposition 4. The next example demonstrates that the converse does not hold. Central to the example is to note that a direct consequence of Definition 5 is that if  $C$  is reconstructible on the partial assignment  $\omega$ , then  $\omega$  weakly dominates the partial assignment  $\neg C$ .

► **Example 6.** Let  $\mathcal{I} = (F, (O_1, O_2))$  be an MO-MaxSAT instance with  $F = (a_1 \vee a_2) \wedge (b_1 \vee b_2) \wedge (a_1 \vee b_1) \wedge (a_1 \vee b_2) \wedge (a_2 \vee b_1) \wedge (a_2 \vee b_2)$ ,  $O_1 = a_1 + a_2$ , and  $O_2 = b_1 + b_2$ . Then  $\text{non-dominated}(\mathcal{I}) = \{(1, 2), (2, 1)\}$ , and the Pareto-optimal solutions are  $\tau_1 = \{a_1, \neg a_2, b_1, b_2\}$ ,  $\tau_2 = \{\neg a_1, a_2, b_1, b_2\}$ ,  $\tau_3 = \{a_1, a_2, \neg b_1, b_2\}$ , and  $\tau_4 = \{a_1, a_2, b_1, \neg b_2\}$ . Consider the clause  $C = (\neg a_2 \vee \neg b_2)$ . Since  $\tau_1$  and  $\tau_4$  are solutions to  $F \wedge C$ , adding  $C$  does not change the non-dominated set of the instance. Thus,  $C$  is redundant wrt  $\mathcal{I}$ . To see that  $C$  is not reconstructible we show that no partial assignment  $\omega$  that satisfies  $C$  weakly dominates  $\neg C$ . There are two possible candidates for such  $\omega$  (as  $C$  contains two literals):  $\omega_1 = \{\neg a_2\}$  and  $\omega_2 = \{\neg b_2\}$ . The only solution of  $\mathcal{I}$  that  $\omega_1$  can be extended to is  $\tau_1$ . However,  $\neg C$  can be extended to  $\tau_3$ , which is not weakly dominated by  $\tau_1$ . Similarly,  $\omega_2 = \{\neg b_2\}$  does not weakly dominate  $\tau_2 \supset \neg C$ , showing that  $\omega_2 \not\preceq \neg C$ .

Contrasting Example 6, the next proposition shows that the notions of (clause) redundancy according to Definition 3 and reconstructible clauses according to Definition 5 coincide for single-objective MaxSAT instances that have solutions.

► **Proposition 7.** *For a single-objective MaxSAT instance  $\mathcal{I} = (F, (O_1))$  with at least one solution  $\tau$  and clause  $C$ , it holds that  $C$  is reconstructible for  $\mathcal{I}$  iff  $C$  is redundant.*

**Proof (sketch).** For the non-trivial direction, assume that  $C$  is redundant. Then there is an optimal (minimum-cost) solution  $\tau^\circ$  to  $\mathcal{I}$  that satisfies  $C$ . As  $\mathcal{I}$  only has a single objective,  $\tau^\circ$  weakly dominates all solutions to  $\mathcal{I}$ . Therefore,  $C$  is reconstructible on  $\tau^\circ$ . ◀

As a further notion of redundancy, we consider *literal-reconstructible* clauses as a special case of reconstructible clauses where the witness is required to consist of a single literal. In Section 4 we discuss properties that literal-reconstructible clauses specifically satisfy and overview in Section 5.1 preprocessing techniques that can be characterized by adding and removing literal-reconstructible clauses.

► **Definition 8 (Literal-reconstructible clauses).** *A clause  $C$  is literal-reconstructible wrt an instance  $\mathcal{I} = (F, O)$  if either (i) all solutions to  $F$  satisfy  $C$ , or (ii) there is a non-objective literal  $l \in C \setminus \mathcal{B}(\mathcal{I})$  s.t. if  $\tau \supset \neg C$  is a solution to  $F$ , then  $\tau_l = (\tau \setminus \{\neg l\}) \cup \{l\}$  is a solution to  $F \wedge C$ . If condition (ii) holds, we say that  $C$  is literal-reconstructible on the literal  $l$ .*

Note that the definition of literal-reconstructible clauses does not explicitly require that  $\tau_l$  weakly dominates  $\tau$ , as this follows from  $l$  not being an objective literal. The following proposition states that literal-reconstructible clauses are redundant in terms of Definition 3.

► **Proposition 9.** *If a clause  $C$  is literal-reconstructible wrt an MO-MaxSAT instance  $\mathcal{I} = (F, (O_1, \dots, O_p))$ , then  $\text{non-dominated}(\mathcal{I}) = \text{non-dominated}(\mathcal{I} \wedge C)$ .*

**Proof (sketch).** For the interesting case, assume that there is a solution  $\tau \supset \neg C$  to  $\mathcal{I}$  that does not satisfy  $C$ . Let  $l \in C \setminus \mathcal{B}(\mathcal{I})$  be the literal on which  $C$  is literal-reconstructible and consider the solution  $\tau_l = (\tau \setminus \{\neg l\}) \cup \{l\}$ . Then by the assumption  $\tau_l$  is a solution to  $\mathcal{I} \wedge C$  and as  $l \notin \mathcal{B}(\mathcal{I})$  we have that  $O_i(\tau_l) \leq O_i(\tau)$  for all objectives, i.e., for each  $i = 1, \dots, p$ . As  $\tau_l$  is a solution to both  $\mathcal{I}$  and  $\mathcal{I} \wedge C$ , the result follows.  $\blacktriangleleft$

A clause that is literal-reconstructible on  $l$  is also reconstructible on the witness  $\omega = \{l\}$ . The following example shows that the opposite does not hold in general, i.e., there are reconstructible clauses that are not literal-reconstructible.

► **Example 10.** Consider the MO-MaxSAT instance  $\mathcal{I} = (F, (O_1))$  with  $F = (a_1 \vee a_2)$  and  $O_1 = a_1 + a_2$ . The clause  $C = (\neg a_1)$  is reconstructible on the witness  $\omega = \{\neg a_1, a_2\}$ . The assignment  $\tau = \{a_1, \neg a_2\}$  is a solution to  $F$  but does not satisfy  $C$ . The only literal  $l \in C \setminus \mathcal{B}(\mathcal{I})$  is  $\neg a_1$ , but  $(\{a_1, \neg a_2\} \setminus \{a_1\}) \cup \{\neg a_1\} = \{\neg a_1, \neg a_2\}$  is not a solution to  $F \wedge C$ . It follows that  $C$  is not literal-reconstructible.

The relative generality of these three MO-MaxSAT redundancy notion can be summarized as follows. For any MO-MaxSAT instance  $\mathcal{I}$ , the set of redundant clauses  $\text{Red}(\mathcal{I})$  is a superset of the set of reconstructible clauses  $\text{Rec}(\mathcal{I})$ , and  $\text{Rec}(\mathcal{I})$  is a superset of the set of literal-reconstructible clauses  $\text{LRec}(\mathcal{I})$ . Furthermore, there are clauses that are reconstructible but not literal-reconstructible (i.e., there is an instance  $\mathcal{I}$  for which  $\text{Rec}(\mathcal{I}) \supsetneq \text{LRec}(\mathcal{I})$ ), and clauses that are redundant (in terms of Definition 3) that are not reconstructible (i.e., there is an instance  $\mathcal{I}'$  for which  $\text{Red}(\mathcal{I}') \supsetneq \text{Rec}(\mathcal{I}')$ ). In contrast to single-objective MaxSAT, the last statement holds also for instances that have solutions as for a single objective instance  $\mathcal{I}''$  we have that  $\text{Red}(\mathcal{I}'') \neq \text{Rec}(\mathcal{I}'')$  if and only if  $\mathcal{I}''$  does not have solutions.

As a side-remark, literal-reconstructible clauses are related to so-called cost literal propagation redundant clauses [24] recently proposed for (single-objective) MaxSAT: any cost literal propagation redundant clause is literal-reconstructible under a single objective. The opposite holds only when conditions (i) and (ii) in Definition 8 can be deterministically checked by standard Boolean constraint propagation on clauses (i.e., unit propagation). Intuitively, literal-reconstructible clauses extend and slightly generalize the concept of cost literal propagation redundant clauses for the multi-objective setting.

## 4 Redundancy and Pareto-MCSes

We move on to analysing the effect that adding (literal-)reconstructible clauses to an MO-MaxSAT instance has on the solution space in terms of so-called Pareto minimal correction sets (Pareto-MCSes) [42, 43], that – informally speaking – correspond to subset-minimal sets of objective literals that are assigned to 1 by at least one Pareto-optimal solution.

► **Definition 11 (Pareto-MCS).** Consider an MO-MaxSAT instance  $\mathcal{I} = (F, O)$ . A subset  $M \subset \mathcal{B}(\mathcal{I})$  of objective literals is a correction set if there is a solution  $\tau$  of  $\mathcal{I}$  that assigns  $\tau(l) = 0$  for every objective literal  $l$  not appearing in  $M$ .  $M$  is a minimal correction set (MCS) (or multi minimal correction subset as in [42]) if no  $M' \subsetneq M$  is a correction set. Finally,  $M$  is a Pareto-MCS if each solution  $\tau$  that assign  $\tau(l) = 0$  for every  $l \in \mathcal{B}(\mathcal{I}) \setminus M$  is Pareto-optimal. The set  $\text{ParetoMCS}(\mathcal{I})$  consists of the Pareto-MCSes of  $\mathcal{I}$ .

For some intuition, note that assigning an objective literal to 1 can be seen as falsifying a soft constraint. If  $M$  is an MCS or Pareto-MCS, then for any solution with  $\tau(l) = 0$  for every literal not in  $M$  we also have  $\tau(l') = 1$  for every literal  $l'$  in  $M$ . From this point of view, these definitions of MCSes align with the (arguably more classical) ones in terms of subset-minimal sets of soft constraints falsified by some solution. Specifically, if  $\mathcal{I}$  only has a single objective, this definition is identical to MCSes in single-objective MaxSAT [36].

The relationship between Pareto-optimal solutions, Pareto-MCSes and elements of the non-dominated set is not one-to-one. For a Pareto-MCS  $M$  of an MO-MaxSAT instance  $\mathcal{I} = (F, O)$ , there is at least one corresponding Pareto-optimal solution  $\tau$  to  $\mathcal{I}$ . The cost of each such  $\tau$  wrt each objective in  $O$  is the sum of the objective coefficients of the objective literals included in  $M$ . There can be multiple Pareto-optimal solutions that correspond to a Pareto-MCS which differ in how non-objective variables are assigned. Furthermore, for a single element (cost tuple) in the non-dominated set, there can be multiple corresponding Pareto-MCSes, since two different Pareto-MCSes can incur the same cost wrt each objective of  $\mathcal{I}$ . Hence, preserving the Pareto-MCSes of an input MO-MaxSAT instance  $\mathcal{I}$  is a sufficient *but not necessary* condition for preserving  $\text{non-dominated}(\mathcal{I})$ . For computing the non-dominated set, it suffices that at least one corresponding Pareto-MCS for each element in the non-dominated set is preserved.

We establish the fact that preservation of the set of Pareto-MCSes is a property of literal-reconstructible clauses, distinguishing this notion from the more general notion of reconstructible clauses which does not have this property. More precisely, the following summarizes the main theorem of this section: adding/removing literal-reconstructible clauses does not change the set of Pareto-MCSes.

► **Theorem 12.** *Assume that  $C$  is literal-reconstructible wrt an MO-MaxSAT instance  $\mathcal{I} = (F, O)$ . Then  $\text{ParetoMCS}(\mathcal{I}) = \text{ParetoMCS}(\mathcal{I} \wedge C)$ .*

A proof of Theorem 12 relies on showing that, given any Pareto-optimal solution  $\tau \supset \neg C$  of  $\mathcal{I}$  that does not satisfy  $C$ , the weakly-dominating (Pareto-optimal) witness  $\tau_l$  obtained by flipping the value of the literal  $l \in C \setminus \mathcal{B}(\mathcal{I})$  that  $C$  is literal-reconstructible on corresponds to the exact same Pareto-MCS as  $\tau$ . Toward formalizing this intuition, we show that if the negation of  $l$  is in any objective, then there is no Pareto-optimal solution that falsifies  $C$ .

► **Lemma 13.** *Let  $C$  be literal-reconstructible on  $l$  wrt  $\mathcal{I} = (F, O)$  and  $\neg l$  an objective literal, i.e.,  $\neg l \in \mathcal{B}(\mathcal{I})$ . Then there is no Pareto-optimal solution  $\tau \supset \neg C$  to  $\mathcal{I}$  that falsifies  $C$ .*

**Proof of Lemma 13.** As  $C$  is literal-reconstructible on  $l$ ,  $\tau' = (\tau \setminus \{\neg l\}) \cup \{l\}$  is a solution to  $\mathcal{I}$ . Because  $\neg l \in \mathcal{B}(\mathcal{I})$  and therefore at least one of the objectives evaluates to less for  $\tau'$  than for  $\tau$ ,  $\tau' \prec \tau$ . Therefore,  $\tau$  is not Pareto-optimal. ◀

With the inverse of Lemma 13 covering the (special) case of some Pareto-optimal solutions falsifying  $C$ , we turn to the proof of Theorem 12.

**Proof of Theorem 12.** If  $C$  is literal-reconstructible because every solution of  $\mathcal{I}$  satisfies  $C$ , the solutions and therefore the set of Pareto-MCSes of  $\mathcal{I}$  and  $\mathcal{I} \wedge C$  are the same. Otherwise, let  $C$  be literal-reconstructible on  $l$  and consider the following.

*$\text{ParetoMCS}(\mathcal{I}) \subset \text{ParetoMCS}(\mathcal{I} \wedge C)$ :* Let  $M \in \text{ParetoMCS}(\mathcal{I})$  and consider the Pareto-optimal solution  $\tau^M \supset \{\neg b \mid b \in \mathcal{B}(\mathcal{I}) \setminus M\}$  to  $\mathcal{I}$  that sets  $\tau(b) = 0$  for every objective literal  $b$  not in  $M$ . Since  $C$  is literal-reconstructible on  $l$ , there is a solution  $\delta$  to  $F \wedge C$  that weakly dominates  $\tau^M$ . If  $\tau^M$  satisfies  $C$ , then  $\delta = \tau^M$ . Otherwise,  $\delta = (\tau^M \setminus \{\neg l\}) \cup \{l\}$ , and since  $\tau^M$  is Pareto-optimal and falsifies  $C$ , by Lemma 13  $\neg l$  is not an objective literal. In both cases  $\delta$  corresponds to the same MCS ( $M$ ) as  $\tau^M$ . Furthermore,  $M$  must be a Pareto-MCS of  $\mathcal{I} \wedge C$  as any solution dominating  $\delta$  would also be a solution to  $\mathcal{I}$  and therefore  $M$  would not be a Pareto-MCS of  $\mathcal{I}$ .

*$\text{ParetoMCS}(\mathcal{I} \wedge C) \subset \text{ParetoMCS}(\mathcal{I})$ :* Given  $M \in \text{ParetoMCS}(\mathcal{I} \wedge C)$  and a Pareto-optimal  $\tau^M \supset \{\neg b \mid b \in \mathcal{B}(\mathcal{I}) \setminus M\}$  to  $\mathcal{I} \wedge C$ ,  $\tau^M$  is also Pareto-optimal for  $\mathcal{I}$  as any dominating solution could be reconstructed (by flipping the value of  $l$ ) into a solution to  $\mathcal{I} \wedge C$  dominating  $\tau^M$ . ◀



Contrasting Theorem 12, we show that a similar result cannot be obtained for reconstructible clauses.

► **Proposition 14.** *There is an MO-MaxSAT instance  $\mathcal{I}$  and a reconstructible clause  $C$  wrt  $\mathcal{I}$  for which  $\text{ParetoMCS}(\mathcal{I} \wedge C) \subsetneq \text{ParetoMCS}(\mathcal{I})$ .*

**Proof.** Consider the MO-MaxSAT instance  $\mathcal{I} = (F, (O_1, O_2))$  with  $F = (a_1 \vee b_1 \vee b_2)$ ,  $O_1 = a_1$ ,  $O_2 = b_1 + b_2$ , and  $C = (\neg b_2)$ . We have that  $\text{ParetoMCS}(\mathcal{I}) = \{\{a_1\}, \{b_1\}, \{b_2\}\}$  and  $\text{ParetoMCS}(\mathcal{I} \wedge C) = \{\{a_1\}, \{b_1\}\}$ .

Since the only clause in  $F$  and  $C$  are both satisfied by  $\omega = \{b_1, \neg b_2\}$ , every superset of  $\omega$  is a solution to  $F \wedge C$ . Furthermore, given a solution  $\tau$  to  $F$  that falsifies  $C$ , the solution  $\tau_\omega = (\tau \setminus \neg\omega) \cup \omega$  has  $O_1(\tau_\omega) = O_1(\tau)$  and  $O_2(\tau_\omega) \leq O_2(\tau)$ , hence  $\tau_\omega \preceq \tau$ . It follows that  $C$  is reconstructible on  $\omega$  wrt  $\mathcal{I}$ . ◀

This distinction between literal-reconstructible and reconstructible clauses in terms of the preservation of Pareto-MCSes provides two important insights.

Firstly, the fact that adding/removing literal-reconstructible clauses does not change the set of Pareto-MCSes implies that (single-objective) MaxSAT preprocessing techniques that can be viewed as sequences of adding and removing literal-reconstructible clauses are techniques that are “directly applicable” to MO-MaxSAT. In particular, it has been shown that the non-dominated set of an MO-MaxSAT instance  $\mathcal{I} = (F, O)$  can be computed by enumerating its Pareto-MCSes, which can in turn be achieved by enumerating the MCSes of the (single-objective) MaxSAT instance  $(F, O^m)$  with the single objective  $O^m = \sum_{O_i \in O} O_i$  that sums all objectives of  $\mathcal{I}$  [43]. Thus, any preprocessing technique for single-objective MaxSAT that preserves MCSes is directly applicable to MO-MaxSAT by applying it to  $(F, (O^m))$  and using the preprocessed formula in the MO-MaxSAT instance. The correctness of such techniques – which we will overview shortly – can either be directly argued on the MO-MaxSAT level by viewing them as sequences of adding and removing literal-reconstructible clauses, *or* by using (MCS-preserving) redundancy notions such as cost literal propagation redundancy on the level of single-objective MaxSAT. On the other hand, preprocessing techniques captured by reconstructible clauses but which cannot be captured by literal-reconstructible clauses – as detailed later on – go beyond preserving Pareto-MCSes, having the ability to eliminate Pareto-MCSes that are redundant in terms of the non-dominating set. Hence, reconstructible clauses are key in capturing the correctness of such techniques in a uniform way.

## 5 Preprocessing for MO-MaxSAT

We proceed with overviews a range of preprocessing techniques for MO-MaxSAT, lifting earlier-proposed techniques from single-objective MaxSAT (some of which originate from SAT) to the multi-objective setting. We detail in short which of the techniques are captured by the notions of reconstructible or literal-reconstructible clauses by simulating the techniques via sequences of additions and removals of redundant clauses of a specific type.

### 5.1 Preprocessing Techniques Captured by Literal-Reconstructible Clauses

First, we shortly recall well-known single-objective MaxSAT preprocessing techniques that are known to preserve MCSes [24]. We note again that the correctness of these techniques follows from the previously mentioned fact that each of them preserve MCSes in single-objective

MaxSAT, which further follows naturally from earlier work on capturing these techniques in the setting of SAT solving via redundancy notions developed for SAT. Alternatively – as we will detail in the following – the correctness arguments can be directly made on the MO-MaxSAT level by showing that each technique can be simulated via removing and adding literal-reconstructible clauses. For the following list of techniques, let  $\mathcal{I} = (F, O)$  be an MO-MaxSAT instance with  $O = (O_1, \dots, O_p)$ .

**Bounded Variable Elimination (BVE) [41, 12].** BVE as arguably the most important SAT preprocessing technique allows eliminating a non-objective variable  $x \notin \mathcal{B}(\mathcal{I})$  (and  $\neg x \notin \mathcal{B}(\mathcal{I})$ ) from  $\mathcal{I}$ . A step of BVE on  $\mathcal{I}$  and  $x$  results in the MO-MaxSAT instance  $\text{bve}(\mathcal{I}, x) = (F \cup F_{\text{res}} \setminus (F_x \cup F_{\neg x}), O)$ , where  $F_x = \{C \in F \mid x \in C\}$ ,  $F_{\neg x} = \{C \in F \mid \neg x \in C\}$  are the sets of clauses containing  $x$  and  $\neg x$ , respectively, and  $F_{\text{res}} = \{(A \vee B) \mid (A \vee x), (B \vee \neg x) \in F\}$  is the set of all non-tautological resolvents on  $x$  of the clauses in  $F$ , bounded in practice to eliminate variables when this decreases the number of clauses. Working directly on the MO-MaxSAT level,  $\text{bve}(\mathcal{I}, x)$  can be obtained from  $\mathcal{I} = (F, O)$  by a sequence of additions and removals of literal-reconstructible clauses as follows. First add  $F_{\text{res}}$  to  $F$  which does not change the non-dominated set because every clause in  $F_{\text{res}}$  is satisfied by any solution to  $F$  and therefore literal-reconstructible wrt  $\mathcal{I}$  and any instance obtained by adding clauses from  $F_{\text{res}}$  to  $\mathcal{I}$ . Note that for every  $(A \vee B) \in F_{\text{res}}$ , by construction of  $F_{\text{res}}$ ,  $(A \vee x), (B \vee \neg x) \in F$ . Second, remove the clauses  $F_x \cup F_{\neg x}$  from the intermediate instance  $\mathcal{I}' = (F \cup F_{\text{res}}, O)$ : every clause in  $F_x$  (resp.  $F_{\neg x}$ ) is literal-reconstructible on  $x$  (resp.  $\neg x$ ) wrt  $\mathcal{I}'$  and any instance obtained by removing clauses in  $F_x \cup F_{\neg x}$  from  $\mathcal{I}'$ .

**Blocked Clause Elimination (BCE) [27].** BCE removes blocked clauses [33]: a clause  $(C \vee l) \in F$  is blocked on a literal  $l \notin \mathcal{B}(\mathcal{I})$  if for every clause  $(D \vee \neg l) \in F$  containing  $\neg l$ , the resolvent  $(C \vee D)$  is a tautology. Note that a clause blocked on  $l$  is literal-reconstructible on  $l$ .

**Subsumption Elimination (SE).** A clause  $C \in F$  is subsumed by another clause  $D \in F$  if  $D \subset C$ . One step of SE removes a subsumed clause  $C$ , resulting in the instance  $\text{se}(\mathcal{I}, C) = (F \setminus \{C\}, O)$ . Note that any solution to  $\text{se}(\mathcal{I}, C)$  also satisfies  $C$ , and thus  $C$  is literal-reconstructible wrt  $\text{se}(\mathcal{I}, C)$ .

**Unit Propagation (UP).** Given a non-objective literal  $l \notin \mathcal{B}(\mathcal{I})$  and a unit clause  $(l) \in F$ , unit propagation removes each clause  $C \in F$  containing  $l$  ( $l \in C$ ) and removes the negation  $\neg l$  from the remaining clauses. Similarly as in SAT, UP can be viewed as an application of BVE on  $l$  (to remove negation  $\neg l$  from all clauses) followed by an application SE (to remove resolvents introduced by BVE).

**Self-Subsuming Resolution (SSR) [38, 12].** Given two clauses  $(x \vee A), (\neg x \vee B) \in F$  s.t.  $A \subset B$ ,  $x \notin \mathcal{B}(\mathcal{I})$ , and  $\neg x \notin \mathcal{B}(\mathcal{I})$ , a step of SSR results in the formula  $\text{ssr}(\mathcal{I}, (\neg x \vee B)) = ((F \cup \{B\}) \setminus \{(\neg x \vee B)\}, O)$ . Note that  $B$  is literal-reconstructible wrt  $\mathcal{I}$  and that  $(\neg x \vee B)$  is subsumed in  $F \wedge B$ .

**Failed Literal Elimination (FLE) and TrimMaxSAT.** FLE [46, 18, 34] and TrimMaxSAT [39] allow for detecting unit clauses entailed by  $F$ , i.e., clauses satisfied by every solution to  $\mathcal{I}$ . Such clauses are by definition literal-reconstructible.

**Equivalent Literal Substitution (ELS) [35, 9, 45].** Two literals  $l_1, l_2$  are equivalent if  $\tau(l_1) = \tau(l_2)$  for every solution  $\tau$ . If neither literal nor their negation occur in objectives, equivalent literal substitution replaces every occurrence of  $l_2$  with  $l_1$  and  $\neg l_2$  with  $\neg l_1$ . Viewed in terms of literal-reconstructible clauses, first add the clauses in which  $l_2$  ( $\neg l_2$ ) has been replaced and then remove clauses containing  $l_2$  ( $\neg l_2$ ). Both of these sets of clauses are literal-reconstructible wrt the instance they are added to / removed from because  $\tau(l_1) = \tau(l_2)$ .

## 5.2 Preprocessing Techniques Captured by Reconstructible Clauses

We now turn to techniques that do not preserve all Pareto-MCSes and therefore require a more general notion of redundancy. Specifically, we lift (group-)subsumed label elimination ((G)SLE) [5, 31] from MaxSAT, extending subsumption to objective literals, to MO-MaxSAT and show that it is captured by adding *reconstructible* clauses.

► **Definition 15.** An objective literal  $l \in \mathcal{B}(\mathcal{I})$  of an MO-MaxSAT instance  $\mathcal{I} = (F, (O_1, \dots, O_p))$  is subsumed if there is a group of objective literals  $S \subset \mathcal{B}(\mathcal{I})$  for which (i)  $c_i(l) \geq c_i(\neg l) + \sum_{s \in S} c_i(s)$  for all objectives ( $i = 1, \dots, p$ ), (ii) every clause  $C \in F$  that contains  $l$  also contains some literal  $s \in S$ , and (iii) every clause  $C \in F$  that contains the negation of any  $s \in S$  also contains  $\neg l$ .

Informally speaking, a step of GSLE on an MO-MaxSAT instance  $\mathcal{I}$  wrt a subsumed literal  $l$  fixes  $l = 0$ . More formally, it results in the instance  $\text{gsle}(\mathcal{I}, l) = \mathcal{I} \wedge (\neg l)$ .

In contrast to the preprocessing techniques discussed in the preceding subsection, GSLE cannot be lifted from single-objective MaxSAT to MO-MaxSAT by simply combining multiple objectives into a sum. To see this, consider the MO-MaxSAT instance from the proof of Proposition 14. When applying *single-objective* GSLE by summing the objectives as  $O_1 + O_2 = a_1 + b_1 + b_2$ ,  $a_1$  is subsumed by  $\{b_1\}$ . However, adding the clause  $(\neg a_1)$  removes  $(1, 0)$  from the non-dominated set.

The following example demonstrates that GSLE can remove Pareto-MCSes.

► **Example 16.** Consider the MO-MaxSAT instance in the proof of Proposition 14. According to Definition 15  $b_2$  is subsumed by  $\{b_1\}$ , hence  $\text{gsle}(\mathcal{I}, b_2) = (F \wedge (\neg b_2), (O_1, O_2))$ , and thus  $\text{ParetoMCS}(\text{gsle}(\mathcal{I}, b_2)) \subsetneq \text{ParetoMCS}(\mathcal{I})$

For an alternative proof of the fact that GSLE cannot be viewed as a sequence of adding/removing literal-reconstructible clauses, consider Example 10 where it was argued that the clause  $(\neg a_1)$  is reconstructible but not literal-reconstructible. Note that in the example  $a_1$  is subsumed by  $\{a_2\}$ , and hence applying GSLE on the instance wrt  $a_1$  would result in adding exactly the clause  $(\neg a_1)$  into the instance.

The correctness of GSLE for MO-MaxSAT follows by observing that it can be viewed as the addition of a reconstructible clause.

► **Proposition 17.** If  $l$  is subsumed in an MO-MaxSAT instance  $\mathcal{I} = (F, O)$ , then the clause  $C = (\neg l)$  is reconstructible wrt  $\mathcal{I}$ .

**Proof.** Let  $S \subset \mathcal{B}(\mathcal{I})$  be the group of literals that subsumes  $l$ , and  $\tau \supset \neg C$  with  $\tau(F) = 1$  a solution that falsifies  $C$ . Consider the witness  $\omega = S \cup \{\neg l\}$  and the solution  $\tau_\omega = (\tau \setminus \neg \omega) \cup \omega$ . Since all clauses that  $l$  appears in contain at least one literal  $s \in S$  (condition (ii) of Definition 15) and all clauses that a negated literal from  $S$  appears in also contain  $\neg l$  (condition (iii) of Definition 15),  $\tau_\omega$  is a solution to  $F \wedge C$ . Because  $l$  (note that  $\tau(l) = 1$ ) increases every objective more than all of  $\omega$  (condition (i) of Definition 15),  $\tau_\omega$  weakly dominates  $\tau$ . ◀

### 5.3 Preprocessing with Changes to Objectives

All techniques we have so far considered solely change the formula of an instance and not the objectives. However, towards practical preprocessing for MO-MaxSAT, we note that MaxSAT preprocessing techniques that may change the single objective in MaxSAT can also be lifted to MO-MaxSAT. These include unit propagation and equivalent literal substitution on objective literals, intrinsic at-most-ones, and binary core removal (BCR). Alike their MaxSAT counterparts, these liftings cannot be expressed directly as a sequence of additions and removals of redundant clauses due to the very fact that redundant clauses by definition do not change costs of instances.

**Unit Propagation on an Objective Literal.** In addition to removing all clauses containing  $l$  and removing  $\neg l$  from all clauses, unit propagation on an objective literal  $l \in \mathcal{B}(\mathcal{I})$  replaces the terms  $c_i(l) \cdot l$  with the respective constant  $c_i(l)$  in each objective  $O_i$  for  $i = 1, \dots, p$ . It is straightforward to see that by doing so costs of solutions are left unchanged.

**Equivalent Literal Substitution on Objective Literals.** An objective literal  $l_1$  is replaced with another objective literal  $l_2$  by equivalent literal substitution if  $l_1$  and  $l_2$  are equivalent – regardless of whether the literals or their negations appear in an objective. Specifically, every occurrence of  $l_2$  (resp.  $\neg l_2$ ) is replaced by  $l_1$  (resp.  $\neg l_1$ ) and in every objective  $O_i$  ( $i = 1, \dots, p$ )  $l_1$  (resp.  $\neg l_1$ ) gets the coefficient  $c_i(l_1) + c_i(l_2)$  (resp.  $c_i(\neg l_1) + c_i(\neg l_2)$ ). In this way, the costs of solutions to the preprocessed instance are left unchanged.

**Intrinsic At-Most-One Technique [23, 24].** Lifted to MO-MaxSAT from MaxSAT, the intrinsic at-most-one technique works as follows. Given a set  $L$  of objective literals at most one of which are falsified in each solution to  $\mathcal{I}$  (i.e., at least  $|L| - 1$  of the literals in  $L$  will incur cost in all solutions; such an  $L$  is sought heuristically using unit propagation), (i) a new literal  $l_L$  is introduced, (ii) the clause  $(l_L \vee \bigvee_{l \in L} \neg l)$  is added to  $F$ , and (iii) for every objective  $O_i$  ( $i = 1, \dots, p$ ) the coefficients of all literals in  $L$  are reduced by the minimum of these coefficients  $c_i^m = \min\{c_i(l) \mid l \in L\}$  and the terms  $c_i^m \cdot l_L + (|L| - 1) \cdot c_i^m$  are added. For intuition, the preliminary condition implies that for any solution to  $\mathcal{I}$ , at least  $|L| - 1$  of the literals in  $L$  will incur cost at least  $c_i^m$  wrt each of the objectives. The added clause (ii) enforces that  $l_L$  must be true when all literals in  $L$  are true, incurring additional cost  $c_i^m$  wrt each of the objectives.

**Binary Core Removal (BCR) [19, 31].** BCR can be phrased as first applying a restriction of the intrinsic at-most-one technique and then applying BVE. In detail, assume that a set  $L = \{l_1, l_2\}$  with  $c_i(l_1) = c_i(l_2)$  for all objectives satisfies the condition required for the intrinsic-at-most-ones and that  $\neg l_1$  and  $\neg l_2$  do not appear in  $F$ . Then BCR can be viewed as an application of intrinsic at-most-ones on  $L$  followed by applying BVE to eliminate  $l_1$  and  $l_2$  (in practice when the size of the instance does not increase). Thus, its correctness for MO-MaxSAT follows directly from the correctness of intrinsic at-most-ones and BVE.

## 6 Experiments

Complementing our theoretical observations on redundancy notions for MO-MaxSAT, we detail results from an empirical evaluation of the combined effect of the various MO-MaxSAT preprocessing techniques overviewed in the preceding section in terms of their ability to

reduce the size of real-world MO-MaxSAT instances and effect on runtime behaviour of recent MO-MaxSAT solvers. To the best of our understanding, this is the first evaluation on the effect of preprocessing on MO-MaxSAT instances and the runtime performance of MO-MaxSAT solvers.

We extended the MaxSAT preprocessor MaxPre 2 [31, 24] to MaxPre 2.1, covering MO-MaxSAT. The preprocessor implementation, empirical data, and benchmarks are available via <https://bitbucket.org/coreo-group/mo-prepro>. As the technique specification for MaxPre 2.1 in the experiments we used `[[uvsvrgc]VRTG]`, including unit propagation, BVE, SE, SSR, GSLE, BCR, TrimMaxSAT, FLE, ELS, and intrinsic at-most-ones.<sup>2</sup>

We consider four MO-MaxSAT solvers: LEXIMAXIST<sup>3</sup> [10], BIOPTSAT<sup>4</sup> [25], CLM<sup>5</sup> [11], and SCUTTLE, our own implementation of a SAT-based approach based on enumerating so-called  $P$ -minimal models [40, 32]. BIOPTSAT, CLM, and SCUTTLE compute a Pareto-optimal solution for each element in the non-dominating set. BIOPTSAT is specific for MO-MaxSAT under two objectives (i.e., bi-objective MaxSAT), while SCUTTLE and CLM handle any number of objectives. LEXIMAXIST restricts to the simpler task of computing a leximax-optimal solution, which corresponds to computing a specific element in the non-dominated set [16]. As BIOPTSAT and LEXIMAXIST offer different configurations, for BIOPTSAT we consider the three central variants of a linear SAT-UNSAT based algorithm (denoted LSU), a core-guided (MSU3) algorithm (denoted CG), and a hybrid between the two that was found to perform best in [25] (denoted Hybrid). For LEXIMAXIST, we consider both a linear SAT-UNSAT and a core-guided version of the approach. For CLM we evaluate both the core-guided (denoted CG) and the implicit hitting set algorithm (denoted IHS). To achieve a tight integration, we included MaxPre 2.1 as a library into the source code of each solver. The modified source code of each solver is also available through <https://bitbucket.org/coreo-group/mo-prepro>.

We used three real-world benchmarks from the literature: package upgradeability (PackUP) [26], learning interpretable decision rules (LIDR) [37], and development assurance level (DAL) [6]. For PackUP we used the set of 3692 instances from [10], obtained from Mancoosi International Solver Competition (<https://www.mancoosi.org/misc-2011/>) instances using all combinations of 2–5 of the 5 original objectives. The 366 LIDR benchmark instances with two objectives originate from [25], encoding the classification task for public benchmark datasets. The 96 DAL benchmark instances originate from the LION9 challenge (<https://www.cristal.univ-lille.fr/LION9/challenge.html>), each with 7 objectives. The pseudo-boolean constraints in the DAL instances were encoded with a (generalized) totalizer encoding [2, 30].

All runtime experiments were executed on 2.60-GHz Intel Xeon E5-2670 machines with 64-GB RAM in RHEL under a 1.5-hour per-instance time and 16-GB memory limit. Times reported include the runtimes of MaxPre 2.1 whenever preprocessing is used.

## 6.1 Effect of Preprocessing on Instance Characteristics

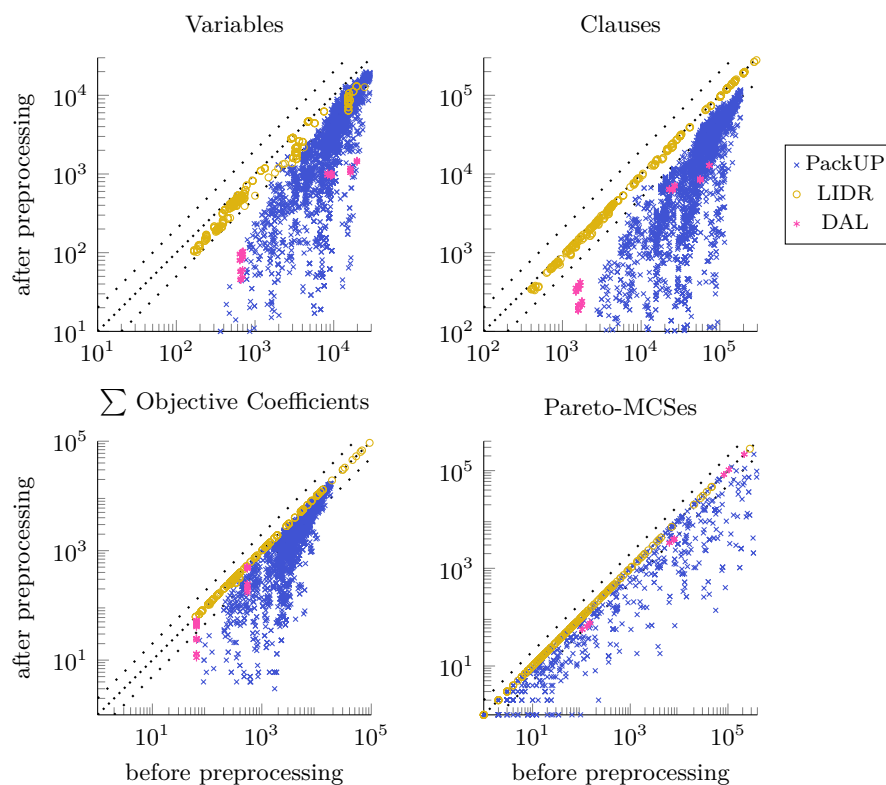
We first consider the effect of preprocessing on four central characteristics of MO-MaxSAT instances: the number of variables, the number of clauses, the sum of objective coefficients, and the number of Pareto-MCSeS.

<sup>2</sup> We excluded BCE as using it in preliminary testing led to slightly worse runtimes overall.

<sup>3</sup> LEXIMAXIST obtained from <https://github.com/miguelcabral/leximaxist>.

<sup>4</sup> BIOPTSAT obtained from <https://bitbucket.org/coreo-group/bioptsat>

<sup>5</sup> CLM obtained from <https://gitlab.inesc-id.pt/u001810/moco>



■ **Figure 1** Per-instance instance size reductions achieved by preprocessing.

Figure 1 shows the reduction of variables (top left), number of clauses (top right), the sum of objective coefficients (bottom left), and the number of Pareto-MCSes (bottom right) obtained with MaxPre 2.1 for each of the three benchmark domains. The number of variables (Figure 1 top left) is reduced significantly on each benchmark domain. In terms of medians, after preprocessing 9.5% of the original variables remain for DAL, 32% for PackUP, and 64% for LIDR instances. In terms of clauses (Figure 1 top right), the reductions are very significant for both DAL and PackUP, with 9.3% and 24% of the original number of clauses remaining after preprocessing in terms of the median, respectively. For LIDR the number of clauses is reduced less significantly, although a reduction can still be observed; 93% of the original number of clauses remain.

Given an instance  $\mathcal{I} = (F, (O_1, \dots, O_p))$ , we measure the sum of objective coefficients, i.e.,  $\sum_{i=1}^p \sum_{l \in \mathcal{B}_i(\mathcal{I})} c_i(l)$ . Note that preprocessing can change the sum of objective coefficients both by inferring that some objective literals can be set to 0 – conceptually decreasing the trivial upper bound on the objective – and by inferring that some literal must be assigned to 1 – conceptually increasing the lower bound. Figure 1 (bottom left) shows the reduction in the sum of objective coefficients achieved by preprocessing on each benchmark instance. The magnitude of reductions achieved by preprocessing depend significantly on the benchmark domain. For LIDR, preprocessing only seldom reduces objective coefficients. For PackUP a significant reduction is observed; the median sum of objective coefficients after preprocessing is 57% of the original. Furthermore, on 297 of the PackUP instances preprocessing reduced at least one of the objectives to *zero*, removing it from the instance. For DAL, while for some instances the objective coefficients are reduced only slightly, on every single instance preprocessing reduced at least one of the objectives to zero. The median sum of objective coefficients after preprocessing is 54% of the original for DAL.

■ **Table 1** Solved instances (#), uniquely solved instances (uniq.), and cumulative runtimes over solved ( $\sum t$ ) in  $10^3$  seconds, with and without preprocessing (Prepro.).

Solver	Prepro.	PackUP			LIDR			DAL		
		#	uniq.	$\sum t$	#	uniq.	$\sum t$	#	uniq.	$\sum t$
BIOPTSAT (LSU) (bi opt.)	no	1134	0	61.4	220	1	52.4	–	–	–
	yes	<b>1161</b>	27	<b>47.7</b>	<b>223</b>	4	<b>34.1</b>	–	–	–
BIOPTSAT (CG) (bi opt.)	no	1154	1	40.9	222	1	43.9	–	–	–
	yes	<b>1159</b>	6	<b>34.5</b>	222	1	<b>38.4</b>	–	–	–
BIOPTSAT (Hybrid) (bi opt.)	no	1154	1	46.6	222	0	40.4	–	–	–
	yes	<b>1159</b>	6	<b>33.0</b>	222	0	<b>35.5</b>	–	–	–
SCUTTLE (multi opt.)	no	1772	40	284.5	<b>219</b>	1	51.3	66	0	5.9
	yes	<b>1778</b>	46	<b>244.1</b>	218	0	44.1	<b>67</b>	1	<b>5.3</b>
CLM (CG) (multi opt.)	no	<b>1593</b>	88	<b>301.3</b>	206	2	<b>48.0</b>	<b>60</b>	7	<b>8.1</b>
	yes	1588	83	315.8	206	2	49.4	53	0	12.8
CLM (IHS) (multi opt.)	no	<b>1301</b>	91	258.5	<b>134</b>	19	26.8	<b>48</b>	7	0.3
	yes	1282	72	189.8	115	0	23.5	41	0	0.2
LEXIMAXIST (LSU) (leximax opt.)	no	2276	2	434.6	224	0	<b>28.4</b>	72	0	<b>4.3</b>
	yes	<b>2347</b>	73	<b>268.5</b>	224	0	29.6	72	0	5.2
LEXIMAXIST (CG) (leximax opt.)	no	2450	13	140.7	<b>220</b>	2	43.7	72	1	12.9
	yes	<b>2453</b>	16	<b>140.0</b>	218	0	38.0	<b>73</b>	2	<b>9.7</b>

For investigating how preprocessing affects the number of Pareto-MCSEs, we used SCUTTLE to enumerate Pareto-MCSEs of each benchmark instance under a 1.5-h per-instance time limit. On the LIDR instances, no reduction in the number of Pareto-MCSEs was observed. For PackUP and DAL, respectively, preprocessing reduced the number of Pareto-MCSEs significantly, by more than one third for 33% and 60% of the instances, respectively. Furthermore, considering the per-instance reduction shown in Figure 1 (bottom right), we observed that for PackUP the number of Pareto-MCSEs is often reduced significantly further.

## 6.2 Effect of Preprocessing on Solver Runtimes

We now turn to investigating the effect of preprocessing on the runtime performance of MO-MaxSAT solvers.

Table 1 shows the number of solved instances, number of instances uniquely solved with or without preprocessing, and cumulative runtimes over solved instances (in  $10^3$  seconds) for each solver. We emphasize that here one should focus on comparing the effect of preprocessing on each individual solver and configuration. Most importantly, the numbers reported for the four different solvers – BIOPTSAT, SCUTTLE, CLM, and LEXIMAXIST – are not directly comparable to each other as they solve different variants of MO-MaxSAT: LEXIMAXIST computes a solution corresponding to a single element in the non-dominated set, while BIOPTSAT, SCUTTLE, and CLM enumerate the whole non-dominated set. Furthermore, since BIOPTSAT supports two objectives only, data for BIOPTSAT on PackUP is restricted to the 1420 instances with two objectives, and data on DAL is unavailable as the DAL instances involve more than two objectives.

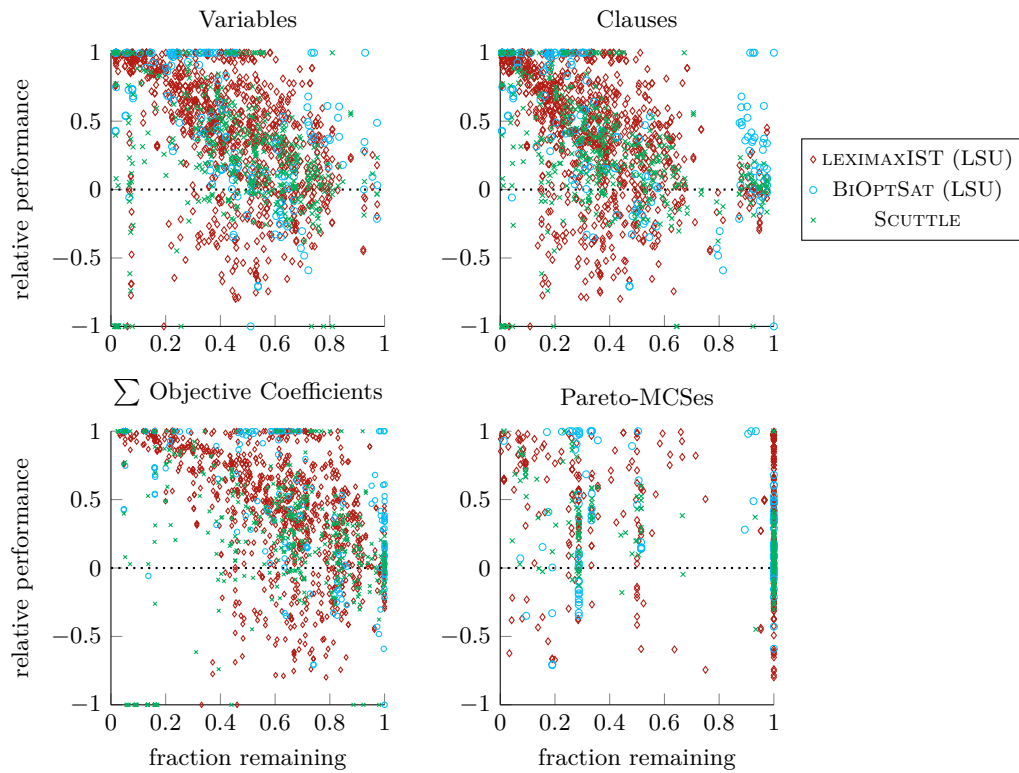
For PackUP, preprocessing has a clear positive impact on both the number of instances solved and the runtimes of all solvers except for CLM: the solvers use less cumulative runtime after preprocessing for solving more instances than what can be solved without preprocessing. We observe that for each of the three variants of BIOPTSAT as well as the LSU variant of LEXIMAXIST, preprocessing strictly increases the number of PackUP instances solved. There are close to no uniquely solved instances when preprocessing is not employed. Interestingly, the runtime improvement obtained using preprocessing for the LSU variants of BIOPTSAT, which without preprocessing is outperformed by the other BIOPTSAT variants, is so significant on PackUP instances that the LSU variant ends up even slightly outperforming the other variants. For SCUTTLE and the CG variant of LEXIMAXIST the number of uniquely solved instances with preprocessing is also higher than without, although there is a more significant number of instances that are uniquely solved without preprocessing. For LIDR, preprocessing speeds up all three configurations of BIOPTSAT and also increases the number of instances solved for the LSU variant. However, preprocessing does not consistently improve the performance of SCUTTLE, CLM, and LEXIMAXIST on LIDR and DAL.

Overall, although somewhat modestly, preprocessing appears to have the most significant positive impact on linear SAT-UNSAT type algorithms, namely, the LSU variants of BIOPTSAT and LEXIMAXIST. This finding is in fact in-line with [24] where, in the context of MaxSAT, the strongest positive impact of preprocessing was observed for a linear SAT-UNSAT (solution-improving) MaxSAT solver.

Finally, we investigate potential correlations between the impact of preprocessing on solver runtimes and the instance characteristics of number of clauses, number of variables, sum of objective coefficients, and number of Pareto-MCSeS. As a metric for the impact of preprocessing on solver runtimes, we use *relative solver performance*, defined for a fixed instance and solver as  $(t_{\text{no\_prepro}} - t_{\text{prepro}}) / (t_{\text{no\_prepro}} + t_{\text{prepro}})$ , where  $t_{\text{(no) prepro}}$  is the solving time with (without) preprocessing. This metric takes values from  $-1$  to  $1$ . A positive value implies that runtime with preprocessing was shorter than without preprocessing, and the value  $1$  (value  $-1$ ) means that the solver was able to solve the instance with (without) preprocessing, but timed out without (with) it; the closer to  $1$  ( $-1$ ), the more significant a positive (negative) effect preprocessing has on overall runtime. As a metric for the impact of preprocessing on instance characteristics, we use *fraction remaining*. For a specific instance and instance characteristic, let  $f_{\text{(no) prepro}}$  be the value of the feature with (without) preprocessing. The fraction remaining is then  $f_{\text{prepro}} / f_{\text{no\_prepro}}$ , taking values from  $0$  to  $1$ . For some intuition, the closer to  $0$  the value is, the more significantly preprocessing affects the instance characteristic: e.g., the value  $0$  for the number of clauses means that preprocessing removes all clauses from an instance, and a value of  $0.5$  ( $1$ ) means that the preprocessed instance contains half as many (exactly as many) clauses as the original instance.

Figure 2 relates relative solver performance and the fractions remaining for the four instance characteristics for each solver using the configuration the runtimes of which were improved the most by preprocessing: BIOPTSAT (LSU), SCUTTLE, and LEXIMAXIST (LSU), focusing on “non-trivial” instances with runtimes  $> 60$  seconds either with or without preprocessing. We observe that a lower fraction of variables remaining (Figure 2 top left), clauses (top right), or objective coefficient sum (bottom left) by preprocessing often also somewhat tends to result in faster solver runtimes (i.e., a higher relative performance of the solver), especially for LEXIMAXIST. Interestingly, the data for the LSU variant of BIOPTSAT as well as for SCUTTLE are quite scattered, with no clear correlations observed between relative solver performance and changes in instance characteristics. Finally, we note that the number of Pareto-MCSeS appears to have little to no impact on the relative performance





■ **Figure 2** Relating solver runtimes with instance characteristic.

of these specific solvers. One possible explanation for this observation is that none of these specific solvers explicitly enumerate Pareto-MCSes in their search. On the other hand, based on the data, reducing the sum of the objective coefficients by preprocessing may be beneficial for solver performance; also in light of this developing further techniques that are capable of reducing the objective ranges appears to be an interesting direction for further work.

## 7 Conclusions

Motivated by recent advances in (Max)SAT-based approaches to multi-objective optimization, we proposed redundancy notions and liftings of MaxSAT preprocessing techniques for the multi-objective setting. We showed that the redundancy notions capture different preprocessing techniques, with the (in)ability to remove Pareto-MCSes as the underlying differentiating property. We provided a stand-alone preprocessor implementation of the preprocessing techniques, and empirically evaluated the impact of preprocessing in multi-objective MaxSAT. The preprocessor can significantly reduce the size of real-world multi-objective MaxSAT instances and also has in cases a positive effect on runtimes of current state-of-the-art multi-objective MaxSAT solvers. Interesting directions for future work include developing redundancy notions that can capture changes to objectives; more fine-grained analysis of preprocessing for the restricted case of leximax optimization; and empirical evaluation of preprocessing on further problem settings with varying instance properties such as different distributions of objective coefficients.

---

**References**

---

- 1 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability—Second Edition*, volume 336 of *FAIA*, chapter 24, pages 929–991. IOS Press, 2021. doi:10.3233/FAIA201008.
- 2 Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming—9th International Conference, CP*, volume 2833 of *LNCS*, pages 108–122. Springer, 2003. doi:10.1007/978-3-540-45193-8\_8.
- 3 Anton Belov, António Morgado, and João Marques-Silva. SAT-based preprocessing for MaxSAT. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning—19th International Conference, LPAR*, volume 8312 of *LNCS*, pages 96–111. Springer, 2013. doi:10.1007/978-3-642-45221-5\_7.
- 4 Jeremias Berg and Matti Järvisalo. Unifying reasoning and core-guided search for maximum satisfiability. In Francesco Calimeri, Nicola Leone, and Marco Manna, editors, *Logics in Artificial Intelligence—16th European Conference, JELIA*, volume 11468 of *LNCS*, pages 287–303. Springer, 2019. doi:10.1007/978-3-030-19570-0\_19.
- 5 Jeremias Berg, Paul Saikko, and Matti Järvisalo. Subsumed label elimination for maximum satisfiability. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *22nd European Conference on Artificial Intelligence, ECAI 2016*, volume 285 of *FAIA*, pages 630–638. IOS Press, 2016. doi:10.3233/978-1-61499-672-9-630.
- 6 Pierre Bieber, Remi Delmas, and Christel Seguin. DALculus—Theory and tool for development assurance level allocation. In Francesco Flammini, Sandro Bologna, and Valeria Vittorini, editors, *Computer Safety, Reliability, and Security—30th International Conference, SAFECOMP*, volume 6894 of *LNCS*, pages 43–56. Springer, 2011. doi:10.1007/978-3-642-24270-0\_4.
- 7 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability—Second Edition*, volume 336 of *FAIA*. IOS Press, 2021. doi:10.3233/FAIA336.
- 8 Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability—Second Edition*, volume 336 of *FAIA*, chapter 9, pages 391–435. IOS Press, 2021. doi:10.3233/FAIA200992.
- 9 Ronen I. Brafman. A simplifier for propositional formulas with many binary clauses. *IEEE Trans. Syst. Man Cybern. Part B*, 34(1):52–59, 2004. doi:10.1109/TSMCB.2002.805807.
- 10 Miguel Cabral, Mikolás Janota, and Vasco M. Manquinho. SAT-based leximax optimisation algorithms. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 236 of *LIPICs*, pages 29:1–29:19. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SAT.2022.29.
- 11 João Cortes, Inês Lynce, and Vasco M. Manquinho. New core-guided and hitting set algorithms for multi-objective combinatorial optimization. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 29th International Conference, TACAS*, volume 13994 of *LNCS*, pages 55–73. Springer, 2023. doi:10.1007/978-3-031-30820-8\_7.
- 12 Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005. doi:10.1007/11499107\_5.
- 13 Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. In Ofer Strichman and Armin Biere, editors, *First International Workshop on Bounded Model Checking, BMC@CAV*, volume 89 of *Electronic Notes in Theoretical Computer Science*, pages 543–560. Elsevier, 2003. doi:10.1016/S1571-0661(05)82542-3.

- 14 Matthias Ehrgott. Efficiency and nondominance. In *Multicriteria Optimization (2. ed.)*, chapter 2, pages 23–64. Springer, 2005.
- 15 Matthias Ehrgott. Multiobjective combinatorial optimization. In *Multicriteria Optimization (2. ed.)*, chapter 8, pages 197–220. Springer, 2005.
- 16 Matthias Ehrgott. Other definitions of optimality—nonscalarizing methods. In *Multicriteria Optimization (2. ed.)*, chapter 5, pages 127–150. Springer, 2005.
- 17 Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectr.*, 22(4):425–460, 2000. doi:10.1007/s002910000046.
- 18 Jon William Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, USA, 1995. UMI Order No. GAX95-32175.
- 19 James F. Gimpel. A reduction technique for prime implicant tables. In *5th Annual Symposium on Switching Circuit Theory and Logical Design, SWCT*, pages 183–191. IEEE Computer Society, 1964. doi:10.1109/SWCT.1964.4.
- 20 Andreia P. Guerreiro, João Cortes, Daniel Vanderpooten, Cristina Bazgan, Inês Lynce, Vasco M. Manquinho, and José Rui Figueira. Exact and approximate determination of the pareto front using minimal correction subsets. *Comput. Oper. Res.*, 153:106153, 2023. doi:10.1016/j.cor.2023.106153.
- 21 Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere. Clause elimination for SAT and QSAT. *J. Artif. Intell. Res.*, 53:127–168, 2015. doi:10.1613/jair.4694.
- 22 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Strong extension-free proof systems. *J. Autom. Reason.*, 64(3):533–554, 2020. doi:10.1007/s10817-019-09516-0.
- 23 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. doi:10.3233/SAT190116.
- 24 Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. Clause redundancy and preprocessing in maximum satisfiability. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning—11th International Joint Conference, IJCAR*, volume 13385 of *LNCS*, pages 75–94. Springer, 2022. doi:10.1007/978-3-031-10769-6\_6.
- 25 Christoph Jabs, Jeremias Berg, Andreas Niskanen, and Matti Järvisalo. MaxSAT-based bi-objective boolean optimization. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 236 of *LIPICs*, pages 12:1–12:23. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SAT.2022.12.
- 26 Mikolás Janota, Inês Lynce, Vasco M. Manquinho, and João Marques-Silva. PackUp: Tools for package upgradability solving. *J. Satisf. Boolean Model. Comput.*, 8(1/2):89–94, 2012. doi:10.3233/sat190090.
- 27 Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010*, volume 6015 of *LNCS*, pages 129–144. Springer, 2010. doi:10.1007/978-3-642-12002-2\_10.
- 28 Matti Järvisalo, Armin Biere, and Marijn Heule. Simulating circuit-level simplifications on CNF. *J. Autom. Reason.*, 49(4):583–619, 2012. doi:10.1007/s10817-011-9239-9.
- 29 Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning—6th International Joint Conference, IJCAR*, volume 7364 of *LNCS*, pages 355–370. Springer, 2012. doi:10.1007/978-3-642-31365-3\_28.
- 30 Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-boolean constraints. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming—21st International Conference, CP*, volume 9255 of *LNCS*, pages 200–209. Springer, 2015. doi:10.1007/978-3-319-23219-5\_15.

- 31 Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An extended MaxSAT preprocessor. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing—20th International Conference, SAT*, volume 10491 of *LNCS*, pages 449–456. Springer, 2017. doi:10.1007/978-3-319-66263-3\_28.
- 32 Miyuki Koshimura, Hidetomo Nabeshima, Hiroshi Fujita, and Ryuzo Hasegawa. Minimal model generation with respect to an atom set. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *7th International Workshop on First-Order Theorem Proving, FTP*, volume 556 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. URL: <http://ceur-ws.org/Vol-556/paper06.pdf>.
- 33 Oliver Kullmann. On a generalization of extended resolution. *Discret. Appl. Math.*, 96-97:149–176, 1999. doi:10.1016/S0166-218X(99)00037-2.
- 34 Daniel Le Berre. Exploiting the real power of unit propagation lookahead. *Electron. Notes Discret. Math.*, 9:59–80, 2001. doi:10.1016/S1571-0653(04)00314-2.
- 35 Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. In Henry A. Kautz and Bruce W. Porter, editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 291–296. AAAI Press / The MIT Press, 2000. URL: <http://www.aaai.org/Library/AAAI/2000/aaai00-045.php>.
- 36 Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reason.*, 40(1):1–33, 2008.
- 37 Dmitry Malioutov and Kuldeep S. Meel. MLIC: A maxsat-based framework for learning interpretable classification rules. In John N. Hooker, editor, *Principles and Practice of Constraint Programming—24th International Conference, CP*, volume 11008 of *LNCS*, pages 312–327. Springer, 2018. doi:10.1007/978-3-319-98334-9\_21.
- 38 Richard Ostrowski, Éric Grégoire, Bertrand Mazure, and Lakhdar Sais. Recovering and exploiting structural knowledge from CNF formulas. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming—8th International Conference, CP*, volume 2470 of *LNCS*, pages 185–199. Springer, 2002. doi:10.1007/3-540-46135-3\_13.
- 39 Tobias Paxian, Pascal Raiola, and Bernd Becker. On preprocessing for weighted MaxSAT. In *Verification, Model Checking, and Abstract Interpretation, VMCAI*, volume 12597 of *LNCS*, pages 556–577. Springer, 2021.
- 40 Takehide Soh, Mutsunori Banbara, Naoyuki Tamura, and Daniel Le Berre. Solving multiobjective discrete optimization problems with propositional minimal model generation. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming—23rd International Conference, CP*, volume 10416 of *LNCS*, pages 596–614. Springer, 2017. doi:10.1007/978-3-319-66158-2\_38.
- 41 Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT*, volume 3542 of *LNCS*, pages 276–291. Springer, 2004. doi:10.1007/11527695\_22.
- 42 Miguel Terra-Neves, Inês Lynce, and Vasco M. Manquinho. Introducing pareto minimal correction subsets. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing—20th International Conference, SAT*, volume 10491 of *LNCS*, pages 195–211. Springer, 2017. doi:10.1007/978-3-319-66263-3\_13.
- 43 Miguel Terra-Neves, Inês Lynce, and Vasco M. Manquinho. Multi-objective optimization through pareto minimal correction subsets. In Jérôme Lang, editor, *27th Joint Conference on Artificial Intelligence, IJCAI*, pages 5379–5383. ijcai.org, 2018. doi:10.24963/ijcai.2018/757.
- 44 Ekunda L. Ulungu and Jacques Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-criteria Decision Analysis*, 3:83–104, 1994.
- 45 Allen Van Gelder. Toward leaner binary-clause reasoning in a satisfiability solver. *Ann. Math. Artif. Intell.*, 43(1):239–253, 2005. doi:10.1007/s10472-005-0433-5.



## 18:20 Preprocessing in SAT-Based Multi-Objective Combinatorial Optimization

- 46 Ramin Zabih and David A. McAllester. A rearrangement search strategy for determining propositional satisfiability. In Howard E. Shrobe, Tom M. Mitchell, and Reid G. Smith, editors, *Proceedings of the 7th National Conference on Artificial Intelligence, AAAI*, pages 155–160. AAAI Press / The MIT Press, 1988. URL: <http://www.aaai.org/Library/AAAI/1988/aaai88-028.php>.

# An Efficient Constraint Programming Approach to Preemptive Job Shop Scheduling

Carla Juvin 

LAAS-CNRS, Université de Toulouse, France

Emmanuel Hebrard  

LAAS-CNRS, Université de Toulouse, France

Laurent Houssin  

ISAE-SUPAERO, Université de Toulouse, France

Pierre Lopez  

LAAS-CNRS, Université de Toulouse, France

---

## Abstract

Constraint Programming has been widely, and very successfully, applied to scheduling problems. However, the focus has been on uninterruptible tasks, and preemptive scheduling problems are typically harder for existing constraint solvers. Indeed, one usually needs to represent all potential task interruptions thus introducing many variables and symmetrical or dominated choices.

In this paper, building on mostly known results, we observe that a large class of preemptive disjunctive scheduling problems do not require an explicit model of task interruptions. We then introduce a new constraint programming approach for this class of problems that significantly outperforms state-of-the-art dedicated approaches in our experimental results.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Computing methodologies → Planning and scheduling

**Keywords and phrases** Constraint Programming, Scheduling, Preemptive Resources

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.19

**Supplementary Material** *Software (Source Code)*: <https://github.com/ehebrard/Mistral-2.0.git>, archived at [swh:1:dir:90fe19df244134e5dfddd79360cd4b554fecabfb](https://swh.1:1:dir:90fe19df244134e5dfddd79360cd4b554fecabfb)

**Funding** *Emmanuel Hebrard*: ANR Project AD-Lib (ANR-22-CE23-0014).

**Acknowledgements** We would like to thank Claude-Guy Quimper for the advice and discussions while writing this paper.

## 1 Introduction

Many applications involve scheduling a set of tasks subject to resource constraints. Constraint programming (CP) techniques lead to significant advances in this domain and, conversely, some of the early work on the propagation of global constraints originated from scheduling applications.

In *preemptive* scheduling problems, the processing of some tasks can be interrupted, to be resumed at a later time. CP is generally much more successful on non-preemptive rather than preemptive scheduling problems. The standard approach to modelling interruptible tasks in constraint programming, while preserving completeness<sup>1</sup>, is to decompose each task into as many unit-length tasks as its duration. A disjunctive resource can then be modelled as an ALLDIFFERENT constraint, a parallel-machine resource as a GLOBALCARDINALITY

---

<sup>1</sup> Without this restriction, the range of possible approaches is significantly wider.



constraint, and a cumulative resource as a general flow constraint. Another approach is to decompose the tasks into variable-size tasks. In this case, either the duration variables have a null lower bound, which severely hinders constraint propagation, or constraints have to be posted on the number of fragments and their duration, which entails searching over the different ways to split the tasks. As a result, the former approach is often the best. In either cases, the complexity of the methods heavily depends on the duration of the tasks, and therefore do not scale well in practice.

In this paper, we use the observation that when preemptive resources are disjoint, there is no need to compute the fragmentation of the tasks during search. In other words, the problem can be modelled as a constraint satisfaction problem (CSP) with two variables per task: one for its start time and one for its end time. Deciding whether a set of interruptible tasks with release and due dates can be processed on a disjunctive constraint is easy (e.g., via the *Jackson Preemptive Schedule*). Moreover, if some tentative release and due dates for the tasks pass the *overload check*<sup>2</sup> [29], then a feasible fragmentation of the tasks is guaranteed to exist. Therefore, provided that the disjunctive resources are *disjoint* and that constraint propagation is at least as strong as the overload check, one can simply branch on start and end time variables for every task.

This condition of disjointness is true in many problems, and there is no further restriction on the constraint graph. For instance in this paper we focus on the preemptive Job Shop Scheduling Problem (pJSSP). In this problem, the resources are naturally disjoint, but tasks requiring distinct machines can be linked by precedence constraints. Our method can be applied to several other preemptive versions of disjunctive scheduling problems (e.g., open shop scheduling where job sequences are to be decided, job shop scheduling augmented with setup times, or generalised precedences, etc.).

The paper is organised as follows: In Section 2 we recall some background and define the disjunctive preemptive constraint as well as the preemptive Job shop Scheduling problem. In Section 3 we briefly review the existing CP models for preemptive scheduling. We show that the standard approach of using the constraints ALLDIFFERENT or NOOVERLOAD on unit-length tasks are weaker than the same approach using the ALLDIFFPREC constraint. Then we discuss the main observations, which are not original, but are key to our main result: it is often not necessary to introduce unit-length tasks, and without unit-length tasks Edge-Finding is sufficient to enforce bounds consistency. Then we introduce a novel constraint model for the preemptive Job Shop Scheduling Problem based on these observations in Section 4 and finally we discuss the state of the art for the pJSSP and experimentally compare our approach to it in Section 5.

## 2 Background and Related Work

### 2.1 CSP and bounds consistency

A *constraint network* is a triple  $(\mathbf{x}, \mathcal{D}, \mathbf{c})$  where  $\mathbf{x}$  is a finite totally ordered set of *variables*,  $\mathcal{D}$  is a finite set, and  $\mathbf{c}$  is a finite set of *constraints*. A constraint  $c$  is a pair  $(S_c, P_c)$  where the *scope*  $S_c$  is a subset of  $\mathbf{x}$  and  $P_c$  is a predicate on the variables  $S_c$ . An assignment  $\sigma : \mathbf{x} \mapsto \mathcal{D}$  is a mapping from variables in  $\mathbf{x}$  to values in  $\mathcal{D}$ , and  $\sigma(x)$  denotes the value assigned to variable  $x$  by  $\sigma$ . Given a constraint network  $(\mathbf{x}, \mathcal{D}, \mathbf{c})$ , the *Constraint Satisfaction Problem (CSP)* asks whether there is an assignment  $\sigma$  of values in  $\mathcal{D}$  to the variables  $\mathbf{x}$  such that for every constraint  $c \in \mathbf{c}$ , the predicate  $P_c$  is true on the projection of  $\sigma$  onto  $S_c$ .

<sup>2</sup> That the total duration of any set of tasks is not larger than their execution window.

Most constraint solvers store a current domain  $D(x) \subseteq \mathcal{D}$  for every variable  $x \in \mathbf{x}$  and use some form of consistency reasoning to reduce these domains without losing solutions. We assume that the domain  $\mathcal{D}$  is totally ordered and denote  $\min(x)$  (resp.  $\max(x)$ ) the minimum (resp. maximum) value in  $D(x)$ . Moreover, we denote  $[l, u]$  the discrete interval containing every element in  $\mathcal{D}$  that is larger than or equal to  $l$  and lower than or equal to  $u$ .

► **Definition 1** (Bounds Consistency). *Let  $c$  be a constraint and  $x \in S_c$  be a variable constrained by  $c$ . An assignment  $\sigma$  is a bound support of the pair  $(x, v)$  in a current domain  $D$  if and only if:*

- $\sigma(x) = v$ ;
- the predicate  $P_c$  is satisfied by  $\sigma$  ( $\sigma$  is said to be consistent and we write  $P_c(\sigma) = \mathbf{true}$ );
- and for every variable  $y \in \mathbf{x} \setminus x$ ,  $\sigma(y) \in [\min(y), \max(y)]$  ( $\sigma$  is said to be valid).

*A constraint  $c$  is bounds consistent (BC) for the current domain  $D$  if and only if, for every  $x \in S_c$ ,  $(x, \min(x))$  and  $(x, \max(x))$  have a bound support in  $D$ .*

*A constraint network  $\mathcal{N} = (\mathbf{x}, \mathcal{D}, \mathbf{c})$  is BC in a current domain  $D$  if and only if, for every  $c \in \mathbf{c}$ ,  $c$  is BC in  $D$ .*

The notion of consistency can be used to compare constraint models, and in particular compare constraints to their decompositions.

► **Definition 2** (Pruning power). *Let  $c$  be a constraint and  $\mathcal{N} = (\mathbf{x}, \mathcal{D}, \mathbf{c})$  be a decomposition of the constraint, i.e., a constraint network such that:*

- $S_c \subseteq \mathbf{x}$  and
- $\sigma$  satisfies  $P_c$  if and only if, there is a solution  $\sigma'$  of  $\mathcal{N}$  such that the projection of  $\sigma'$  onto  $S_c$  is equal to  $\sigma$ .

*We say that BC on the decomposition  $\mathcal{N}$  is as strong as BC on the constraint  $c$  if for any current domain  $D$ , it holds that  $\mathcal{N}$  is BC in  $D$  implies  $c$  is BC in  $D$ . Otherwise, we say that BC on the decomposition is weaker than on the constraint.*

## 2.2 Preemptive Scheduling

The constraint `PREEMPTIVE_NOOVERLAP` ensures that a set of interruptible tasks requiring a disjunctive resource do not overlap. Let  $\mathcal{T} = \{t_1, \dots, t_n\}$  be a set of tasks, and let  $\mathbf{s} = \{s_1, \dots, s_n\}$  and  $\mathbf{e} = \{e_1, \dots, e_n\}$  be two sets of variables standing respectively for the earliest start and latest end times of the tasks, i.e., a task  $t_i \in \mathcal{T}$  must be processed in the interval  $[s_i, e_i)$  (closed at the start and opened at the end). Notice that although we will often use the terms “start (or end) variable” for convenience, these variables actually define an interval in which the task is processed. In particular, task  $t_j$  might not be processed at all at time  $s_j$  and may be finished at a time strictly earlier than  $e_j$ . This is apparent in Definitions 3 and 4, and the reasons for this choice are discussed in Section 3.2.1.

Moreover, let  $p_i$  be the duration of task  $t_i$ . For a set of tasks  $\Omega \subseteq \mathcal{T}$ , we write  $s_\Omega$  for the earliest start time of any task in  $\Omega$  ( $s_\Omega = \min(\{s_i \mid t_i \in \Omega\})$ ),  $e_\Omega$  for the latest end time of any task in  $\Omega$  ( $e_\Omega = \max(\{e_i \mid t_i \in \Omega\})$ ), and  $p_\Omega = \sum_{t_i \in \Omega} p_i$  for the total processing time of tasks in  $\Omega$ . Since the tasks are interruptible, there must exist a “fragmentation” function that maps at most one task to each time point. Let  $\mathcal{H} = [0, ub)$  be a time interval where  $ub$  is some upper bound on the end times of tasks in  $\mathcal{T}$ , and let  $\mathbf{a} = \{a_{i,\tau} \mid t_i \in \mathcal{T}, \tau \in \mathcal{H}\}$  be Boolean *activation variables*, where  $a_{i,\tau}$  indicates whether task  $t_i$  is active at time  $\tau$ .



► **Definition 3.** *PREEMPTIVENOOVERLAP (on activation variables)*

$$\begin{aligned} & \text{PREEMPTIVENOOVERLAP}(\mathcal{T}, \mathbf{a}, \mathbf{s}, \mathbf{e}) \\ & \iff \\ & \forall \tau \in \mathcal{H} \sum_{t_i \in \mathcal{T}} a_{i,\tau} \leq 1 \wedge \forall t_i \in \mathcal{T} \sum_{\tau=s_i}^{e_i-1} a_{i,\tau} = p_i \end{aligned}$$

In this paper we consider the case where we do not make the fragmentation function explicit (via activation variables), but rather only care about the start and end times of the tasks, while only making sure that some fragmentation function exists. This leads to Definition 4:

► **Definition 4.** *PREEMPTIVENOOVERLAP (on start and end variables)*

$$\begin{aligned} & \text{PREEMPTIVENOOVERLAP}(\mathcal{T}, \mathbf{s}, \mathbf{e}) \\ & \iff \\ & \exists f : \mathcal{H} \mapsto \mathcal{T} \cup \{\emptyset\}, \forall t_i \in \mathcal{T}, |\{\tau \in [s_i, e_i) \mid f(\tau) = t_i\}| = p_i \end{aligned}$$

### 2.3 Preemptive Job Shop Scheduling (pJSSP)

In the pJSSP, a set  $\mathcal{J}$  of  $n$  jobs are to be processed on a set  $\mathcal{M}$  of machines. Each job  $J_i \in \mathcal{J}$  consists of a sequence of  $n_i$  tasks that must be executed in order. Each task  $t_{i,j} \in J_i$  must be executed on one machine  $M_{i,j} \in \mathcal{M}$  with a processing time  $p_{i,j} \in \mathbb{N}$ . Preemption is allowed, i.e. tasks can use a machine for some time, stop to let another task be processed, and then resume at a later time. The objective is to minimise the total makespan, that is, the maximum completion time of any task (denoted  $C_{\max}$ ).

The pJSSP is NP-hard even with two machines and three jobs ( $J2|n=3, pmtn|C_{\max}$ ) [7] while the non-preemptive version ( $J2|n=3|C_{\max}$ ) is solvable in polynomial time; an  $O(r^4)$ -algorithm with  $r = \max_{i \in \mathcal{J}} n_i$  in given in [22]. We have observed that in most academic data sets used to benchmark job shop scheduling algorithms, there is “no recirculation”, that is, jobs have exactly one task per machine. However, this particular case is also NP-hard for as few as three machines since it generalises the flow shop problem which is itself NP-hard [16]. The approach introduced in this paper applies to job shop problems with or without recirculation, although all the instances used in the experiments belong to the latter class.

## 3 Constraint Programming for Preemptive Scheduling

Many propagation algorithms for reasoning on resources for non-interruptible tasks rely on relaxing non-preemption, and can therefore be applied to the preemptive case with very few changes, as observed for instance in [24].

The *Edge-Finding* rule is of particular interest in the preemptive case. It relies on the *overload check* implied by the disjunctive resource constraint:

► **Definition 5.** *NOOVERLOAD*

$$\text{NOOVERLOAD}(\mathcal{T}) \iff \bigwedge_{\Omega \subseteq \mathcal{T}} p_{\Omega} \leq e_{\Omega} - s_{\Omega}$$

In the preemptive case, the overload check is not only implied, it is equivalent to the PREEMPTIVENOOVERLAP constraint. The following theorem is a direct corollary of Proposition 3 in [8]:

► **Theorem 6.**  $PREEMPTIVENOOVERLAP(\mathcal{T}, \mathbf{s}, \mathbf{e}) \iff NOOVERLOAD(\mathcal{T}, \mathbf{s}, \mathbf{e})$

The Edge-Finding rule consists in detecting precedence constraints whose violation would in turn make the overload check false. In the non-preemptive case, they are defined as shown in Definition 7. They are written here as constraints, but notice that they can be directly translated to a propagation rule by considering “optimistic” values for variables in the left-hand side (minima for start times and maxima for end times). Of course both constraints have a symmetric version (by reflection).

► **Definition 7.** *Edge-Finding (non-preemptive case)*

$$s_{\Omega \cup \{t_i\}} + p_i + p_\Omega > e_\Omega \implies s_i \geq s_\Omega + p_\Omega \quad \forall \Omega \subseteq \mathcal{T}, \forall t_i \in \mathcal{T} \setminus \Omega$$

A slight adaptation is required for the preemptive case [4]. The precondition implies that task  $t_i$  must end last among  $\Omega \cup \{t_i\}$ , which is not equivalent to starting last since the task can be interrupted:

► **Definition 8.** *Edge-Finding (Preemptive case)*

$$s_{\Omega \cup \{t_i\}} + p_i + p_\Omega > e_\Omega \implies e_i \geq s_{\Omega \cup \{t_i\}} + p_i + p_\Omega \quad \forall \Omega \subseteq \mathcal{T}, \forall t_i \in \mathcal{T} \setminus \Omega$$

The following theorem is a direct corollary of Proposition 9 in [4] concerning *fully elastic schedules*. A fully elastic schedule is one where tasks are preemptive and do not have a constant demand on the resource when active (however, their total *energy*, i.e., total resource demand integrated over time, is a constant). On disjunctive resource, since the demand is an integer and can only be equal to 0 (inactive) or 1 (active), fully elastic schedules and preemptive schedules are equivalent. Interestingly, this theorem shows that other rules that are useful on non-preemptive scheduling (such as the “not-first/not-last” rule) are useless in the preemptive case.

► **Theorem 9.** *Edge-Finding constraints are all bounds consistent on a set of tasks  $\mathcal{T}$  if and only if  $PREEMPTIVENOOVERLAP(\mathcal{T})$  is bounds consistent.*

### 3.1 Formulation with fragmentation

It is easy to see Definition 4 is not sufficient when the same preemptive task requires more than a single (disjunctive) resource. Consider the instance illustrated in Figure 1a. Task  $t_1$  requires resources  $a$  and  $b$  while tasks  $t_2$  and  $t_3$  require resource  $a$ , and  $t_4$  and  $t_5$  require resource  $b$ . All start and end times are fixed, and under Definition 4, the constraint  $PREEMPTIVENOOVERLAP$  is satisfied both for the scope  $(s_1, s_2, s_3, e_1, e_2, e_3)$  and  $(s_1, s_4, s_5, e_1, e_4, e_5)$ . However, there is no assignment of the variables  $\mathbf{a} = a_{1,0}, \dots, a_{1,5}$  which satisfies both constraints under Definition 3. In this section we review the existing formulations of the disjunctive constraint that model task fragmentation.

A standard way to model preemptive resources is to decompose each task  $t_i$  into  $p_i$  unit-length tasks, where variable  $x_{i,k}$  stands for the processing time of the  $k$ -th unit of task  $t_i$ . Then a disjunctive resource can be represented as an  $ALLDIFFERENT$  constraint:

► **Definition 10.**  *$ALLDIFFERENT$  decomposition of  $PREEMPTIVENOOVERLAP$  (w.r.t. Definition 3)*

$$PREEMPTIVENOOVERLAP(\mathcal{T}, \mathbf{a}, \mathbf{s}, \mathbf{e}) \iff ALLDIFFERENT(\{x_{i,k} \mid t_i \in \mathcal{T} \forall k \in [0, p_i]\}) \quad (1)$$

$$\forall t_i \in \mathcal{T} \forall k \in [0, p_i] \forall \tau \in \mathcal{H} \quad x_{i,k} = \tau \iff a_{i,\tau} \quad (2)$$

$$\forall t_i \in \mathcal{T} \forall k \in [0, p_i] \quad s_i \leq x_{i,k} < e_i \quad (3)$$

	$p_i$	$s_i$	$e_i$	Res.
$t_1$	3	0	6	$a$ and $b$
$t_2$	1	1	2	$a$
$t_3$	1	3	4	$a$
$t_4$	1	2	3	$b$
$t_5$	1	4	5	$b$

(a) with vs. without fragmentation

	$p_i$	$s_i$	$e_i$		
$t_1$	2	0	3	2	5
$t_2$	2	0	3	2	5
$t_3$	2	0	5	2	7

(b) ALLDIFFERENT decomposition

■ **Figure 1** An example showing that activation variables are necessary when a task requires several distinct resources (Figure 1a), and an example showing that the decomposition using the ALLDIFFERENT constraint hinders propagation (Figure 1b).

A similar decomposition holds for the constraint  $\text{PREEMPTIVENOOVERLAP}(\mathbf{s}, \mathbf{e})$  when we ignore the activation variables, and hence Constraint (2).

These two decompositions hinder propagation, i.e., BC on the decomposition is not equivalent to BC on the global constraint.

► **Theorem 11.** *Bounds consistency on Constraints 1, 2 and 3 is weaker than bounds consistency on  $\text{PREEMPTIVENOOVERLAP}(\mathcal{T}, \mathbf{s}, \mathbf{e})$ .*

**Proof.** Consider the three tasks shown in Figure 1b. The ALLDIFFERENT constraint has 6 variables in both decompositions, and there is no Hall interval and hence is BC. Therefore, the channelling constraints are also BC. Yet the values 2 to 5 are not BC for  $e_3$  in the global constraint, since that would produce an overload in the interval  $[0, 5)$ . ◀

The decomposition is weaker because the  $p_i$  units of task  $t_i$  are interchangeable. Indeed we may add the following symmetry breaking constraints:

$$\forall t_i \in \mathcal{T} \forall k \in [1, p_i) \quad x_{i,k-1} < x_{i,k} \quad (4)$$

However, there is a quadratic algorithm to achieve BC on the conjunction of an ALLDIFFERENT constraint and a set of binary precedence constraints (the ALLDIFFPREC constraint [5]), and which therefore be used to achieve BC on the conjunction of Constraints 1 and 4. Let this formulation be “the ALLDIFFPREC decomposition”.

► **Theorem 12.** *Bounds consistency on the ALLDIFFPREC decomposition is as strong as bounds consistency on  $\text{PREEMPTIVENOOVERLAP}(\mathcal{T}, \mathbf{s}, \mathbf{e})$ .*

**Proof.** Let  $\mathcal{T}$  be a set of tasks with start and end variables  $\mathbf{s}, \mathbf{e}$ , and suppose that  $\text{PREEMPTIVENOOVERLAP}(\mathcal{T}, \mathbf{s}, \mathbf{e})$  is not BC. Then there exists a task  $t_i \in \mathcal{T}$  whose upper bound is not BC. By Theorem 6, it means that there exist  $v \leq \max(e_i)$  and  $\Omega \subset \mathcal{T}$  such that  $p_\Omega + p_i > \max(e_\Omega, v) - \min(s_\Omega, \min(s_i))$ . Now, consider the assignment  $x_{i,p_i} \leftarrow v$  in the decomposition. In order to satisfy the precedences  $x_{i,k-1} < x_{i,k}$  for  $k \in [1, p_i)$ , all these variables must take values less than or equal to  $v$ . Therefore, in the decomposition, there are  $p_\Omega + p_i$  variables which must take a value in the interval  $[\min(s_\Omega, \min(s_i)), \max(e_\Omega, v))$  which is unfeasible. It follows that the assignment  $x_{i,p_i} \leftarrow v$  is not BC for the ALLDIFFPREC constraint. ◀

This decomposition, however, requires  $O(N)$  variables with  $N = \sum_{t_i \in \mathcal{T}} p_i$ , and BC can be achieved in  $O(N^2)$  time. However, it can be achieved in a time complexity that does not depend on  $N$  by direct application of Theorem 9. Indeed, there are algorithms

in  $O(n \log n)$  [29] or even in linear time (after sorting) [13] to achieve BC on at least one Edge-Finding constraint. Moreover, since there are at most  $n^2$  precedences to enforce, achieving BC on the PREEMPTIVE`NOOVERLAP` constraint can be done in time polynomial in  $n$  only.<sup>3</sup>

## 3.2 Formulation without fragmentation

When resources are *disjoint*, that is, when no task requires more than a single resource, then the problem can be modelled without representing task fragmentation. Indeed, Definition 4 ensures that a fragmentation such that no two tasks requiring that resource are processed simultaneously, and each task  $t_i$  is processed within the time interval  $[s_i, e_i)$ , in other words, we have:

$$\exists \mathbf{a} \text{ PREEMPTIVE`NOOVERLAP`}(\mathcal{T}, \mathbf{a}, \mathbf{s}, \mathbf{e}) \iff \text{PREEMPTIVE`NOOVERLAP`}(\mathcal{T}, \mathbf{s}, \mathbf{e})$$

Therefore, when activation variables ( $\mathbf{a}$ ) are not constrained otherwise, the two formulations are equivalent.

Checking this constraint can be done efficiently: the Jackson Preemptive Schedule algorithm [9, 18] finds a fragmentation, or proves that none exists in  $O(n \log n)$  time. Moreover, Theorem 6 entails that one does not need to find a witness fragmentation if the constraint propagation of the disjunctive constraint involves the overload check.

### 3.2.1 Monotonicity

Notice that the definition of the PREEMPTIVE`NOOVERLAP` constraint does not force a task  $t_i$  to be in process at time  $s_i$ , nor at time  $e_i - 1$ . In other words, start and end times are simply bounds within which the task can be processed. It follows that this constraint is *monotonic*: decreasing the start time of a task (or increasing its end time) in a satisfying assignment can never make this assignment inconsistent.

► **Definition 13** (Monotonic constraints). *Let  $\sigma_{x \leftarrow v}$  be the assignment that associates value  $v$  to variable  $x$  and that is equal to  $\sigma$  otherwise. We say that a constraint  $c$  is monotonic with respect to a function  $f : \mathbf{x} \times \mathcal{D} \mapsto \mathcal{D}$  if and only if:*

$$P_c(\sigma) = \text{true} \implies (\forall x \in S_c, P_c(\sigma_{x \leftarrow f(x, \sigma(x))}) = \text{true})$$

► **Lemma 14.** *The constraint PREEMPTIVE`NOOVERLAP`( $\mathcal{T}, \mathbf{s}, \mathbf{e}$ ) is monotonic with respect to any function that is non-increasing for start-time variables, or non-decreasing for end-time variables.*

**Proof.** The fragmentation of a task remains valid if its start time is decreased or its end time is increased. ◀

► **Corollary 15.** *If the constraint PREEMPTIVE`NOOVERLAP`( $\mathcal{T}, \mathbf{s}, \mathbf{e}$ ) is satisfiable, then, for any  $t_i \in \mathcal{T}$ , the assignments  $s_i \leftarrow \min(s_i)$  and  $e_i \leftarrow \max(e_i)$  are bounds consistent.*

**Proof.** If the constraint is satisfiable, there exists a consistent and valid assignment, and by Lemma 14, changing the value of  $s_i$  to  $\min(s_i)$  (resp.  $e_i$  to  $\max(e_i)$ ) is a non-increasing (resp. non-decreasing) operation and hence yields a consistent and valid assignment. ◀

<sup>3</sup> In practice a fix-point can be reached in far fewer iterations.

There are two consequences to the `PREEMPTIVENOOVERLAP` constraint being monotonic.

Firstly, achieving bounds consistency on the `PREEMPTIVENOOVERLAP` constraint can only prune the upper (resp. lower) bound of the start (resp. end) time variables. However, bounds of end time variables could be pruned beyond BC without losing solutions. For instance, assume that task  $t_i$  is such that  $s_i = 0, e_i = 3$  and  $p_i = 3$ . Clearly, this task requires the resource on the whole interval  $[0, 3)$  and therefore no other task can start at a time point earlier than 3. This corresponds to achieving BC on a restriction of Definition 4 where we constrain every task  $t_j \in \mathcal{T}$  to be in process at time  $s_j$  and at time  $e_j - 1$ . We have experimented with this formulation, and BC can be achieved in the same time complexity as enforcing Edge-Finding, using a slight generalisation of the propagation algorithm for BC on the `ALLDIFFERENT` constraint [27]. However, besides complexifying the definitions and the algorithm, it turns out that achieving this extra pruning is counter-productive, at least on pJSSP benchmarks.

Secondly, in the job shop scheduling problem, the only constraints besides disjunctive resources are the chain of precedences to represent the job sequences. Therefore, the start time of a task can always be extended to the end time of the previous task on that job (or to 0 if it is the first task) without invalidating the schedule. Similarly, its end time can be extended to the start time of the next task in that job. As a result, we can assume that a task ends exactly when the next task of its job starts and forbid any idle gap between the tasks of a job.

#### 4 A Constraint Programming Approach to pJSSP

From the observation made in Section 3.2, we can propose the following constraint model for the preemptive job shop scheduling problem, with  $s_{i,j}$  (resp.  $e_{i,j}$ ) standing for the start (resp. end) variable associated to the  $j$ -th task of job  $i$ , and with  $\mathcal{T} = \{t_{i,j} \mid J_i \in \mathcal{J}, \forall j \in [1, n_i]\}$ .

$$\min C_{\max} \quad (5)$$

$$s.t. \quad C_{\max} \geq e_{i,n_i} \quad \forall J_i \in \mathcal{J} \quad (6)$$

$$e_{i,j} \geq s_{i,j} + p_{i,j} \quad \forall J_i \in \mathcal{J}, \forall j \in [1, n_i] \quad (7)$$

$$e_{i,j} \leq s_{i,j+1} \quad \forall J_i \in \mathcal{J}, \forall j \in [1, n_i] \quad (8)$$

$$\text{PREEMPTIVENOOVERLAP}(\mathcal{T}_m, \mathbf{s}_m, \mathbf{e}_m) \quad \forall m \in \mathcal{M} \quad (9)$$

The objective variable  $C_{\max}$  represents the *makespan*, that is, the maximum completion time of any task (Constraint 6). Constraints 7 and 8 encode respectively the durations of the task, and the job sequences. As discussed in this section, Constraints 9 (with  $\mathcal{T}_m = \{t_{i,j} \mid M_{i,j} = m, J_i \in \mathcal{J}\}$ ,  $\mathbf{s}_m = \{s_{i,j} \mid t_{i,j} \in \mathcal{T}_m\}$  and  $\mathbf{e}_m = \{e_{i,j} \mid t_{i,j} \in \mathcal{T}_m\}$ ) are sufficient to ensure that a preemptive schedule exists, and can be computed efficiently once all start and end time variables are fixed.

Moreover, because of Corollary 15, we know that extending the end time of a task cannot violate the resource constraints. Since the only other constraints are chains of precedences, extending the end time of task up to the start time of the next task in its job (or extending its start time to the end time of the preceding task in the job) cannot violate any constraint.

We can therefore replace Constraints 6 and 7 with the following constraints:

$$s_{i,0} = 0 \quad \forall i \in \mathcal{J} \quad (10)$$

$$e_{i,j} = s_{i,j+1} \quad \forall i \in \mathcal{J}, \forall j \in [1, n_i] \quad (11)$$

$$e_{i,n_i} = C_{\max} \quad \forall i \in \mathcal{J} \quad (12)$$

Constraints 10 and 12 force the start (resp. end) time of the first (resp. last) task of every job to be equal to 0 (resp. the makespan), and Constraint 11 ensures that there is no gap between the end of a task and the start of the next task on its job. With these constraints, dominated solutions that leave a gap between two consecutive tasks of the same job are pruned out.

## 5 Experimental Results

In this section we compare our approach with the state-of-the-art approaches for the pJSSP. We first describe the two comparison methods: a recent dedicated branch-and-bound algorithm and the commercial solver CP Optimizer.

### 5.1 State-of-the-art Approaches

Most solution methods for the pJSSP are approximation algorithms [3, 15, 20] and heuristics [31]. Among the exact methods, Dantzig [10] introduced a linear programming model based on time index. Le Pape and Baptiste [24, 25] proposed a branch-and-bound procedure using classical constraint propagation techniques (timetable, disjunctive constraints and Edge-Finding) extended to preemptive problems. Ebadi and Moslehi have recently proposed two exact solution methods for the pJSSP, a mixed-integer programming (MIP) approach [11], and a dedicated branch-and-bound algorithm [12]. As our method, the MIP model requires no activation variable. It only involves variables for the start and times of the tasks, with the same guarantee that feasible (resp. optimal) solutions of this MIP correspond to feasible (resp. optimal) complete schedules, which task fragmentation can be computed e.g., via application of the Jackson's preemptive algorithm. However, in order to guarantee that the overload check is satisfied for a given resource, the model involves a set of linear constraints of size exponential in the number of tasks requiring this resource. This MIP model is less efficient than the dedicated branch-and-bound method proposed by the same authors, and hence we used the latter as reference in our experimental evaluation.

We do not compare with the recent method introduced in [21] in our experiments since this approach deals with the more general preemptive *and flexible JSSP*. It uses a logic-based Benders decomposition that splits the problem into a master problem of assigning operations to machines and into non-flexible pJSSP subproblems. The master problem is solved by mixed-integer programming while the subproblems are solved by existing approaches, such as the one introduced in this paper.

#### 5.1.1 Dedicated Branch-and-Bound

Ebadi and Moslehi's branch-and-bound procedure [12] employs a depth-first search strategy to explore the set of feasible schedules without *proactively* creating unit-length tasks.

However, since the propagation in their method is not as strong as the overload check, a different technique is used to ensure that the produced schedule follows the Jackson's preemptive rule on each machine: unit-length tasks are created lazily when branching. In the search tree, each node represents a partial schedule with a set of already scheduled unit-length tasks and a disjunctive graph representing the current precedence relations. At the root node, the set of scheduled tasks is empty, and the arcs of the graph are only the precedences between the tasks of the same job. At each decision point, the machine processing the non-scheduled unit-length task with the smallest availability date is selected. The branching strategy consists in creating a node for each such task on this machine, with

this task scheduled at its earliest possible start time. Moreover, dominance rules ensure that many unit-length tasks can be created and added to the partial schedule at once as edges in the disjunctive graph. Lower bounds are computed at each node, based on the disjunctive graph, for pruning the search tree.

This method improved the state of the art for this problem at time of publication, and in particular Le Pape and Baptiste's CP model. It was the first to solve large instances (up to 50 jobs and 10 machines) to optimality. To our knowledge, this is currently the most efficient method to solve the pJSSP problem.

### 5.1.2 CP Optimizer model

IBM CP Optimizer solver is the most efficient off-the-shelf tool in many scheduling problems. We describe in this section the standard model for the preemptive job shop scheduling problem in CP Optimizer.

CP Optimizer allows the use of specific decision variables and constraints. In particular, interval variables can be used to represent the time during which a task is processed. Interval variables are defined by a start value, an end value and a size, which are the decision variables of the problem. We denote  $t_{i,j}$  this interval variable, whose start and end time variables correspond to  $s_i$  and  $e_i$  respectively. Moreover, in this model, each preemptive task is divided into unit-length parts. Therefore, for each task  $t_{i,j}$ , we introduce  $p_{i,j}$  unit-length interval variables  $x_{i,j,k}$  with  $k \in [1, p_{i,j}]$  besides the interval variable  $t_{i,j}$ .

The problem is described as follows:

$$\min C_{\max} \quad (13)$$

$$s.t. \quad C_{\max} \geq e_{i,n_i} \quad \forall J_i \in \mathcal{J} \quad (14)$$

$$EndBeforeStart(t_{i,j}, t_{i,j+1}) \quad \forall J_i \in \mathcal{J}, \forall j \in [1, n_i] \quad (15)$$

$$EndBeforeStart(x_{i,j,k}, x_{i,j,k+1}) \quad \forall J_i \in \mathcal{J}, \forall j \in [1, n_i], \forall k \in [1, p_{i,j}] \quad (16)$$

$$Span(t_{i,j}, x_{i,j,k} : \forall k \in [1, p_{i,j}]) \quad \forall J_i \in \mathcal{J}, \forall j \in [1, n_i] \quad (17)$$

$$NOOVERLAP(x_{i,j,k} : \forall J_i \in \mathcal{J}, \forall j \in [1, n_i], \forall k \in [1, p_{i,j}] \mid M_{i,j} = m) \quad \forall m \in \mathcal{M} \quad (18)$$

The objective function (13) is to minimise the makespan. Constraints (14) define the makespan. The global constraint *EndBeforeStart* is used to model the precedence constraints, as in the two following constraint sets. Constraints (15) ensure that the tasks of the same job will be processed with respect to the job sequence. Constraints (16) aim at ordering the parts of the task and so avoid symmetries, and ensure that each part is treated one after the other. With the *Span* global constraint, Constraints (17) are used to ensure that task interval spans over all its processing parts (i.e., each task starts with its first part and ends with its last part). With the *NOOVERLAP* global constraint, Constraints (18) forbid the overlapping of tasks processed on the same machine. We denote this model  $CPO^{p=1}$ , and we experimented with several variants of this models where a task  $t_j$  is cut in fewer than  $p_j$  pieces. These variants are sound but incomplete: the optimal schedule on these models is feasible but not necessarily optimal for the original problem. However, the idea is that they should be easier to solve and hopefully approximate the optimal solution.

- $CPO^{p=\ell}$  refers to the model where each task  $t_{i,j}$  is cut into  $\lfloor \frac{p_{i,j}}{\ell} \rfloor$  subtasks of duration  $\ell$  and one task of duration  $p_{i,j} \bmod \ell$ .
- $CPO^{n=\ell}$  refers to the model where each task  $t_{i,j}$  is cut into  $\ell$  tasks of variable duration but whose total is constrained to be  $p_{i,j}$  (Constraint 19).

$$\sum_{k=1}^{\ell} \text{sizeOf}(x_{i,j,k}) = p_{i,j} \quad \forall J_i \in \mathcal{J}, \forall j \in [1, n_i] \quad (19)$$

To avoid symmetries, we made sub-task interval variables optional and add the following constraints:

$$\text{PresenceOf}(x_{i,j,k+1}) \implies \text{PresenceOf}(x_{i,j,k}) \quad \forall J_i \in \mathcal{J}, \forall j \in [1, n_i], \forall k \in [1, \ell] \quad (20)$$

$$\text{EndOf}(x_{i,j,k}) < \text{StartOf}(x_{i,j,k+1}) \quad \forall J_i \in \mathcal{J}, \forall j \in [1, n_i], \forall k \in [1, \ell] \quad (21)$$

Constraints (20) ensure that a sub-task can only be present if the previous sub-task is also present. Constraints (21) guarantees that two successive pieces of the same task do not immediately follow each other (a task is split only if it has preemption).

Interestingly, the CP Optimizer model using the constraint ALLDIFFERENT (as shown in Definition 10, with the symmetry breaking Constraints 4) instead of NOOVERLAP turned out to be much less efficient in our experiments.<sup>4</sup> This is surprising because the latter constraint is more general and yet equivalent when tasks are unit-length. We conjecture that this could be explained by some hidden preprocessing in CP Optimizer.

## 5.2 Experimental protocol

We used some standard benchmark instances available in the literature [1, 2, 14, 23, 28]. These instances were proposed for the JSSP without preemption, but are often used in the preemptive case as well. Characteristics of these benchmarks are summarised in Table 1. For each benchmark, we report the number of instances (63 in total) that compose it, the size of these instances (number of jobs  $\times$  number of machines) as well as the intervals the processing times are generated from. Detailed information on these instances is presented in [19].

All experiments were performed on three cluster nodes with Intel Xeon E5-2695 v4 CPU at 2.1 GHz with a 1 hour time cutoff. The branch-and-bound algorithm is implemented in C++, the exact  $\text{CPO}^{p=1}$  model and all of its variants ( $\text{CPO}^{p=\ell}$  and  $\text{CPO}^{n=\ell}$  for  $\ell \in \{3, 10\}$ ) are implemented with the C++ interface of CP Optimizer 12.10.

Our approach was implemented in C++ using MISTRAL [17]<sup>5</sup> with the following search strategy (corresponding to the default settings): the variable ordering uses the *minimal ratio between domain size and weighted degree* heuristic [6], with an exponential decay on the weights of 0.96 and with the *last conflict* procedure [26]. The value ordering uses binary branching with the constraints  $x \leq \lfloor \min(x) + \max(x) \rfloor / 2$  and  $x > \lfloor \min(x) + \max(x) \rfloor / 2$ , and a geometric restart policy [30] starting at 200 fails and increasing by a factor 1.05.

## 5.3 Numerical results

Figure 2a shows how many instances are optimally solved by each method as a function of time. We include the results of the incomplete variants (dashed lines) although these proofs are weaker: they show that there is no better solution for the restricted model. Nonetheless,  $\text{CPO}^{p=\ell}$  obtains fewer such proofs for  $\ell < 10$ , and  $\text{CPO}^{n=\ell}$  can only prove a single instance.

<sup>4</sup> Hence we only report results for the best model using NOOVERLAP.

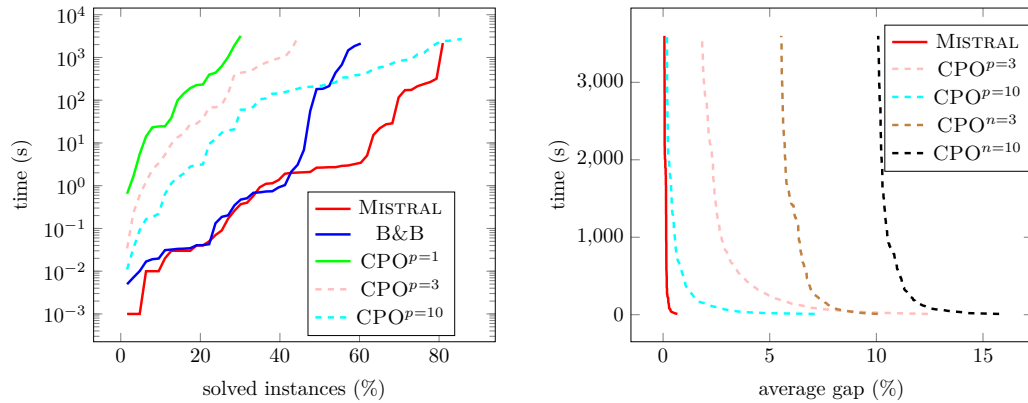
<sup>5</sup> The source code of MISTRAL is available here: <https://github.com/ehebrard/Mistral-2.0> and features the model used in our experiments.



■ **Table 1** JSSP instances characteristics.

Data set	Reference	Number of instances	Sizes	Processing times
ft6,10,20	[14]	3	6× 6, 10× 10, 20× 20 10× 5, 15× 5, 20× 5,	[1,10],[1,99]
la01-40	[23]	40	10× 10, 15× 10, 20× 10, 30× 10, 15× 15	[5,99]
abz5-9	[1]	5	10× 10, 20× 15	[50,100], [25,100], [11,40]
orb1-10	[2]	10	10× 10	[5,99]
swv16-20	[28]	5	50× 10	[1,100]

Essentially, among exact methods, CP Optimizer ( $CPO^{p=1}$ ) solves far fewer instances than other methods in the time available and is slower on the instances it does solve. For the easiest 50% of instances, both the branch-and-bound method (B&B) and MISTRAL can solve them quickly, in about 10 seconds. For the other instances, MISTRAL is faster and manages to solve 80% of the instances to optimality against 60% for the branch-and-bound.



(a) Number of solved instances over time.

(b) Gap over time.

 ■ **Figure 2** Number of proofs and gap to the best overall solution over time. Dashed lines correspond to incomplete methods.

Figure 2b shows the average gap to the best known solution over time for MISTRAL and four approximate methods namely  $CPO^{p=3}$ ,  $CPO^{p=10}$ ,  $CPO^{n=3}$  and  $CPO^{n=10}$ . We observe that none of these variants can find better solutions than MISTRAL, even considering a short calculation time. We also notice that the variants that considers fixed subtasks duration ( $CPO^{p=\ell}$ ) are more efficient than the variant that considers a fixed number of subtasks ( $CPO^{n=\ell}$ ) and that for these two variants, the fewer the subtasks, the more efficient is the method. In fact  $CPO^{p=10}$  finds better solutions than MISTRAL on two instances: **abz7** and **abz8**.

Results on individual instances are reported in Table 2. Our approach is better than the branch-and-bound on all but one instance: **orb01** where the latter method proves optimality in 2118 seconds whereas MISTRAL does not return a proof within one hour. Moreover, within the one hour cutoff, it finds the best solution over all of the methods considered in these experiments on all but two of the instances, for which approximate models are more efficient.

■ **Table 2** Results on every benchmark instance, proven optimal schedules are marked with a “\*”, best  $C_{\max}$  are in **bold font**.

Inst.	MISTRAL		B&B		CPO <sup>p=1</sup>		CPO <sup>p=10</sup>		CPO <sup>n=3</sup>	
	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)
abz5	<b>1203*</b>	213.22	1212	3600.00	1299	3600.00	1204	817.51	1266	3600.00
abz6	<b>924*</b>	27.28	<b>924*</b>	185.14	961	3600.00	<b>924</b>	205.52	957	3600.00
abz7	681	3600.00	749	3600.00	723	3600.00	<b>672</b>	3600.00	746	3600.00
abz8	694	3600.00	750	3600.00	723	3600.00	<b>677</b>	3600.00	766	3600.00
abz9	<b>691</b>	3600.00	752	3600.00	751	3600.00	695	3600.00	817	3600.00
ft06	<b>54*</b>	< 0.01	<b>54*</b>	< 0.01	<b>54*</b>	0.65	55	0.02	<b>54</b>	3600.00
ft10	<b>900*</b>	238.81	<b>900*</b>	1846.53	955	3600.00	<b>900</b>	526.71	1035	3600.00
ft20	<b>1165*</b>	0.93	<b>1165*</b>	6.89	1207	3600.00	<b>1165</b>	260.83	1204	3600.00
1a01	<b>666*</b>	< 0.01	<b>666*</b>	0.02	<b>666*</b>	24.59	<b>666</b>	0.17	670	3600.00
1a02	<b>655*</b>	0.03	<b>655*</b>	0.05	<b>655*</b>	631.14	<b>655</b>	20.91	692	3600.00
1a03	<b>597*</b>	0.08	<b>597*</b>	0.04	<b>597</b>	3600.00	<b>597</b>	19.64	635	3600.00
1a04	<b>567*</b>	0.06	<b>567*</b>	0.14	583	3600.00	<b>567</b>	17.35	599	3600.00
1a05	<b>593*</b>	< 0.01	<b>593*</b>	< 0.01	<b>593*</b>	1.65	<b>593</b>	0.04	<b>593</b>	3600.00
1a06	<b>926*</b>	< 0.01	<b>926*</b>	0.02	<b>926*</b>	5.52	<b>926</b>	0.1	<b>926</b>	3600.00
1a07	<b>890*</b>	0.03	<b>890*</b>	0.05	<b>890*</b>	38.49	<b>890</b>	3.05	<b>890</b>	3600.00
1a08	<b>863*</b>	0.04	<b>863*</b>	0.04	<b>863*</b>	144.41	<b>863</b>	2.81	<b>863</b>	3600.00
1a09	<b>951*</b>	0.04	<b>951*</b>	0.02	<b>951*</b>	23.18	<b>951</b>	0.64	<b>951</b>	3600.00
1a10	<b>958*</b>	< 0.01	<b>958*</b>	0.02	<b>958*</b>	24.34	<b>958</b>	0.19	<b>958</b>	3600.00
1a11	<b>1222*</b>	0.03	<b>1222*</b>	0.04	<b>1222*</b>	1025.62	<b>1222</b>	3.15	<b>1222</b>	3600.00
1a12	<b>1039*</b>	0.03	<b>1039*</b>	0.05	<b>1039*</b>	396.78	<b>1039</b>	1.14	<b>1039</b>	3600.00
1a13	<b>1150*</b>	< 0.01	<b>1150*</b>	0.04	<b>1150*</b>	99.89	<b>1150</b>	1.56	<b>1150</b>	3600.00
1a14	<b>1292*</b>	0.02	<b>1292*</b>	0.04	<b>1292*</b>	14.17	<b>1292</b>	0.23	<b>1292</b>	3600.00
1a15	<b>1207*</b>	0.18	<b>1207*</b>	0.19	<b>1207*</b>	1920.59	<b>1207</b>	12.38	<b>1207</b>	3600.00
1a16	<b>934*</b>	22.03	<b>934</b>	3600.00	961	3600.00	<b>934</b>	244.25	997	3600.00
1a17	<b>747*</b>	0.1	759	3600.00	794	3600.00	<b>747</b>	110.23	793	3600.00
1a18	<b>822*</b>	2.65	<b>822*</b>	676.12	850	3600.00	<b>822</b>	185.64	864	3600.00
1a19	<b>812*</b>	318.24	<b>812*</b>	1469.22	825	3600.00	814	902.75	894	3600.00
1a20	<b>871*</b>	3.21	892	3600.00	922	3600.00	875	342.32	926	3600.00
1a21	<b>1033*</b>	2179	1110	3600.00	1121	3600.00	<b>1033</b>	2674.68	1122	3600.00
1a22	<b>913*</b>	2.92	930	3600.00	982	3600.00	<b>913</b>	1695.98	1005	3600.00
1a23	<b>1032*</b>	0.38	<b>1032*</b>	1.04	1054	3600.00	<b>1032</b>	221.06	1039	3600.00
1a24	<b>909</b>	3600.00	939	3600.00	973	3600.00	910	3600.00	1001	3600.00
1a25	<b>947</b>	3600.00	983	3600.00	1071	3600.00	<b>947</b>	2428.74	1073	3600.00
1a26	<b>1218*</b>	3.45	1232	3600.00	1386	3600.00	<b>1218</b>	733	1272	3600.00
1a27	<b>1235*</b>	116.59	1346	3600.00	1360	3600.00	<b>1235</b>	2458.09	1337	3600.00
1a28	<b>1216*</b>	2.72	1255	3600.00	1402	3600.00	<b>1216</b>	1282.21	1299	3600.00
1a29	<b>1173</b>	3600.00	1225	3600.00	1325	3600.00	1196	3600.00	1283	3600.00
1a30	<b>1355*</b>	0.58	<b>1355*</b>	0.51	1499	3600.00	<b>1355</b>	143.42	1396	3600.00
1a31	<b>1784*</b>	1.91	<b>1784*</b>	2.13	1835	3600.00	<b>1784</b>	60.13	1790	3600.00
1a32	<b>1850*</b>	1.12	<b>1850*</b>	0.21	1874	3600.00	<b>1850</b>	104.93	1850	3600.00
1a33	<b>1719*</b>	2.06	<b>1719*</b>	0.35	1817	3600.00	<b>1719</b>	59.96	1719	3600.00
1a34	<b>1721*</b>	2.1	<b>1721*</b>	0.92	1836	3600.00	<b>1721</b>	397.13	1768	3600.00

Table 2: continued from previous page

Inst.	MISTRAL		B&B		CPO <sup>p=1</sup>		CPO <sup>p=10</sup>		CPO <sup>n=3</sup>	
	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)	$C_{\max}$	time (s)
1a35	<b>1888*</b>	1.37	<b>1888*</b>	0.48	1944	3600.00	<b>1888</b>	587.35	1894	3600.00
1a36	<b>1252</b>	3600.00	1297	3600.00	1358	3600.00	<b>1252</b>	3600.00	1360	3600.00
1a37	<b>1397*</b>	2.72	1411	3600.00	1500	3600.00	<b>1397</b>	2609.27	1503	3600.00
1a38	<b>1175</b>	3600.00	1255	3600.00	1308	3600.00	1191	3600.00	1350	3600.00
1a39	<b>1221*</b>	2.98	1362	3600.00	1389	3600.00	1221	2119.53	1378	3600.00
1a40	<b>1199</b>	3600.00	1311	3600.00	1357	3600.00	1234	3600.00	1332	3600.00
orb01	<b>1035</b>	3600.00	<b>1035*</b>	2118.7	1115	3600.00	1036	3600.00	1114	3600.00
orb02	<b>864*</b>	174.17	869	3600.00	887	3600.00	865	621.22	943	3600.00
orb03	<b>973</b>	3600.00	1054	3600.00	1043	3600.00	975	1191.62	1093	3600.00
orb04	<b>980*</b>	266.45	<b>980*</b>	182.2	1023	3600.00	981	382.11	1045	3600.00
orb05	<b>849*</b>	28.81	852	3600.00	870	3600.00	853	339.27	977	3600.00
orb06	<b>985</b>	3600.00	997	3600.00	1109	3600.00	<b>985</b>	870.98	1079	3600.00
orb07	<b>389*</b>	355.36	<b>389*</b>	439.51	395	3600.45	390	127.81	<b>389</b>	1511.23
orb08	<b>894*</b>	0.26	<b>894*</b>	55.94	959	3600.00	<b>894</b>	169.13	960	3600.00
orb09	<b>917*</b>	2.67	<b>917*</b>	214.58	969	3600.00	920	154.89	988	3600.00
orb10	<b>930*</b>	15.36	941	3600.00	1011	3600.00	<b>930</b>	211.73	986	3600.00
swv16	<b>2924*</b>	2.03	<b>2924*</b>	0.69	<b>2924*</b>	191.93	<b>2924</b>	1.94	<b>2924</b>	3600.00
swv17	<b>2794*</b>	0.41	<b>2794*</b>	0.7	<b>2794*</b>	3203.24	<b>2794</b>	9.5	<b>2794</b>	3600.00
swv18	<b>2852*</b>	2	<b>2852*</b>	0.74	<b>2852*</b>	232.57	<b>2852</b>	70.48	<b>2852</b>	3600.00
swv19	<b>2843*</b>	5.08	<b>2843*</b>	3.09	2970	3600.00	<b>2843</b>	141.85	<b>2843</b>	3600.00
swv20	<b>2823*</b>	1.14	<b>2823*</b>	0.75	<b>2823*</b>	226.14	<b>2823</b>	204.49	<b>2823</b>	3600.00

## 5.4 Evaluation of the compact model

Finally, we conducted further experiments to assess the gain attributable to the addition of Constraints 10, 11 and 12 in order to reduce the number of variables and eliminate solutions that leave a gap between two consecutive tasks of the same job. We ran MISTRAL on the basic model (i.e., without Constraints 10, 11 and 12) on the same benchmark instances. We only present aggregated data here.

The conclusion of these experiments is that both models (with or without) those constraints are fairly equivalent when considering the objective value only. The average gain, on the data set  $I$  containing only instances that are not proven optimal by both models, is:

$$\frac{1}{|I|} \sum_{i \in I} \frac{C_{\max}(i) - C_{\max}^*(i)}{C_{\max}(i)} = 0.03$$

where  $C_{\max}^*(i)$  denote the objective value for instance  $i$  with the extra constraints and  $C_{\max}(i)$  the objective value without these constraints. The difference is extremely small, and either model can be best on a given instance.

On instances that were proven optimal, however, the difference is clear and significant: proving optimality is done in 32.76 seconds on average with the extra constraints, whereas it takes 81.42 seconds on average without them. Moreover, one instance (1a21) can only be proven optimal within the 1h time cutoff when the extra constraints are used.

## 6 Conclusion

In this paper, we introduced a CP model for the preemptive job shop scheduling problem, and our experimental evaluation shows that it significantly improves the state of the art for this problem. The key aspect of this approach is the observation that when resources are disjoint, the Edge-Finding propagation algorithm guarantees that a preemptive schedule exists, without the need to explicitly compute a fragmentation of the tasks. This approach generalises to all disjunctive scheduling problems where resources are disjoint.

Extending this approach to general resource hypergraphs is an interesting avenue for future work. It could for instance be done in a decomposition scheme whereby after solving the model described in this paper, unit-length tasks are added, however only for those tasks whose fragmentations on two resources are in conflict.


---

## References

- 1 Joseph Adams, Egon Balas, and Daniel Zawack. The Shifting Bottleneck Procedure for Job shop Scheduling. *Management science*, 34(3):391–401, 1988.
- 2 David Applegate and William Cook. A Computational Study of the Job-shop Scheduling Problem. *ORSA Journal on computing*, 3(2):149–156, 1991.
- 3 Nikhil Bansal, Tracy Kimbrel, and Maxim Sviridenko. Job Shop Scheduling with Unit Processing Times. *Mathematics of Operations Research*, 31(2):381–389, 2006.
- 4 Philippe Baptiste, Claude Pape, and Wim Nuijten. *Constraint-Based Scheduling*. Kluwer Academic Publishers, 2001.
- 5 Christian Bessiere, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. The AllDifferent Constraint with Precedences. In *Proceedings of CPAIOR 2011*, pages 36–52, 2011.
- 6 Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting Systematic Search by Weighting Constraints. In Ramón López de Mántaras and Lorenza Saitta, editors, *Proceedings of ECAI 2004*, pages 146–150, 2004.
- 7 Peter Brucker, Svetlana A. Kravchenko, and Yuri N. Sotskov. Preemptive Job-shop Scheduling Problems with a Fixed Number of Jobs. *Mathematical Methods of Operations Research*, 49(1):41–76, 1999.
- 8 Jacques Carlier. The One-machine Sequencing Problem. *European Journal of Operational Research*, 11(1):42–47, 1982.
- 9 Jacques Carlier and Eric Pinson. A Practical Use of Jackson’s Preemptive Schedule for Solving the Job-Shop Problem. *Annals of Operations Research*, 26:269–287, 1990.
- 10 George B. Dantzig. A Machine-job Scheduling Model. *Management Science*, 6(2):191–196, 1960.
- 11 Abbas Ebadi and Ghasem Moslehi. Mathematical Models for Preemptive Shop Scheduling Problems. *Computers & Operations Research*, 39(7):1605–1614, 2012.
- 12 Abbas Ebadi and Ghasem Moslehi. An Optimal Method for the Preemptive Job Shop Scheduling Problem. *Computers & Operations Research*, 40(5):1314–1327, 2013.
- 13 Hamed Fahimi and Claude-Guy Quimper. Linear-Time Filtering Algorithms for the Disjunctive Constraint. In *Proceedings of AAAI’2014*, pages 2637–2643, 2014.
- 14 Henry Fisher and G.L. Thompson. Probabilistic Learning Combinations of Local Job-shop Scheduling Rules. In J.F. Muth and G.L. Thompson, editors, *Industrial scheduling*, pages 225–251. Prentice-Hall, Englewood Cliffs, N.J., 1963.
- 15 Leslie Ann Goldberg, Mike Paterson, Aravind Srinivasan, and Elizabeth Sweedyk. Better Approximation Guarantees for Job-shop Scheduling. *SIAM Journal on Discrete Mathematics*, 14(1):67–92, 2001.
- 16 Teofilo Gonzalez and Sartaj Sahni. Flowshop and Jobshop Schedules: Complexity and Approximation. *Operations research*, 26(1):36–52, 1978.

- 17 Emmanuel Hebrard. Mistral, a Constraint Satisfaction Library. *Third International CSP Solver Competition*, pages 31–39, 2008. URL: <http://www.cril.univ-artois.fr/CPAI08/Competition-08.pdf>.
- 18 W.A. Horn. Some Simple Scheduling Algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, 1974.
- 19 Anant Singh Jain and Sheik Meeran. Deterministic Job-shop Scheduling: Past, Present and Future. *European journal of operational research*, 113(2):390–434, 1999.
- 20 Klaus Jansen, Roberto Solis-Oba, and Maxim Sviridenko. Makespan Minimization in Job Shops: A Linear Time Approximation Scheme. *SIAM Journal on Discrete Mathematics*, 16(2):288–300, 2003.
- 21 Carla Juvin, Laurent Houssin, and Pierre Lopez. Logic-based Benders decomposition for the preemptive flexible job-shop scheduling problem. *Comput. Oper. Res.*, 152:106156, 2023. doi:10.1016/j.cor.2023.106156.
- 22 Svetlana A. Kravchenko and Yuri N. Sotskov. Complexity of the Two Machine Job-shop Scheduling Problem with a Fixed Number of Jobs. *Central European Journal for Operations Research and Economics*, 1995.
- 23 Stephen Lawrence. Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.
- 24 Claude Le Pape and Philippe Baptiste. Resource Constraints for Preemptive Job-shop Scheduling. *Constraints*, 3:263–287, 1998.
- 25 Claude Le Pape and Philippe Baptiste. Heuristic Control of a Constraint-based Algorithm for the Preemptive Job-shop Scheduling Problem. *Journal of Heuristics*, 5:305–325, 1999.
- 26 Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. Last Conflict Based Reasoning. In *Proceedings of ECAI 2006*, pages 133–137, 2006.
- 27 Alejandro López-Ortiz, Claude-Guy Quimper, John Tromp, and Peter van Beek. A Fast and Simple Algorithm for Bounds Consistency of the AllDifferent Constraint. In Georg Gottlob and Toby Walsh, editors, *Proceedings IJCAI 2003*, pages 245–250. Morgan Kaufmann, 2003. URL: <http://ijcai.org/Proceedings/03/Papers/036.pdf>.
- 28 Robert H. Storer, S. David Wu, and Renzo Vaccari. New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. *Management science*, 38(10):1495–1509, 1992.
- 29 Petr Vilím.  $O(n \log n)$  Filtering Algorithms for Unary Resource Constraint. In *Proceedings of CPAIOR 2004*, pages 335–347, 2004.
- 30 Toby Walsh. Search in a Small World. In Thomas Dean, editor, *Proceedings of IJCAI 1999*, pages 1172–1177, 1999.
- 31 Young Su Yun. Genetic Algorithm with Fuzzy Logic Controller for Preemptive and Non-preemptive Job-shop Scheduling Problems. *Computers & Industrial Engineering*, 43(3):623–644, 2002.

# Horizontally Elastic Edge Finder Rule for Cumulative Constraint Based on Slack and Density

Roger Kameugne ✉ 


Faculty of Sciences, Department of Mathematics and Computer Science, University of Maroua, Cameroon

Sévérine Fetgo Betmbe ✉ 

Faculty of Sciences, Department of Mathematics and Computer Science, University of Dschang, Cameroon

Thierry Noulamo ✉ 

UIT Fotso Victor of Bandjoun, Department of Computer Engineering, University of Dschang, Cameroon

Clémentin Tayou Djamegni ✉ 

Faculty of Sciences, Department of Mathematics and Computer Science, University of Dschang, Cameroon

UIT Fotso Victor of Bandjoun, Department of Computer Engineering, University of Dschang, Cameroon

---

## Abstract

In this paper, we propose an enhancement of the filtering power of the edge finding rule, based on the *Profile* and the *TimeTable* data structures. The minimal slack and the maximum density criteria are used to select potential task intervals for the edge finding rule. The strong detection rule of the horizontally elastic edge finder of Fetgo and Tayou is then applied on those intervals, which results in a new filtering rule, named *Slack-Density Horizontally Elastic Edge Finder*. The new rule subsumes the edge finding rule and it is not comparable to the Gingras and Quimper horizontally elastic edge finder rule and the *TimeTable* edge finder rule. A two-phase filtering algorithm of complexity  $\mathcal{O}(n^2)$  (where  $n$  is the number of tasks sharing the resource) is proposed for the new rule. Improvements based on the *TimeTable* are obtained by considering fix part of external tasks which overlap with the potential task intervals. The detection and the adjustment of the improve algorithm are further increased, while the algorithm remains quadratic. Experimental results, on a well-known suite of benchmark instances of Resource-Constrained Project Scheduling Problems, show that the propounded algorithms are competitive with the state-of-the-art algorithms, in terms of running time and tree search reduction.

**2012 ACM Subject Classification** Computing methodologies → Planning and scheduling; Theory of computation → Constraint and logic programming

**Keywords and phrases** Horizontally Elastic Scheduling, Edge Finder Rule, Profile, TimeTable, Resource-Constrained Project Scheduling Problem

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.20

**Acknowledgements** The authors thank Dr. Dany Nantchouang and Prof. Emmanuel Hebrard for their proofreading.

## 1 Introduction

Scheduling is the allocation of scarce resources to tasks or activities over time. The economic impact of the scheduling in industries and organizations [6] makes it an important combinatorial optimization problem. Industrial resources are workers, machines, electricity power and raw materials, etc. In computer science, the resources are processors while tasks are processes to be proceeded. The success of constraint programming on scheduling problems



© Roger Kameugne, Sévérine Fetgo Betmbe, Thierry Noulamo, and Clémentin Tayou Djamegni; licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 20; pp. 20:1–20:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is due to the existence of specific global constraints like UNARY [28] and CUMULATIVE [1] combined with special search strategies [2, 3, 28]. The global constraint is repeatedly called at each node of the tree search, to remove values from the variables domains when there are inconsistent by resource constraint. It is NP-hard [12] to remove all such inconsistent values from the variables domain.

The filtering algorithms used to prune the domain of variables are generally based on the relaxation of the problem. It can be called many times at each node of the tree search. Therefore, each filtering algorithm has to be fast, sound, and powerful. The global constraint CUMULATIVE [1] generally embeds timetabling [14, 21, 4] and edge finding [29, 19, 22] as basis. Some extensions and enhancement of the edge-finding filtering power have been proposed for more pruning of the domains of variables [18, 23, 30, 15, 11]. Energetic reasoning [24, 2, 7] and not-first/not-last rules [9, 16, 17, 27] are two other rules generally embedded in the CUMULATIVE constraint when the problem is highly cumulative [2]. The energetic reasoning rule subsumes all other rules except the not-first / not-last rule [2, 30, 11].

In this paper, we propose a new enhancement of the edge finder rule with the *Profile* [15] and to the *TimeTable* [30] data structures. The new rule called *Slack-Density Horizontally Elastic Edge Finder* uses the minimum slack and the maximum density criteria of [19] to select the potential edge finding task intervals on which the detection rule of [11] is going to be applied. This new rule subsumes the edge finding rule, is not comparable to the Gingras and Quimper rule [15], and the TimeTable edge finding rule [30]. A quadratic algorithm of two-phase (Detection and Adjustment) is proposed for the new rule. To further enhance the algorithm, we improve the new algorithm by considering the fix part of external tasks, which overlap with the intervals during the horizontally elastic scheduling of the task intervals. The final algorithm remains with a quadratic complexity. Experimental results on a well-known suite of benchmark instances of Resource-Constrained Project Scheduling Problems (RCPSPs) from the BL set [2] and the PSPLib set [20] show that, the propounded algorithms are competitive to the state-of-the-art algorithms, in terms of running time and tree search reduction.

The remaining part of the paper is organized as follows: Section 2 is devoted to preliminary notions and related works on previous enhancement of the edge finder rule with data structures; Section 3 presents the new rule based on the global minimum slack and maximum density criteria; Section 4 proposes an  $\mathcal{O}(n^2)$  algorithm (where  $n$  is the number of tasks sharing the resource) for the new rule; Section 5 uses the *TimeTable* data structure to improve the detection and the adjustment of the previous algorithm; Section 6 consists of an empirical evaluation of the propounded algorithms with the state-of-the-art algorithms on RCPSP instances; and Section 7 concludes the paper.

## 2 Preliminaries

This section specifies the Cumulative Scheduling Problem (CuSP) and reviews the edge finder rule and its enhancements based on data structures.

### 2.1 A Cumulative Scheduling Problem (CuSP)

In a Cumulative Scheduling Problem (CuSP), a set of tasks  $T$  has to be executed on a resource of capacity  $C$ . Each task  $i \in T$  is executed without interruption during  $p_i$  time units and uses  $c_i \leq C$  units of the resource. For a task  $i \in T$ , the earliest starting time  $est_i$  and the latest completion time  $lct_i$  are specified. A solution of a CuSP instance is an assignment of a valid starting time  $s_i$  to each task  $i \in T$  such that the resource constraint is

satisfied, i.e.,

$$\forall i \in T, \quad \text{est}_i \leq s_i \leq s_i + p_i \leq \text{lct}_i \quad (1)$$

$$\forall \tau, \quad \sum_{i \in T, s_i \leq \tau < s_i + p_i} c_i \leq C \quad (2)$$

Inequalities in (1) ensure that each task is assigned a feasible starting and ending time, while (2) enforces the resource constraint. We use the notation  $e_i$  to denote the energy of a task  $i \in T$  and it is computed with  $e_i = c_i \cdot p_i$ . Each task  $i \in T$  has an earliest completion time  $\text{ect}_i = \text{est}_i + p_i$  and a latest starting time  $\text{lst}_i = \text{lct}_i - p_i$ . These notations can be extended to nonempty sets of tasks as follows:

$$e_\Omega = \sum_{j \in \Omega} e_j, \quad \text{est}_\Omega = \min_{j \in \Omega} \text{est}_j, \quad \text{lct}_\Omega = \max_{j \in \Omega} \text{lct}_j. \quad (3)$$

By convention, for empty sets, we have:  $e_\emptyset = 0$ ,  $\text{est}_\emptyset = +\infty$  and  $\text{lct}_\emptyset = -\infty$ . Throughout the paper, we assume that for any task  $i \in T$ ,  $\text{ect}_i \leq \text{lct}_i$  and  $c_i \leq C$ , otherwise the problem has no solution. We let  $n = |T|$  denote the number of tasks and  $k = |\{c_i, i \in T\}|$  denote the number of distinct capacity demands of the tasks. The global constraint CUMULATIVE [1] is used to solve the CuSP problem, which is a NP-Hard problem [12]. The constraint removes inconsistent values from the domain of starting time variable  $s_i \in [\text{est}_i, \text{lst}_i]$  and it is NP-Hard to remove all such values.

## 2.2 Task Interval, Minimum Slack and Maximum Density

Given two tasks  $l$  and  $u$  with  $\text{est}_l < \text{lct}_u$ . A task interval denoted  $\Omega_u^l$  is a set of tasks that must run entirely within the interval  $[\text{est}_l, \text{lct}_u]$ . It is formally specified in the following definition.

► **Definition 1.** [8] *Let  $u$  and  $l$  be two tasks that satisfy  $\text{est}_l < \text{lct}_u$ . The task interval  $\Omega_u^l$  is a set of tasks specified by  $\Omega_u^l = \{j \in T \mid \text{est}_l \leq \text{est}_j \wedge \text{lct}_j \leq \text{lct}_u\}$ .*

When the condition  $\text{est}_l \leq \text{est}_j$  is released in the definition of the task interval, it is called in [29] the left cut of  $T$  by task  $u$  denoted  $\text{LCut}(T, u)$ , i.e.,  $\text{LCut}(T, u) = \{j \in T \mid \text{lct}_j \leq \text{lct}_u\}$ . The slack and the density of a task interval are concepts that are useful when designing an edge-finding algorithm [19].

► **Definition 2.** *Let  $l$  and  $u$  be two tasks that satisfy  $\text{est}_l < \text{lct}_u$ .*

1. *The slack of the task interval  $\Omega_u^l$  is the integer denoted  $sl(\Omega_u^l)$  and defined by*

$$sl(\Omega_u^l) = C \cdot (\text{lct}_u - \text{est}_l) - e_{\Omega_u^l}. \quad (4)$$

2. *The density of the task interval  $\Omega_u^l$  is the real number denoted  $ds(\Omega_u^l)$  and defined by*

$$ds(\Omega_u^l) = \frac{e_{\Omega_u^l}}{\text{lct}_u - \text{est}_l}. \quad (5)$$

In [19], the minimum slack and the maximum density criteria are successfully used to design a fast edge-finding algorithm.

► **Definition 3.** [19] *Let  $u \in T$  be a task.*

1. *The task interval of minimum slack  $\Omega_u^{\tau(u)}$  (where  $\tau(u)$  is a task depending on task  $u$ ) is the task interval satisfying the condition:*

$$sl(\Omega_u^{\tau(u)}) \leq sl(\Omega_u^l), \quad \forall l \in T \text{ with } \text{est}_l < \text{lct}_u. \quad (6)$$



2. The task interval of maximum density  $\Omega_u^{\rho(u)}$  (where  $\rho(u)$  is a task depending on task  $u$ ) is the task interval satisfying the condition:

$$ds(\Omega_u^{\rho(u)}) \geq ds(\Omega_u^l), \quad \forall l \in T \text{ with } est_l < lct_u. \quad (7)$$

These two concepts are used in [19, 18] to select the task intervals that are of interest for the (extended) edge-finder rule.

### 2.3 (Extended) Edge Finding Rule

Let  $\Omega \subset T$  be a set of tasks, and  $i \notin \Omega$  be a task. If the scheduling of the set of tasks  $\Omega$  and the contribution of task  $i$  in the interval  $[est_\Omega, lct_\Omega)$  when task  $i$  starts at  $est_i$  cause an overload of the interval  $[est_\Omega, lct_\Omega)$ , then it is deduced that all tasks in  $\Omega$  end before the end of  $i$  and is denoted  $\Omega \prec i$ . The detection rules are specified by the formulas:  $\forall \Omega \subset T, \forall i \in T \setminus \Omega$

$$e_\Omega + e_i > C \cdot (lct_\Omega - est_{\Omega \cup \{i\}}) \Rightarrow \Omega \prec i \quad (\text{EF})$$

$$\begin{cases} est_i < est_\Omega \\ \wedge \\ e_\Omega + c_i \cdot (ect_i - est_\Omega) > C \cdot (lct_\Omega - est_\Omega) \end{cases} \Rightarrow \Omega \prec i \quad (\text{EEF})$$

The rule (EF) is called edge-finder detection rule, while (EEF) is known as its extension. After each detection, the adjustment follows, using this rule:

$$\Omega \prec i \Rightarrow est_i \geq \max_{\Theta \subseteq \Omega \wedge rest(\Theta, c_i) > 0} est_\Theta + \left\lceil \frac{rest(\Theta, c_i)}{c_i} \right\rceil \quad (\text{Adj})$$

where  $rest(\Theta, c_i) = e_\Theta - (C - c_i) \cdot (lct_\Theta - est_\Theta)$  is the energy of  $e_\Theta$  that disables the starting time of task  $i$  when scheduled on a resource of capacity  $C - c_i$  in the interval  $[est_\Theta, lct_\Theta)$ . A two-phase algorithm of complexity  $\mathcal{O}(kn \log(n))$  (where  $n$  is the number of tasks and  $k$  the different number of resource demands of tasks) based on  $\Theta$ - $\Lambda$ -tree data structure was proposed in [29]. A quadratic algorithm based on the minimum slack and the maximum density of task intervals is proposed in [19].

### 2.4 TimeTable Edge Finding Rule

Research revealed that, the first enhancement of the filtering power of the edge finder rule was made with the *TimeTable* data structure in [30]. The fix part of the task is taken into account in the computation of the energy of the set of tasks, resulting in a strengthened rule. If  $i \in T$  is a task satisfying  $lst_i < ect_i$ , then the interval  $[lst_i, ect_i)$  determines the mandatory spanning interval of  $i$ . We denote by  $f(\Omega, t)$  the aggregate of the fix parts over time  $t$  by tasks in  $\Omega$ , and  $f(\Omega, [a, b])$  the fix parts aggregation over the time interval  $[a, b)$  by the tasks in  $\Omega$ .

$$f(\Omega, t) = \sum_{i \in \Omega | lst_i \leq t < ect_i} c_i; \quad (8)$$

$$f(\Omega, [a, b]) = \sum_{t \in [a, b)} f(\Omega, t). \quad (9)$$

Let  $e_\Omega^{TT}$  be the *TimeTable* energy of tasks in  $\Omega$ . This energy is equal to the energy of  $\Omega$  plus the fix energy of the tasks in  $T \setminus \Omega$  spent within the interval  $[est_\Omega, lct_\Omega)$ , i.e.,  $e_\Omega^{TT} = e_\Omega + f(T \setminus \Omega, [est_\Omega, lct_\Omega))$ . The *TimeTable* edge finder rule, denoted (TT-EF), is obtained from (EF), (EEF) and (Adj) by substituting  $e_\Omega$  and  $e_\Theta$  by  $e_\Omega^{TT}$  and  $e_\Theta^{TT}$  respectively.

A quadratic algorithm was proposed in [30] and later refined in [23]. This algorithm was used in [26] with the lazy clause generation approach, where the explanation of its propagation is incorporated as nogoods.

## 2.5 Horizontally Elastic Earliest Completion Time of Tasks Set

The computation of the earliest completion time of a set of tasks for scheduling problems with resource constraints is known to be NP-hard [12]. Depending on the way tasks are scheduled, a lower bound of the earliest completion time can be computed. According to [2], the set of tasks  $\Omega$  is said to be fully elastic scheduled, if each task  $i \in \Omega$  starts at  $\text{est}_\Omega$  and occupies a total area of  $e_i = c_i \cdot p_i$ . At any time  $t \in [\text{est}_\Omega, \text{lct}_\Omega)$ , a task  $i$  can occupy more or less  $c_i$  units of height and when time  $t$  reaches the height  $C$ , time  $t + 1$  starts being occupied. The fully elastic earliest completion time of  $\Omega$  (denoted  $\text{ect}_\Omega^F$ ) occurs when all tasks are completed. It is computed in [29] with the formula.

$$\text{ect}_\Omega^F = \left\lceil \frac{\max\{C\text{est}_{\Omega'} + e_{\Omega'} \mid \Omega' \subseteq \Omega\}}{C} \right\rceil \quad (10)$$

In [15], a new way of scheduling a set of tasks called horizontally elastic scheduling is introduced. A set of tasks  $\Omega$  is said to be horizontally elastic scheduled, when each task  $i \in \Omega$  starts at its earliest starting time  $\text{est}_i$  and cannot consume more than its required capacity at any time during the time interval  $[\text{est}_i, \text{lct}_i)$ . At any time  $t \in [\text{est}_\Omega, \text{lct}_\Omega)$ , the energy that cannot be executed, due to the limited capacity, is accumulated as an overflow and released when the resource is no longer saturated. The horizontally elastic earliest completion time of  $\Omega$  denoted  $\text{ect}_\Omega^H$  occurs when all tasks are completed. The computation of the horizontally elastic earliest completion time of tasks set is done with a data structure called *Profile* that stores the resource utilization over time. The tuples  $\langle \text{time}, \text{cap}, \Delta_{\max}, \Delta_{\text{req}} \rangle$  (where *time* is the starting time, *cap* is the remaining capacity of the resource at the starting time,  $\Delta_{\max}$  is the maximum resource available at starting time, and  $\Delta_{\text{req}}$  is the maximum resource required by tasks at starting time) are stored in a sorted linked-list whose nodes are called time points. The *Profile* is initialized with a time point of capacity  $C$  for every distinct value of *est*, *ect* and *lct*. A sufficiently large time point is added to act as a sentinel. Finally, while initializing the data structure, the pointers  $t.\text{est}_i$ ,  $t.\text{ect}_i$  and  $t.\text{lct}_i$  are used to return the time point associated with  $\text{est}_i$ ,  $\text{ect}_i$ , and  $\text{lct}_i$  for each task  $i \in T$ . The horizontally elastic earliest completion time of a set of tasks  $\Omega$  denoted  $\text{ect}_\Omega^H$  is computed using the functions  $c_{\text{req}}(t)$ ,  $c_{\text{max}}(t)$ ,  $c_{\text{cons}}(t)$  and  $ov(t)$  on the *Profile*  $P$ .

- $c_{\text{max}}(t) = \min \left( \sum_{i \in \Omega \mid \text{est}_i \leq t < \text{lct}_i} c_i, C \right)$  is the amount of resource that can be allocated to the tasks in  $\Omega$  at time  $t$ ;
- $c_{\text{req}}(t) = \sum_{i \in \Omega \mid \text{est}_i \leq t < \text{ect}_i} c_i$  is the amount of resource required at time  $t$  by the tasks in  $\Omega$  if they were all starting at their earliest starting times;
- $ov(t)$  is the overflow of energy from  $c_{\text{req}}(t)$  that cannot be executed at time  $t$  due to the limited capacity  $c_{\text{max}}(t)$ , and
- $c_{\text{cons}}(t)$  is the amount of resource that is actually consumed at time  $t$  with  $c_{\text{cons}}(t) = \min(c_{\text{req}}(t) + ov(t-1), c_{\text{max}}(t))$ ;  $ov(t) = ov(t-1) + c_{\text{req}}(t) - c_{\text{cons}}(t)$  and  $ov(-1) = 0$ .

The horizontally elastic earliest completion time occurs when all tasks are completed. Given a set of tasks  $\Omega$ , the horizontally elastic schedule of  $\Omega$  uses in [15] the function  $\text{ScheduleTasks}(\Omega, C)$  of Algorithm 1 to compute the horizontally elastic earliest completion

## 20:6 Horizontally Elastic Edge Finder Rule Based on Slack and Density

time of  $\Omega$ . The properties of this data structure is the linearity of the function *ScheduleTasks*, since the *Profile* contains at most  $4n + 1$  time points. For a set of tasks  $\Omega$ , it is proved in [15] that  $\text{ect}_{\Omega}^F \leq \text{ect}_{\Omega}^H \leq \text{ect}_{\Omega}$  where  $\text{ect}_{\Omega}$  is the earliest completion time of  $\Omega$ .

■ **Algorithm 1** *ScheduleTasks*( $\Omega, C$ ) algorithm in  $\mathcal{O}(n)$  time from [15].

---

**Input:** All time point  $t$  of the profile  $P$ ,  $\Omega$  a set of tasks,  $C$  the resource capacity.  
**Output:** the earliest completion time  $\text{ect}^H$  of the set  $\Omega$

```

1 forall  $t \in P$  do
2   |  $t.\Delta_{\max} \leftarrow 0$  and  $t.\Delta_{\text{req}} \leftarrow 0$ 
3 for  $i \in \Omega$  do
4   | Increase  $t.\text{est}_i.\Delta_{\max}$  and  $t.\text{est}_i.\Delta_{\text{req}}$  by  $c_i$ 
5   | Decrease  $t.\text{lct}_i.\Delta_{\max}$  and  $t.\text{ect}_i.\Delta_{\text{req}}$  by  $c_i$ 
6  $\text{ect} \leftarrow -\infty$ ;  $ov \leftarrow 0$ ;  $c_{\text{req}} \leftarrow 0$ ;  $S \leftarrow 0$ 
7  $t \leftarrow P.\text{first}$ 
8 while  $t.\text{time} < \text{lct}_{\Omega}$  do
9   |  $l \leftarrow t.\text{next.time} - t.\text{time}$ ;  $S \leftarrow S + t.\Delta_{\max}$ ;  $c_{\text{max}} \leftarrow \min(S, C)$ ;
   |  $c_{\text{req}} \leftarrow c_{\text{req}} + t.\Delta_{\text{req}}$ ;  $c_{\text{cons}} \leftarrow \min(c_{\text{req}} + ov, c_{\text{max}})$ 
10  if  $ov > 0 \wedge ov < (c_{\text{cons}} - c_{\text{req}}) * l$  then
11    |  $l \leftarrow \lceil \frac{ov}{c_{\text{cons}} - c_{\text{req}}} \rceil$ 
12    |  $t.\text{InsertAfter}(t.\text{time} + l, t.\text{cap})$ 
13    |  $ov \leftarrow ov + (c_{\text{req}} - c_{\text{cons}}) * l$ 
14    |  $t.\text{cap} = C - c_{\text{cons}}$ 
15    | if  $t.\text{cap} < C$  then
16      |  $\text{ect} \leftarrow t.\text{next.time}$ ;
17      |  $t \leftarrow t.\text{next}$ 
18  if  $ov > 0$  then
19    | return  $+\infty$ ;
20 return  $\text{ect}$ 

```

---

## 2.6 Existing Horizontally Elastic Edge Finder

The *Profile* data structure is used in [15] to enhance the edge finding rule with the detection rule: for all  $i, j \in T$  with  $\text{lct}_i > \text{lct}_j$ ,

$$\text{ect}_{\text{LCut}(T,j) \cup \{i\}}^H > \text{lct}_j \Rightarrow \text{LCut}(T, j) < i. \quad (\text{GQHE-EF})$$

where  $\text{LCut}(T, j) = \{k \in T \mid \text{lct}_k \leq \text{lct}_j\}$ . The detection proceeds by batching of tasks of the same height (capacity demand) and all precedences are made in  $\mathcal{O}(kn^2)$  where  $k \leq n$  is the number of distinct capacities required by the tasks, and  $n$  the number of tasks that share the resource [15]. If the free energy of height  $c_i$  of the profile of  $\text{LCut}(T, j)$  from  $\text{lct}_j$  to  $\text{est}_i$  is less than the contribution of task  $i$  in the interval  $[\text{est}_i, \text{lct}_j]$  when  $i$  starts at  $\text{est}_i$ , then it is deducted that  $\text{LCut}(T, j) < i$ . When the relation  $\text{LCut}(T, j) < i$  is detected, the adjustment phase schedules the tasks of  $\text{LCut}(T, j)$  on the lower part of the resource of capacity  $C - c_i$ . Because of the capacity restriction, it results in an overflow, scheduled on the upper part of the resource of capacity  $c_i$ . The ending time of the scheduling is where the earliest starting time of task  $i$  ( $\text{est}_i$ ) is adjusted. A quadratic algorithm is presented in [15] for the adjustment phase.

In the horizontally elastic edge finder of [11], task  $i$  is more constrained than in the previous one [15] (preemption is allowed to task  $i$ ). The new rule is based on the formulation: for all  $i, j \in T$  with  $\text{lct}_i > \text{lct}_j$ ,

$$\text{ect}_{\text{LCut}(T,j) \cup \{i'\}}^H > \text{lct}_j \Rightarrow \text{LCut}(T, j) \prec i \quad (\text{FTHE-EF})$$

where  $i'$  is a task derived from tasks  $i$  and  $j$  whose parameters  $\langle \text{est}_{i'}, \text{lct}_{i'}, p_{i'}, c_{i'} \rangle$  are

$$\langle \text{est}_i, \min(\text{ect}_i, \text{lct}_j), \min(\text{ect}_i, \text{lct}_j) - \text{est}_i, c_i \rangle.$$

It is proved in [11] that this rule is stronger than rule (GQHE-EF). Its authors propose an algorithm of complexity  $\mathcal{O}(kn^2)$  (where  $k \leq n$  is the number of distinct capacities required by the tasks and  $n$  the number of tasks sharing the resource) for the detection. A quadratic algorithm was proposed for the adjustment of this rule for an overall complexity of  $\mathcal{O}(kn^2)$  [11].

### 3 Slack-Density Horizontally Elastic Edge Finder Detection Rule

According to [15, 10, 11], the most challenging part of the strengthening of edge finding rule with *Profile* is the detection phase. The rule (GQHE-EF) proposed in [15] is a relaxation of the rule (FTHE-EF) proposed in [11]. In this section, we propose another relaxation of this rule based on the notions of minimum slack and maximum density. In fact, in [19], it is proved that a complete edge finder can be designed using the minimum slack and the maximum density of task intervals.

For a given task  $i \in T$  and for a given task  $u \in T$  with  $\text{lct}_i > \text{lct}_u$ , there exists a task  $\tau(u, i)$  such that  $\text{est}_{\tau(u, i)} \leq \text{est}_i$  and for all  $l \in T$  with  $\text{est}_l \leq \text{est}_i$  we have  $sl(\Omega_u^{\tau(u, i)}) \leq sl(\Omega_l^i)$  (See Definition 6 of [19]). We denote by  $\Omega_{\beta(i)}^{\alpha(i)}$  the task interval of minimum slack among the task interval  $\Omega_u^{\tau(u, i)}$  for all  $u \in T$  with  $\text{lct}_i > \text{lct}_u$ , i.e.,

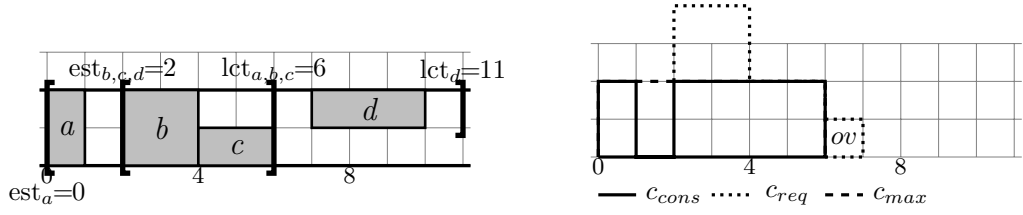
$$\Omega_{\beta(i)}^{\alpha(i)} = \text{argmin}\{sl(\Omega_u^{\tau(u, i)}) \mid \forall u \in T \text{ with } \text{lct}_i > \text{lct}_u\}. \quad (11)$$

In the following lemma, we are going to prove that each detection of the rule (EF) with pair  $(\Omega, i)$  is realized by the rule (FTHE-EF) with pair  $(\text{LCut}(T, \beta(i)), i)$ .

► **Lemma 4.** *Let  $i$  be a task and  $\Omega$  be a set of tasks such that  $i \notin \Omega$ . If the edge finding rule (EF) holds with the set  $\Omega$  and task  $i$ , then the rule (FTHE-EF) holds also with  $\text{LCut}(T, \beta(i))$  and task  $i$ .*

**Proof.** Let  $u$  be the task such that  $\text{lct}_u = \text{lct}_\Omega$ . We have  $sl(\Omega_{\beta(i)}^{\alpha(i)}) \leq sl(\Omega_u^{\tau(u, i)}) \leq sl(\Omega)$  by definition of  $\Omega_{\beta(i)}^{\alpha(i)}$  and  $\Omega_u^{\tau(u, i)}$ . The rule (EF) applies to  $\Omega$  and  $i$  (i.e.,  $e_\Omega + e_i > C(\text{lct}_\Omega - \text{est}_\Omega)$ ) is equivalent to  $sl(\Omega) < e_i$  which implies  $sl(\Omega_{\beta(i)}^{\alpha(i)}) < e_i$ . According to [11, 15], we have the following dominance properties (FTHE-EF)  $\succeq$  (GQHE-EF)  $\succeq$  (EF) + (EEF) where (A)  $\succeq$  (B) means that rule (A) subsumes rule (B) and (A) + (B) means the conjunction of rules (A) and (B). Therefore, from the inequalities  $\text{ect}_{\text{LCut}(T, \beta(i)) \cup \{i'\}}^H \geq \text{ect}_{\text{LCut}(T, \beta(i)) \cup \{i\}}^H \geq \text{ect}_{\text{LCut}(T, \beta(i)) \cup \{i\}}^F > \text{lct}_{\beta(i)}$ , it follows that the rule (FTHE-EF) holds for  $\text{LCut}(T, \beta(i))$  and task  $i$ . ◀

► **Example 5.** Consider the CuSP instance of Figure 1a where four tasks share a resource of capacity 2. The rule (EF) holds for  $\Omega = \{b, c\}$  and  $i = d$ . The tasks interval of minimum slack  $\Omega_{\beta(d)}^{\alpha(d)}$  is the set  $\Omega_{\beta(d)}^{\alpha(d)} = \{b, c\}$  and  $\text{LCut}(T, \beta(d)) = \{a, b, c\}$ . The profile of the set  $\text{LCut}(T, \beta(d)) \cup \{d\}$  is depicted in Figure 1b. The overflow remaining after time 6 allows to detect that  $\text{LCut}(T, \beta(d)) \prec d$ .



(a) CuSP instance of four tasks sharing a resource of capacity 2

(b) The profile of the set  $\text{LCut}(T, \beta(d)) \cup \{d'\} = \{a, b, c, d'\}$

■ **Figure 1** 1a: CuSP instance of four tasks sharing a resource of capacity 2 where (EF) detects  $\{b, c\} < d$ ; 1b: The profile of the set  $\text{LCut}(T, \beta(d)) \cup \{d'\} = \{a, b, c, d'\}$  and it is detected that  $\text{LCut}(T, \beta(d)) < d$ .

For a given task  $i \in T$  and for a given task  $u \in T$  with  $\text{lct}_i > \text{lct}_u$ , there exists a task  $\rho(u, i)$  such that  $\text{est}_i < \text{est}_{\rho(u, i)}$  and for all  $l \in T$  with  $\text{est}_i < \text{est}_l$  we have  $ds(\Omega_u^{\rho(u, i)}) \geq ds(\Omega_u^l)$  (See Definition 8 of [19]). We denote by  $\Omega_{\delta(i)}^{\gamma(i)}$  the task interval of maximum density among the task interval  $\Omega_u^{\rho(u, i)}$  for all  $u \in T$  with  $\text{lct}_i > \text{lct}_u$ , i.e.,

$$\Omega_{\delta(i)}^{\gamma(i)} = \text{argmax}\{ds(\Omega_u^{\rho(u, i)}) \mid \forall u \in T \text{ with } \text{lct}_i > \text{lct}_u\}. \quad (12)$$

If the extended edge finding rule (EEF) holds with a set  $\Omega$  and a task  $i$ , then the rule (FTHE-EF) holds also with  $\text{LCut}(T, \delta(i))$  and task  $i$  as it is proved in the following lemma.

► **Lemma 6.** *Let  $i$  be a task and  $\Omega$  be a set of tasks such that  $i \notin \Omega$ . If the extended edge finding rule (EEF) holds with the set  $\Omega$  and task  $i$ , then the rule (FTHE-EF) holds also with  $\text{LCut}(T, \delta(i))$  and task  $i$ .*

**Proof.** Let  $i$  be a task and  $\Omega$  be a set of tasks such that  $i \notin \Omega$ . We assume that the extended edge finding rule (EEF) holds with the set  $\Omega$  and task  $i$ . Let  $\Theta$  be the set of tasks used to update the earliest starting time of task  $i$  by rule (Adj). The proof of this lemma will distinguish the case  $\text{ect}_i \geq \text{lct}_{\delta(i)}$  from the case  $\text{ect}_i < \text{lct}_{\delta(i)}$ .

- If  $\text{ect}_i \geq \text{lct}_{\delta(i)}$  then  $ds(\Omega_{\delta(i)}^{\gamma(i)}) \geq ds(\Theta) > C - c_i$  since  $\text{rest}(\Theta, c_i) > 0$ . The rule (EEF) is detected by  $\Omega_{\delta(i)}^{\gamma(i)}$  and task  $i$  since the contribution of task  $i$  in the interval  $[\text{est}_{\gamma(i)}, \text{lct}_{\delta(i)}]$  is  $c_i(\text{lct}_{\delta(i)} - \text{est}_{\gamma(i)})$ . Thus,  $\text{ect}_{\text{LCut}(T, \delta(i)) \cup \{i\}}^H \geq \text{ect}_{\text{LCut}(T, \delta(i)) \cup \{i\}}^H \geq \text{ect}_{\text{LCut}(T, \delta(i)) \cup \{i\}}^F > \text{lct}_{\delta(i)}$ , and the rule (FTHE-EF) holds for  $\text{LCut}(T, \delta(i))$  and task  $i$ .
- We assume that  $\text{ect}_i < \text{lct}_{\delta(i)}$ . Let  $u$  be a task such that  $\text{lct}_u = \text{lct}_{\Omega}$ . The task interval  $\Omega_{\delta(i)}^{\gamma(i)}$  has the highest resource consumption spike. Therefore, the set of tasks  $\text{LCut}(T, \delta(i))$  is the most indicated to disable the start time of task  $i$ . The rest of the proof is going to distinguish two cases:  $\text{lct}_{\delta(i)} < \text{lct}_u$  and  $\text{lct}_{\delta(i)} \geq \text{lct}_u$ .
  - If  $\text{lct}_{\delta(i)} < \text{lct}_u$  then,  $\text{LCut}(T, \delta(i)) \subseteq \text{LCut}(T, u)$  and the horizontally elastic scheduling of  $\omega = \text{LCut}(T, u) \setminus \text{LCut}(T, \delta(i))$  is not enough to fill the profile from  $\text{lct}_u$  to  $\text{lct}_{\delta(i)}$ . Therefore, the slack of  $\text{LCut}(T, u)$  is greater than the one of  $\text{LCut}(T, \delta(i))$ . Task  $i$  has the same contribution in both intervals  $\text{LCut}(T, u)$  and  $\text{LCut}(T, \delta(i))$ . When this contribution is considered during the horizontally elastic scheduling of  $\text{LCut}(T, u)$ , it results to in an overload. The same overload happens when the contribution of task  $i$  is considered during the scheduling of the interval  $\text{LCut}(T, \delta(i))$ . Thus, the rule (FTHE-EF) holds for  $\text{LCut}(T, \delta(i))$  and task  $i$ .
  - If  $\text{lct}_{\delta(i)} \geq \text{lct}_u$  then,  $\text{LCut}(T, u) \subseteq \text{LCut}(T, \delta(i))$  and the horizontally elastic scheduling of  $\omega = \text{LCut}(T, \delta(i)) \setminus \text{LCut}(T, u)$  is not enough to fill the profile from  $\text{lct}_{\delta(i)}$  to  $\text{lct}_u$ .

Therefore, the slack of  $\text{LCut}(T, u)$  is greater than the one of  $\text{LCut}(T, \delta(i))$ . The contribution of task  $i$  in  $\text{LCut}(T, \delta(i))$  is greater than its contribution in  $\text{LCut}(T, u)$ . When this contribution is considered during the scheduling of  $\text{LCut}(T, u)$ , it results in an overload. Therefore, the same overload happens when the contribution of task  $i$  is considered during the scheduling of the interval  $\text{LCut}(T, \delta(i))$ . Thus, the rule (FTHE-EF) holds for  $\text{LCut}(T, \delta(i))$  and task  $i$ . ◀

According to Lemmas 4 and 6, the application of rule (FTHE-EF) to the intervals  $\text{LCut}(T, \beta(i))$  and  $\text{LCut}(T, \delta(i))$  is enough to subsume the (extended) edge finding detection rule. The new detection rule is based on the application of the rule (FTHE-EF) on the intervals  $\text{LCut}(T, \beta(i))$  and  $\text{LCut}(T, \delta(i))$  for all  $i \in T$ . This rule, denoted *Slack-Density Horizontally Elastic Edge Finder* detection, is specified by the formula: for all  $i \in T$ ,

$$\begin{cases} \text{ect}_{\text{LCut}(T, \beta(i)) \cup \{i\}}^H > \text{lct}_{\beta(i)} \Rightarrow \text{LCut}(T, \beta(i)) \prec i \\ \text{ect}_{\text{LCut}(T, \delta(i)) \cup \{i\}}^H > \text{lct}_{\delta(i)} \Rightarrow \text{LCut}(T, \delta(i)) \prec i \end{cases} \quad (\text{SDHE-EF})$$

We denote by (SDHE-EF) the filtering rule resulting from the combination of the detection rule (SDHE-EF) with the adjustment of [15]. This rule subsumes the edge finding rule and its extension combined with the rule (Adj).

► **Theorem 7.** *When the detection rule (SDHE-EF) is combined with the adjustment of [15], the resulting filtering rule subsumes the conjunction of rules (EF) and (EEF) combined with the adjustment rule (Adj).*

**Proof.** According to Lemmas 4 and 6, any detection made by (EF) and (EEF) is also done by (SDHE-EF). It is proved in (Theorem 4 of [15]) that the adjustment of [15] is better than the one performed by rule (Adj) after detection made by rules (EF) and (EEF). Therefore, the propagation of (SDHE-EF) combined with the adjustment of [15] subsumes the conjunction of rules (EF) and (EEF) combined with the adjustment rule (Adj). ◀

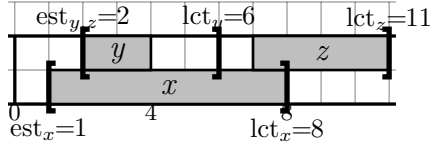
Back to the CuSP of Figure 1a, after filling the lower part of the resource of capacity 1, the overflow of three units of energy remains. This overflow is scheduled on the upper part of the resource of capacity 1. No energy is consumed in the interval  $[1, 2)$  since no task is scheduled in this interval. The scheduling of the upper part of the profile ends at time 4 which corresponds to the same adjustment value made by rule (Adj) with  $\Theta = \{b, c\}$ . Indeed,  $\text{rest}(\Theta, c_d) = 2 > 0$  and  $\text{est}_\Theta + \text{rest}(\Theta, c_d)/c_d = 4$ . It is known from [11] that (GQHE-EF) is a relaxation of (FTHE-EF). Therefore, (GQHE-EF) and (SDHE-EF) are both relaxations of (FTHE-EF) which subsumes the (extended) edge finding rule. In the following theorem, we prove that the rules (GQHE-EF) and (SDHE-EF) are not comparable, i.e., there exists propagation only performed by (GQHE-EF) and propagation only performed by (SDHE-EF).

► **Theorem 8.** *The rules (SDHE-EF) and (GQHE-EF) are not comparable.*

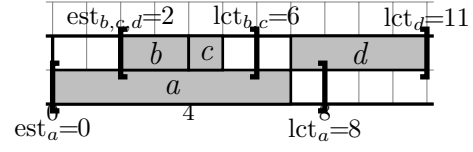
**Proof.** Consider the CuSP instance of Figure 2a, where three tasks  $\{x, y, z\}$  share a resource of capacity 2. We have  $\Omega_{\beta(z)}^{\alpha(z)} = \{x, y\}$  and the rule (SDHE-EF) detects  $\{x, y\} \prec z$ . This precedence is missed by the rule (GQHE-EF) since, in the profile of  $\{x, y\}$ , from time 8 back to time 2, we have enough free energy to schedule task  $z$ .

Converse, we consider the CuSP instance of figure 2b, where four tasks  $\{a, b, c, d\}$  share a resource of capacity 2. In the profile of  $\{a, b, c\}$ , it remains three units of free energy of height 1 when we move from time 8 back to time 2. This free energy is not enough to schedule task  $d$  and the rule (GQHE-EF) detects the relation  $\{a, b, c\} \prec d$ . We have  $\Omega_{\beta(d)}^{\alpha(d)} = \{b, c\}$  and the rule (SDHE-EF) misses the relation  $\{a, b, c\} \prec d$ . ◀

## 20:10 Horizontally Elastic Edge Finder Rule Based on Slack and Density



(a) CuSP instance of three tasks sharing a resource of capacity 2



(b) CuSP instance of four tasks sharing a resource of capacity 2

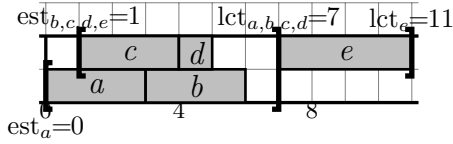
■ **Figure 2** 2a: (SDHE-EF) detects  $\{x, y\} < z$  while (GQHE-EF) misses the detection; (2b): (GQHE-EF) detects  $\{a, b, c\} < d$  while (SDHE-EF) misses the detection.

The rule (SDHE-EF) is a relaxation of the rule (FTHE-EF), since it restricts the detection to two intervals instead of  $n$ . It is proved in [11] that the rule (FTHE-EF) is not comparable to the rule (TT-EF). This result remains between (SDHE-EF) and (TT-EF) as it is proved in the following theorem.

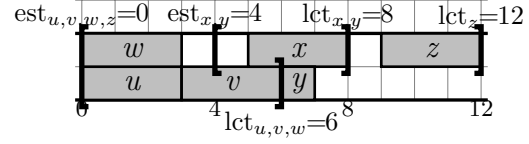
► **Theorem 9.** *The rules (SDHE-EF) and (TTEF) are not comparable.*

**Proof.** Consider the CuSP instance of Figure 3a where five tasks  $\{a, b, c, d, e\}$  share a resource of capacity 2. We have  $\Omega_{\beta(d)}^{\alpha(d)} = \{a, b, c, d\}$  and the rule (SDHE-EF) detects  $\{a, b, c, d\} < e$ . This precedence is missed by the rule (TTEF). Indeed, none of the tasks has a fix part and the edge finding rule (EF) fails to detect  $\{a, b, c, d\} < e$ .

Conversely, we consider the CuSP instance of figure 3b where six tasks  $\{u, v, w, x, y, z\}$  share a resource of capacity 2. When the fix part of task  $x$  which intersects the interval  $[0, 6]$  is considered, it is detected that  $\{u, v, w\} < z$  and the rule (TTEF) holds. This detection is missed by the rule (SDHE-EF), since the contribution of task  $x$  in the interval  $\text{LCut}(T, u) = \{u, v, w\}$  is not considered. ◀



(a) CuSP instance of five tasks sharing a resource of capacity 2



(b) CuSP instance of six tasks sharing a resource of capacity 2

■ **Figure 3** 3a: (SDHE-EF) detects  $\{a, b, c, d\} < e$  while (TTEF) misses the detection; 3b: (TTEF) detects  $\{u, v, w\} < z$  while (SDHE-EF) misses the detection.

## 4 Slack-Density Horizontally Elastic Edge Finder Algorithm

The detection algorithm identifies the right bound of the task intervals  $\Omega_{\beta(i)}^{\alpha(i)}$  and  $\Omega_{\delta(i)}^{\gamma(i)}$  for any task  $i \in T$ . To do so, the function *ComputesBound()* receives as input the set  $T_{\text{est}}$  (resp.  $T_{\text{lct}}$ ) of tasks sorted in increasing order of est (resp. lct). The global maximum density and minimum slack are initialized at line 2, while the local maximum density and minimum slack are initialized at line 4 of the loop of line 3. The loop of line 5 updates the value of the local maximum density at line 9 and the global value at line 12. Similarly, the loop of line 14 updates the value of the local minimum slack at line 17 and the global value at line 19.

The function *ComputesBound* has a quadratic complexity as it is shown in Proposition 10.

► **Proposition 10.** *ComputesBound runs in  $\mathcal{O}(n^2)$  in time.*

■ **Algorithm 2** *ComputesBound*( $T_{est}, T_{lct}$ ) in  $\mathcal{O}(n^2)$  time.

---

**Input:**  $T_{est}$  and  $T_{lct}$  the sets of tasks sorted in increasing order of est and lct respectively .

**Output:** The bounds  $\beta(i)$  and  $\delta(i)$  for all task  $i \in T$ .

```

1 forall  $i \in T_{est}$  do
2   |  $\beta(i) \leftarrow -1, \delta(i) \leftarrow -1, maxDensity(i) \leftarrow 0, minSlack(i) \leftarrow \infty$ 
3 forall  $j \in T_{lct}$  with  $lct_j < lct_{j+1}$  do
4   |  $E \leftarrow 0, maxD \leftarrow 0, minS \leftarrow \infty$ 
5   | forall  $k \in T_{est}$  in reverse order of est do
6   |   | if  $lct_k \leq lct_j$  then
7   |   |   |  $E \leftarrow E + e_k, density \leftarrow E/(lct_j - est_k)$ 
8   |   |   | if  $density \geq maxD$  then
9   |   |   |   |  $maxD \leftarrow density$ 
10  |   | else
11  |   |   | if  $maxD \geq maxDensity(k)$  then
12  |   |   |   |  $maxDensity(k) \leftarrow maxD, \delta(k) \leftarrow j$ 
13  |   |   |  $Energy(k) \leftarrow E$ 
14  |   | forall  $k \in T_{est}$  do
15  |   |   |  $slack \leftarrow C(lct_j - est_k) - Energy(k)$ 
16  |   |   | if  $slack < minS$  then
17  |   |   |   |  $minS \leftarrow slack$ 
18  |   |   | if  $lct_k > lct_j \wedge minS < minSlack(k)$  then
19  |   |   |   |  $minSlack(k) \leftarrow minS, \beta(k) \leftarrow j$ 

```

---

**Proof.** The loop of line 3 of complexity  $\mathcal{O}(n)$  calls sequentially the loops of lines 5 and 14 of complexity  $\mathcal{O}(n)$  each. Therefore, the overall complexity of *ComputesBound* is  $\mathcal{O}(n(n+n)) = \mathcal{O}(n^2)$ . ◀

In Algorithm 3, the bounds of the task interval of minimum slack and maximum density are computed at line 1. In the loop of line 2, for any unscheduled task (line 3), it is checked at line 6 whether the task interval of minimum slack precedes the task and the relation is recorded at line 7. When no detection is made by the task interval of minimum slack (line 8), the task interval of maximum density is used to check the relation at line 10 and the relation is recorded at line 11.

The complexity of *Slack-Density Horizontally Elastic Edge Finder* is analyzed in Proposition 11.

► **Proposition 11.** *Slack-Density Horizontally Elastic Edge Finder runs in  $\mathcal{O}(n^2)$  in time.*

**Proof.** The linear function *ScheduleTasks* is sequentially called twice in the linear loop of line 2. Combined with the quadratic complexity of the function *ComputesBound*, the overall complexity of *Slack-Density Horizontally Elastic Edge Finder* is  $\mathcal{O}(n^2 + n(n+n)) = \mathcal{O}(n^2)$ . ◀

We combine this detection algorithm with the quadratic adjustment algorithm of [15] proposed for the rule (GQHE-EF). Therefore, the complexity of the two-phase algorithm (Detection and Adjustment) is  $\mathcal{O}(n^2)$ . To our knowledge, this is the first quadratic horizontally elastic edge finder algorithm.



■ **Algorithm 3** *Slack-Density Horizontally Elastic Edge Finder*( $T_{\text{est}}, T_{\text{lct}}$ ) in  $\mathcal{O}(n^2)$  time.

---

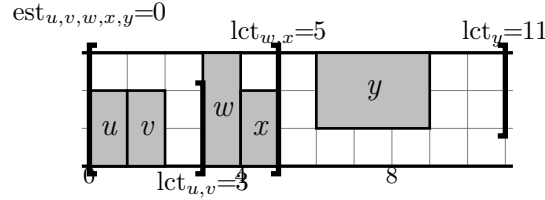
**Input:**  $T_{\text{est}}$  and  $T_{\text{lct}}$  the sets of tasks sorted in increasing order of est and lct respectively.

**Output:** Prec: the precedence relation.

```

1  $\beta, \delta \leftarrow \text{ComputesBound}(T_{\text{est}}, T_{\text{lct}})$ 
2 forall  $i \in T_{\text{lct}}$  do
3   if  $ect_i < lct_i$  then
4     if  $\beta(i) \neq -1$  then
5        $ect \leftarrow \text{ScheduleTask}(\text{LCut}(T, \beta(i)) \cup \{i'\})$ 
6       if  $ect > lct_{\beta(i)}$  then
7         Prec( $i$ )  $\leftarrow \beta(i)$ 
8     if  $\text{Prec}(i) \neq -1 \wedge \delta(i) \neq -1$  then
9        $ect \leftarrow \text{ScheduleTask}(\text{LCut}(T, \delta(i)) \cup \{i'\})$ 
10      if  $ect > lct_{\delta(i)}$  then
11        Prec( $i$ )  $\leftarrow \delta(i)$ 
    
```

---



■ **Figure 4** A CuSP instance of five tasks sharing a resource of capacity 3.

► **Example 12.** Consider the CuSP instance of Figure 4 where five tasks share a resource of capacity 3.

In this example,  $\Omega_u^u = \{u, v\}$ ,  $sl(\Omega_u^u) = 5$  while  $\Omega_w^u = \{u, v, w, x\}$ ,  $sl(\Omega_w^u) = 6$ . Therefore,  $\Omega_{\beta(y)}^{\alpha(y)} = \{u, v\}$  and the rules (EF) and (SDHE-EF) detect that  $\Omega_{\beta(y)}^{\alpha(y)} < y$ .

## 5 Improvements

To improve our algorithm, we have taken into account the fix part of tasks  $T \setminus \text{LCut}(T, \beta(i)) \cup \{i\}$  (resp.  $T \setminus \text{LCut}(T, \delta(i)) \cup \{i\}$ ) which overlap with  $\text{LCut}(T, \beta(i))$  (resp.  $\text{LCut}(T, \delta(i))$ ) during the computation of  $ect_{\text{LCut}(T, \beta(i)) \cup \{i'\}}^H$  (resp.  $ect_{\text{LCut}(T, \delta(i)) \cup \{i'\}}^H$ ) and the adjustment. For a given task  $j \in T \setminus \text{LCut}(T, \beta(i)) \cup \{i\}$  with a fix part (i.e.,  $lst_j < ect_j$ ), which overlap with  $\text{LCut}(T, \beta(i))$  (i.e.,  $lst_j < lct_{\beta(i)}$ ), a new task denoted  $f(j, \beta(i))$  is deduced with the following attributes  $\langle lst_j, \min(ect_j, lct_{\beta(i)}), \min(ect_j, lct_{\beta(i)}) - lst_j, c_j \rangle$ . We denote by  $f(T \setminus \text{LCut}(T, \beta(i)) \cup \{i\}, \beta(i))$  the set of deduced fix parts of tasks which overlap with  $\text{LCut}(T, \beta(i))$ . Analogously, the set of fix part of tasks  $f(T \setminus \text{LCut}(T, \delta(i)) \cup \{i\}, \delta(i))$  which overlap with  $\text{LCut}(T, \delta(i))$  is considered during the computation of  $ect_{\text{LCut}(T, \delta(i)) \cup \{i'\}}^H$ .

To do so, we first extend the initial time points, by adding those corresponding to the  $lst_i$  for all  $i \in T$ . The number of time points moves from  $4n + 1$  to  $5n + 1$  and the function *ScheduleTasks* remains linear. During the initialization of increments, for any task  $j \in T \setminus \text{LCut}(T, \beta(i)) \cup \{i\}$  (resp.  $j \in T \setminus \text{LCut}(T, \delta(i)) \cup \{i\}$ ), if tasks  $j$  has a fix part which overlap with  $\text{LCut}(T, \beta(i))$  (resp.  $\text{LCut}(T, \delta(i))$ ),  $\Delta_{max}$  and  $\Delta_{req}$  are increased by  $c_j$  at  $t.lst_j$ . If  $ect_j < lct_{\beta(i)}$  (resp.  $ect_j < lct_{\delta(i)}$ ) then  $\Delta_{max}$  and  $\Delta_{req}$  are decreased by  $c_j$

at  $t.ect_j$ . If  $ect_j \geq lct_{\beta(i)}$  (resp.  $ect_j \geq lct_{\delta(i)}$ ) then  $\Delta_{max}$  and  $\Delta_{req}$  are decreased by  $c_j$  at  $t.lct_{\beta(i)}$  (resp.  $t.lct_{\delta(i)}$ ). For the adjustment, the fix part tasks  $f(T \setminus \text{LCut}(T, \beta(i)) \cup \{i\}, \beta(i))$  (resp.  $f(T \setminus \text{LCut}(T, \delta(i)) \cup \{i\}, \delta(i))$ ) is considered during the computation of the maximum overflow obtained when the set  $\text{LCut}(T, \beta(i))$  (resp.  $\text{LCut}(T, \delta(i))$ ) is scheduled on a resource of restricted capacity  $C - c_i$ . The additional fix part of tasks can increase both the detection and the adjustment of the *Slack-Density Horizontally Elastic Edge Finder* algorithm. For example, in the CuSP of Figure 2b, when the fix part of task  $a$  is considered, the task interval  $\Omega_{\beta(d)}^{\alpha(d)} = \{b, c\}$  can detect that  $\Omega_{\beta(d)}^{\alpha(d)} < d$  and the adjustment follows. It is also the case for the CuSP instance of Figure 3b, when the fix part of task  $x$  is considered.

## 6 Experimental Results

The new algorithm with improvements was compared to the state-of-the-art horizontally elastic edge finder algorithms ([15, 11]) on Resource-Constrained Project Scheduling Problems (RCPPSPs). Comparisons are done on instances of benchmark suites of RCPSP of libraries BL [2] and PSPLib [20] (on sets j30, j60 and j90). Starting time of tasks and makespan was used as variables of the model. They were constrained by the precedence graph and resource limitations. Each resource was modeled with a single CUMULATIVE constraint [1]. The lower bound of the makespan variable was updated with the horizontally elastic earliest completion time of the whole set of tasks  $T$ . The TimeTabling algorithm of [21] was added to the common core model. The optimization is based on depth-first search and restart. Anytime a solution is found, the resolution restarts with an additional constraint, which states that the next makespan must be (strictly) better than the current one. The optimum solution is the last solution found within the time limit.

Three configurations of the global constraint CUMULATIVE was considered. The first one denoted GQHE-EF uses the horizontally elastic edge-finder from [15], while the second one denoted FTHE-EF considers the horizontally elastic edge-finder algorithm of [11]. The last configuration denoted SDHE-EF is based on the Algorithm 3 for detection, combined with the adjustment algorithm of [15], all with improvements of Section 5.

Three strategies of selection of variables and values were used to speed up the solving process. Static heuristic were unscheduled tasks are selected in the chronological order of the indices and assigned to their lower bound value. COS + DomOvWDeg where the Conflict Ordering Search heuristic (COS) [13] is combined with the default search strategy of Choco solver *domOverWDeg* [5]. COS + Smallest where the COS is combined to the smallest heuristic (a variable of smallest lower bound among those not yet instantiated is selected and assigned to its lower bound). The implementation was done in Java using Choco solver 4.10.8 [25]. Any search taking more than 10 minutes was counted as a failure.

In Table 1, the column “solve” indicates the number of instances solved to optimality by each configuration, “common solve” indicates the numbers of instances each configuration solves commonly with the baseline configuration SDHE-EF. The column “back<sub><</sub>” (resp. “back<sub>></sub>”) indicates the number of instances were each configuration reduces (resp. need) more backtracks than the baseline configuration. The column “back” (resp. “time”) indicates the average number of backtracks (resp. runtime in sec) on common solved instances for each configuration with the baseline configuration.

SDHE-EF always solves more instances than the other configurations, whatever the heuristic selection considers. On common instances solves with static heuristic by SDHE-EF and GQHE-EF (resp. SDHE-EF and FTHE-EF) 100 (resp. 67) instances were solved by SDHE-EF with fewer backtracks (see Table 1). Figures 5a, 5c and 5e compare the number of

■ **Table 1** Number of instances solved, commonly solved with the baseline configuration SDHE-EF, where the number of backtracks is less (resp. great) than the one of the baseline. The average number of backtracks and runtime (in sec) are also provided for commonly solved instances. The one of the baseline configuration are in brackets.

	solve	common solve	back <sub>&lt;</sub>	back <sub>&gt;</sub>	back	time
Static						
GQHE-EF	1037	1030	72	100	<b>5569</b> (5939)	8.294 ( <b>4.666</b> )
FTHE-EF	1031	1024	93	67	<b>6997</b> (7810)	6.552 ( <b>3.695</b> )
SDHE-EF	<b>1044</b>	-	-	-	-	-
COS + DomOvWDeg						
GQHE-EF	1097	1093	213	238	<b>4675</b> (4912)	9.377 ( <b>5.207</b> )
FTHE-EF	1086	1085	209	219	3304 ( <b>3209</b> )	11.788 ( <b>3.479</b> )
SDHE-EF	<b>1121</b>	-	-	-	-	-
COS + Smallest						
GQHE-EF	1053	1053	32	87	6219 ( <b>5859</b> )	7.976 ( <b>4.489</b> )
FTHE-EF	1049	1049	42	69	2417 ( <b>2360</b> )	10.003 ( <b>3.132</b> )
SDHE-EF	<b>1060</b>	-	-	-	-	-

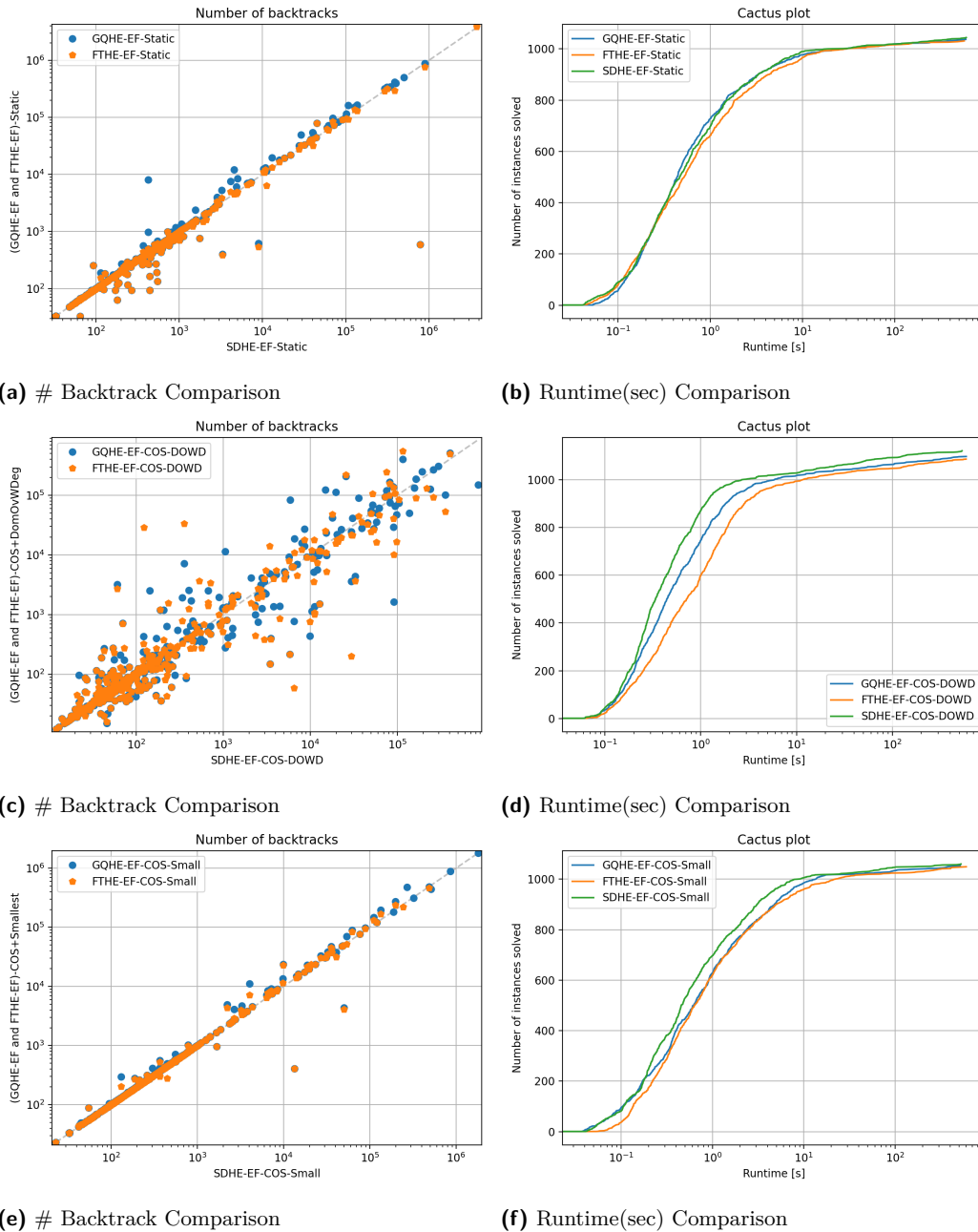
backtracks of GQHE-EF and FTHE-EF with the baseline configuration SDHE-EF on each instances commonly solved. The points above the line  $y = x$  shown that SDHE-EF records less backtracks than the other configurations. Figures 5b, 5d and 5f represent the number of solved instances as a function of time for each configuration. The running time gains between SDHE-EF and the two other configurations for static heuristic, is not significant enough as it is illustrated in Figure 5b.

When the heuristic COS + DomOvWDeg is considered, the average number of backtracks of SDHE-EF is 3209, which is less than the average number of backtracks of FTHE-EF (3304), on commonly solved instances with baseline configuration (see Table 1). The average number of backtracks of GQHE-EF (4675) is less than the one of SDHE-EF (4912). The average runtime of SDHE-EF is half of the other configurations whatever the heuristic selection considered (see Table 1). The running time gains between SDHE-EF and the two other configurations is more prominent as illustrated in Figure 5d and 5f.

## 7 Conclusion

In this paper, we have proposed a new strengthening of the edge finder rule based on the *Profile* data structure. The minimum slack and the maximum density are used to select the potential task interval for the edge finder rule. The application of the *Profile* on those task intervals results in a new rule named *Slack-Density Horizontally Elastic Edge Finder*. This new rule subsumes the classic (extended) edge finder rule, but is not comparable to Gingras and Quimper’s horizontally elastic edge finder rule [15], and the *TimeTable* edge finding rule [30]. A quadratic detection algorithm for the new rule is combined with the quadratic adjustment algorithm of [15], which results in an overall complexity of  $\mathcal{O}(n^2)$  in time. Improvements based on the *TimeTable* data structure are obtained by considering fix part of external tasks during the horizontally elastic scheduling of task intervals. Experimental results showed that our new algorithm is competitive with start-of-the-art strengthening of edge finding algorithms, with *Profile* for time and tree search reduction. It is a good trade-off between the speed and the filtering power for the rule (FTHE-EF) of [11].

Future works will be devoted to the utilization of the *Profile* data structure in the energetic reasoning to reduce its complexity and increase its filtering power.



■ **Figure 5** 5a, 5c and 5e: Comparison of the number of backtracks of different configurations of the CUMULATIVE constraint. SDHE-EF is used as the baseline model. 5b, 5d and 5f : Number of solved instances as a function of time for the different configurations of the CUMULATIVE constraint.

**References**

- 1 Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to Solve Complex Scheduling and Placement Problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993. URL: <https://hal.archives-ouvertes.fr/hal-00442821>.
- 2 P. Baptiste, C.L. Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. International Series in Operations Research & Manage-

- ment Science. Springer US, 2012. URL: <https://books.google.cm/books?id=qUzhBwAAQBAJ>.
- 3 Philippe Baptiste, Philippe Laborie, Claude Le Pape, and Wim Nuijten. Constraint-based scheduling and planning. In *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 761–799. Elsevier, 2006.
  - 4 Nicolas Beldiceanu and Mats Carlsson. A new multi-resource cumulatives constraint with negative heights. In *CP*, volume 2470 of *Lecture Notes in Computer Science*, pages 63–79. Springer, 2002.
  - 5 Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *ECAI*, pages 146–150. IOS Press, 2004.
  - 6 Rajkumar Buyya, M. Manzur Murshed, David Abramson, and Srikumar Venugopal. Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimization algorithm. *Softw. Pract. Exp.*, 35(5):491–512, 2005.
  - 7 Jacques Carlier, Eric Pinson, Abderrahim Sahli, and Antoine Jouglet. An  $O(n^2)$  algorithm for time-bound adjustments for the cumulative scheduling problem. *Eur. J. Oper. Res.*, 286(2):468–476, 2020.
  - 8 Yves Caseau and François Laburthe. Improved CLP scheduling with task intervals. In *ICLP*, pages 369–383. MIT Press, 1994.
  - 9 Hamed Fahimi, Yanick Ouellet, and Claude-Guy Quimper. Linear-time filtering algorithms for the disjunctive constraint and a quadratic filtering algorithm for the cumulative not-first not-last. *Constraints An Int. J.*, 23(3):272–293, 2018.
  - 10 Séverine Fetgo Betmbe and Clémentin Tayou Djamegni. Horizontally Elastic Edge-Finder Algorithm for Cumulative Resource Constraint Revisited. In *CARI 2020*, THIES, Senegal, October 2020. URL: <https://hal.archives-ouvertes.fr/hal-02931383>.
  - 11 Séverine Fetgo Betmbe and Clémentin Tayou Djamegni. Horizontally elastic edge-finder algorithm for cumulative resource constraint revisited. *Oper. Res. Forum*, 3(65), 2022. doi: 10.1007/s43069-022-00172-6.
  - 12 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
  - 13 Steven Gay, Renaud Hartert, Christophe Lecoutre, and Pierre Schaus. Conflict ordering search for scheduling problems. In *CP*, volume 9255 of *Lecture Notes in Computer Science*, pages 140–148. Springer, 2015.
  - 14 Steven Gay, Renaud Hartert, and Pierre Schaus. Simple and scalable time-table filtering for the cumulative constraint. In *CP*, volume 9255 of *Lecture Notes in Computer Science*, pages 149–157. Springer, 2015.
  - 15 Vincent Gingras and Claude-Guy Quimper. Generalizing the edge-finder rule for the cumulative constraint. In *IJCAI*, pages 3103–3109. IJCAI/AAAI Press, 2016.
  - 16 Roger Kameugne, Séverine Betmbe Fetgo, Vincent Gingras, Yanick Ouellet, and Claude-Guy Quimper. Horizontally elastic not-first/not-last filtering algorithm for cumulative resource constraint. In *CPAIOR*, volume 10848 of *Lecture Notes in Computer Science*, pages 316–332. Springer, 2018.
  - 17 Roger Kameugne and Laure Pauline Fotso. A cumulative not-first/not-last filtering algorithm in  $O(n \log n)$ . *Indian Journal of Pure and Applied Mathematics*, 44:95–115, 2013.
  - 18 Roger Kameugne, Laure Pauline Fotso, and Joseph D. Scott. A quadratic extended edge-finding filtering algorithm for cumulative resource constraints. *International Journal of Planning and Scheduling (IJPS)*, 1(4), 2013.
  - 19 Roger Kameugne, Laure Pauline Fotso, Joseph D. Scott, and Youcheu Ngo-Kateu. A quadratic edge-finding filtering algorithm for cumulative resource constraints. *Constraints An Int. J.*, 19(3):243–269, 2014.
  - 20 Rainer Kolisch and Arno Sprecher. Psplib – a project scheduling problem library. *EUROPEAN JOURNAL OF OPERATIONAL RESEARCH*, 96:205–216, 1996.

- 21 Arnaud Letort, Nicolas Beldiceanu, and Mats Carlsson. A scalable sweep algorithm for the cumulative constraint. In *CP*, volume 7514 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 2012.
- 22 Luc Mercier and Pascal Van Hentenryck. Edge finding for cumulative scheduling. *INFORMS J. Comput.*, 20(1):143–153, 2008.
- 23 Pierre Ouellet and Claude-Guy Quimper. Time-table extended-edge-finding for the cumulative constraint. In *CP*, volume 8124 of *Lecture Notes in Computer Science*, pages 562–577. Springer, 2013.
- 24 Yanick Ouellet and Claude-Guy Quimper. A  $o(n \log^2 n)$  checker and  $o(n^2 \log n)$  filtering algorithm for the energetic reasoning. In *CPAIOR*, volume 10848 of *Lecture Notes in Computer Science*, pages 477–494. Springer, 2018.
- 25 Charles Prud’homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Solver Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. URL: <http://www.choco-solver.org>.
- 26 Andreas Schutt, Thibaut Feydy, and Peter J. Stuckey. Explaining time-table-edge-finding propagation for the cumulative resource constraint. In *CPAIOR*, volume 7874 of *Lecture Notes in Computer Science*, pages 234–250. Springer, 2013.
- 27 Andreas Schutt and Armin Wolf. A new  $O(n^2 \log n)$  not-first/not-last pruning algorithm for cumulative resource constraints. In *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 445–459. Springer, 2010.
- 28 Petr Vilím. *Global Constraints in Scheduling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, KTIML MFF, Universita Karlova, Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic, August 2007. URL: <http://vilim.eu/petr/disertace.pdf>.
- 29 Petr Vilím. Edge finding filtering algorithm for discrete cumulative resources in  $O(kn \log n)$ . In *CP’09: Proceedings of the 15th international conference on Principles and practice of constraint programming*, pages 802–816, Berlin, Heidelberg, 2009. Springer-Verlag. URL: <http://vilim.eu/petr/cp2009.pdf>.
- 30 Petr Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In *CPAIOR*, volume 6697 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2011.



# Exploring Hydrogen Supply/Demand Networks: Modeller and Domain Expert Views

**Matthias Klapperstueck** ✉ 

Department of Human-Centred Computing, Faculty of IT, Monash University, Clayton, VIC, Australia

**Frits de Nijs** ✉ 

Department of Data Science and AI, Faculty of IT, Monash University, Clayton, VIC, Australia  
ARC Industrial Training and Transformation Centre OPTIMA, Carlton, VIC, Australia

**Ilankaikone Senthooan** ✉ 

Department of Data Science and AI, Faculty of IT, Monash University, Clayton, VIC, Australia  
ARC Industrial Training and Transformation Centre OPTIMA, Carlton, VIC, Australia

**Jack Lee-Kopij**

Woodside Energy Ltd., Perth, Australia

**Maria Garcia de la Banda** ✉ 

Department of Data Science and AI, Faculty of IT, Monash University, Clayton, VIC, Australia  
ARC Industrial Training and Transformation Centre OPTIMA, Carlton, VIC, Australia

**Michael Wybrow** ✉ 

Department of Human-Centred Computing, Faculty of IT, Monash University, Clayton, VIC, Australia

---

## Abstract

Energy companies are considering producing renewable fuels such as hydrogen/ammonia. Setting up a production network means deciding where to build production plants, and how to operate them at minimum electricity and transport costs. These decisions are complicated by many factors including the difficulty in obtaining accurate current data (e.g., electricity price and transport costs) for potential supply locations, the accuracy of data predictions (e.g., for demand and costs), and the need for some decisions to be made due to external (not modelled) factors. Thus, decision-makers need access to a user-centric decision system that helps them visualise, explore, interact and compare the many possible solutions of many different scenarios. This paper describes the system we have built to support our energy partner in making such decisions, and shows the advantages of having a graphical user-focused interactive tool, and of using a high-level constraint modelling language (MINIZINC) to implement the underlying model.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Theory of computation → Integer programming; Human-centered computing → Information visualization

**Keywords and phrases** Facility Location, Hydrogen Supply Chain, Human-Centric Optimisation

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.21

**Funding** Woodside Energy Ltd.

## 1 Introduction

Moving away from fossil fuels is needed to achieve (net) zero carbon emissions [12]. Part of this move focuses on the production of hydrogen and ammonia as storage carriers for their use in mobile applications and seasonal storage. It is estimated the world production of hydrogen needs to more than double by 2030 [9] and, given around 50% of the current production [5] relies on natural gas, the demand for “green” hydrogen is even higher. This



© Matthias Klapperstueck, Frits de Nijs, Ilankaikone Senthooan, Jack Lee-Kopij, Maria Garcia de la Banda, and Michael Wybrow;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 21; pp. 21:1–21:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



## 21:2 Exploring Hydrogen Supply/Demand Networks

requires energy companies to expand their production capacity as efficiently as possible. To achieve this, energy companies need to solve a complex combinatorial optimisation problem with energy sourcing, operation, transport and demand constraints that can be summarised as making two interdependent decisions – *where to build* the hydrogen production plants and *how to operate* the production and supply process – that minimise costs.

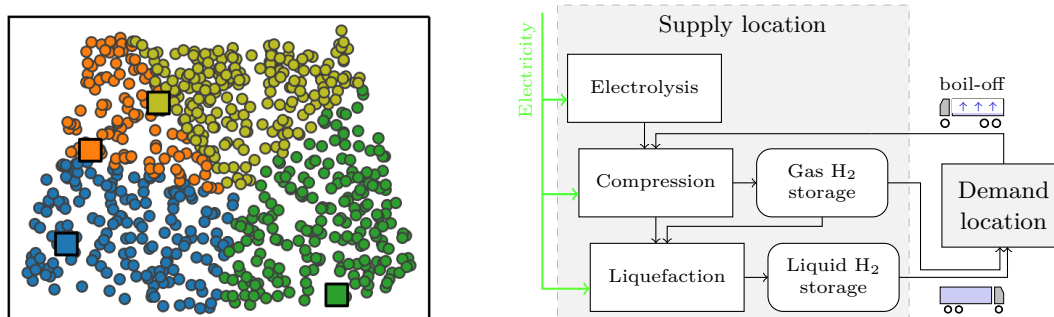
This paper describes the system we have built to support our energy partner in solving this problem, which we refer to as the facility location and operation problem. While already deployed at our partner’s servers, the system is in constant evolution to incorporate the new capabilities our partner requests, while they continue using it. Importantly, the system does not just consist of a problem model that is instantiated with input data and solved using a solver, as this did not satisfy our partner’s needs. Instead, we took advantage of the capabilities of constraint programming modelling languages (MINIZINC [15]) and our knowledge of interactive user interfaces to build a system that supports users in obtaining, exploring and comparing solutions in different ways, tailored to our partner’s needs.

In particular, the system provides users with (a) a *diverse* set of solutions that can be compared visually and numerically, (b) *conflict resolution* methods that, upon infeasibility, identify conflicted constraints and guide users on which to modify, (c) different kinds of *data and model approximations* that allow users to explore near-optimal solutions quickly, (d) the ability to *interactively add/remove/modify constraints* via the interface to explore different scenarios, (e) the ability to impose simple *robustness* constraints that ensure the solution is resilient to unexpected facility outages, and (f) two different models of the same problem that are used to cross-validate the correctness of the solutions. The amount of work required to implement the system was significantly reduced thanks to the use of MINIZINC: its compiler directly supports (a–b), while its high-level nature significantly simplifies (c–f).

Our industry partner has assessed our code against alternative tools and selected ours based both on quantitative and qualitative measures (software design & architecture, user interface, and technology stack), as well as its additional features (diversity and robustness). This provides some measure of its quality and usefulness.

### 2 Facility location and operation problem overview

Intuitively, the facility location and operation problem solved by our Hydrogen Network Optimisation System (or HyNetOS) aims to select the location and configuration of the hydrogen production plants needed to supply the demand of the given demand locations, in such a way as to minimise the total cost of building and operating the resulting production/supply



■ **Figure 1** Left: solution showing 4 plant locations (squares) and the demand locations each of them supplies (circles in the plant’s colour). Right: abstraction of a single plant.

network over a number of years. The left-hand-side of Figure 1 shows a visualisation of a particular solution, where 4 plant locations (the squares) out of all 9 plant locations given as input are selected by HyNetOS to supply all demand locations (the circles), also given as input. The decision of who supplies who is visualised via colour, with demand locations supplied by the same plant sharing the colour of that plant.

As shown in the right-hand-side of Figure 1, each production plant is itself built using one or more units of three production components – electrolyzers, compressors and liquefiers – and two storage systems – for liquid and for compressed gas. The units in which each of these five types of hardware can be delivered, referred to as stock keeping units (SKUs), have different characteristics such as capacity, electricity consumption, and maximum change per hour in production rates, which are given as input. The hydrogen is transported between supply and demand locations using a truck-based transportation network, where the transport costs between each supply-demand location are given via an input table. Note that during the plant sizing, we take into account the impact boil-off has on the final quantity delivered after transport. Boil-off is the decompression due to the boiling of residual hydrogen liquid as a result of increased temperature inside a (nearly) empty truck’s storage tank. Further, we also take into account the location specific price of the available electricity sources necessary to operate each facility, including the generation of a facility operation schedule to minimise electricity costs under variable electricity price and availability.

Importantly, the problem is defined across two timescales: *periods* and *hours*. A period is the number of consecutive years during which the demand for hydrogen is assumed to stay constant. Plants are built during some period and must only increase in size in later periods since, currently, demand is assumed never to decrease. In contrast, the electricity prices for some markets are given hourly for an entire year, yielding up to 8760 possible different prices per market. Because of this, we can potentially perform market arbitrage by adjusting production rates to fit the market price for electricity. However, this means the optimisation needs to make power consumption decisions on an hourly resolution.

The problem’s objective is to minimise the total cost of the network, which can be broken down into the following interdependent cost elements: *building* (CAPEX) and *operating* (OPEX) the hydrogen production plants; *transporting* the product from supply to demand locations; and *powering* the production plants using the available electricity sources. The latter includes a monthly cost (*demand charge*) some electricity providers add to try to flatten their consumer’s load profile, and is proportional to the plant’s highest monthly load.

From an optimisation perspective, this problem is challenging due to the large search space created by the high number of demand locations to be supplied (several hundreds), the different electricity sources often available, and the hourly electricity prices the system must consider across an entire year (up to 8760 per source). This is further complicated by decisions on two different timescales: plant construction decisions are taken for each period, while operating decisions are taken hourly. In addition, the dramatic scale difference between CAPEX (millions) and electricity costs (cents per kWh), makes the model numerically unstable. Together, these factors mean that solving the hydrogen facility model to optimality can quickly become out of reach even for commercial mixed-integer solvers such as GUROBI [7]. Furthermore, due to the costs involved, plant construction decisions necessitate the examination of a range of scenarios, requiring the problem to be solved many times.

### 3 Modeller's view

As HyNet0S is developed for industry and there is no ground-truth, we put special attention on reducing modelling errors. To do this, we separately implemented and integrated two models of the problem whose functional equivalence is continuously checked by ensuring solutions to any model instance can be given to the other without drop in objective or infeasibility. This practice increases the redundancy of the programming task, thereby reducing the residual probability of errors [21]. This section presents the input data used by both models (referred to as **Ori** for the original model, and **Alt** for the alternative one, both always compiled into a MILP problem by MINIZINC), the constraints implemented by one (**Alt**) due to space constraints, and the modelling changes that most improved solving time.

#### 3.1 Input data

Both models require the following input data, which MINIZINC refers to as *parameters*:

- $\mathcal{D}$  and  $\mathcal{S}$ : set of client demand locations and set of candidate supply locations, respectively.
- $\mathcal{K}$ : set of products; currently  $\mathcal{K} = \{\text{LIQ}, \text{GAS}\}$ , i.e., liquid and gas hydrogen, respectively.
- $\mathcal{H}$ : set of hardware; currently  $\mathcal{H} = \{\text{ELEC}, \text{COMP}, \text{LIQF}, \text{GAS\_STR}, \text{LIQ\_STR}\}$  corresponding to electrolyser, compressor, liquifier, gas and liquid storage types, respectively.
- $\mathcal{P}$ : set of constant demand periods and, for each period  $p \in \mathcal{P}$ , how many years  $p_y \in \mathbb{N}$  it covers within the given plan horizon (typically 20 years).
- $\mathcal{SO}_i$ : set of electricity sources (market labels) available at supply location  $s_i \in \mathcal{S}$ . Elements of set  $\mathcal{SO}_i$  correspond to year-long time series of electricity prices, recorded at an hourly resolution. Markets are one of three types: (a) **UTILITY**, a conventional metered connection with mostly fixed prices, (b) **PPA**, a fixed-price power purchase agreement with a renewable energy provider (e.g., solar or wind farm), and (c) **WHOLESALE**, a market with generally the lowest price but exposed to volatile price fluctuations. We distinguish them for their unique features; demand charges typically only apply to **UTILITY**, while **PPA** is a zero carbon emissions source, necessary to achieve carbon targets in the future.
- $\mathcal{T}_i \subseteq \{t_1, t_2, \dots, t_\tau\}$ : set of time steps considered for a single representative year at supply location  $s_i \in \mathcal{S}$ . Each time step  $t \in \mathcal{T}_i$  has a duration in hours  $h_{i,t} \in \{1, \dots, 24\}$ , during which the price of electricity is constant. The value of  $\tau$  ranges between 365 and 8760, corresponding to the cases where every time step represents 1 day and 1 hour, respectively. Their sum  $\sum_{t \in \mathcal{T}_i} h_{i,t}$  must always be 8760.
- $D_{p,j}^k \in \mathbb{R}_{\geq 0}$ : daily demand of product  $k \in \mathcal{K}$  from location  $d_j \in \mathcal{D}$  throughout period  $p \in \mathcal{P}$ , in tonnes per day.
- $T_{p,i,j}^k \in \mathbb{R}_{> 0}$ : transport cost in \$ per kg of product  $k$  sent from supply location  $s_i \in \mathcal{S}$  to demand location  $d_j \in \mathcal{D}$  during period  $p \in \mathcal{P}$ .
- $c_{p,i,t}^{\text{so}} \in \mathbb{R}_{> 0}$ : electricity cost of source  $so \in \mathcal{SO}_i$  at supply location  $s_i \in \mathcal{S}$  during time step  $t \in p_y$  of any year of period  $p \in \mathcal{P}$ , in millions of \$ per MW.
- $\hat{c}_{p,i,t}^{\text{so}} \in \mathbb{R}_{\geq 0}$ : demand charge of source  $so \in \mathcal{SO}_i$  at supply location  $s_i \in \mathcal{S}$  during time step  $t \in p_y$  of any year of period  $p \in \mathcal{P}$  in millions of \$ per MW. Common for **UTILITY**.
- $\text{LB}_i^{\text{so}} \in \mathbb{R}_{\geq 0}$ : minimum annual energy usage in MWh required to be allowed to consume energy from source  $so \in \mathcal{SO}_i$  at location  $s_i \in \mathcal{S}$ . In practice, only relevant for **PPA**.

For each hardware type  $h \in \mathcal{H}$ , we also need the set of concrete stock keeping units (SKUs) in which  $h$  can be delivered, and the following hardware-specific properties:

- $N^h$ : number of available SKUs for  $h$ , from which we build index set  $\mathcal{C} = 1..N^c$ .
- $C^c \in \mathbb{R}_{> 0}$ : capacity of SKU  $c \in \mathcal{C}$  in tonnes per day.
- $K_p^c$ : ownership cost in millions of \$ for one SKU  $c \in \mathcal{C}^h$  starting from period  $p \in \mathcal{P}$ .

- $\nu^h \in [0..1]$ : minimum production (turndown) rate proportional to installed capacity for  $h$ .
- $e^h \in \mathbb{R}_{\geq 0}$ : electricity usage in MWh per tonne of production needed to run  $h$ .
- $\rho^h \in [0..1]$ : overhead to production and storage for  $h \in \{\text{COMP}, \text{LIQF}, \text{GAS\_STR}, \text{LIQ\_STR}\}$  required to compensate for boil-off.
- $\mu \in [0..1]$ : maximum ramping rate of the liquefaction units.

### 3.2 Decision variables, constraints and objective function

Figure 2 shows the objective and constraints in model **Alt**, where blue is used for decision variables and black for input data, split into the following six groups.

**Network constraints.** As hydrogen demand is assumed to stay constant throughout any period  $p \in \mathcal{P}$ , we assume the supply network will also stay constant throughout  $p$ . Thus, we only have to decide once per period  $p$  which location  $s_i \in \mathcal{S}$  supplies (part of) the daily demand of location  $d_j \in \mathcal{D}$  for product  $k \in \mathcal{K}$ . The network constraints encode these assumptions. Variable  $x_{p,i,j}^k$  represents the percentage of the demand  $d_j$  of product  $k$  supplied by location  $s_i$  in period  $p$ . Constraint (1) ensures that every product of every demand location is serviced in its entirety by the supply locations. In order to satisfy the demand induced by the network, production plants must be built at one or more of the supply locations. Binary variable  $b_{p,i}^k$  represents whether a plant for product  $k$  is built at location  $s_i$  at (or before) period  $p$ , i.e., whether there is a plant in  $s_i$  producing  $k$  in period  $p$ . Constraint (2) ensures  $b_{p,i}^k = 1$  whenever any  $x_{p,i}^k > 0$ , thereby capturing the need for a plant. Currently, hybrid facilities (producing *both* gas and liquid for offtake) are disallowed via constraint (3); a requirement of our industry partner.

The objective, shown in expression (26), includes as its first component the sum of the total cost of transporting the product across the links selected with non-zero weight  $x$ , where transporting the entire demand (of  $D_{p,j}^k$  tonnes per day) of product  $k$  from  $s_i \rightarrow d_j$  costs  $T_{p,i,j}^k$  in period  $p$ .

**Offtake constraints.** During a period  $p \in \mathcal{P}$ , the daily demand  $D_{p,j}^k$  of product  $k \in \mathcal{K}$  at every location  $d_j \in \mathcal{D}$  is constant. Thus, we denote as  $z_{p,i}^k$  the *offtake rate* (in tonnes per day) of any supply location  $s_i \in \mathcal{S}$  for period  $p$  and product  $k$ , i.e., the rate at which the plant at location  $s_i$  must produce  $k$  over the long term, using storage to buffer production rate changes in the short term. Constraints (4) and (5) define  $z_{p,i}^k$  for  $k = \text{LIQ}$  and  $k = \text{GAS}$ , respectively. Note that the offtake of gas  $z_{p,i}^{\text{GAS}}$  includes the material needed to produce liquid *from* that gas through liquefaction, and not just the transport requirements of gas.

**Plant capacity constraints.** Producing  $z_{p,i}^k$  of product  $k \in \mathcal{K}$  requires a production plant at location  $s_i \in \mathcal{S}$  during period  $p \in \mathcal{P}$  that is of suitable capacity. This can be modelled using the following five constraints. First, let variable  $u_{p,i}^c$  represent how many of SKU  $c \in \mathcal{C}^h$  of hardware type  $h \in \mathcal{H}$  are *newly* installed at the start of period  $p$  (and thus available for use in  $p$ ) in supply location  $s_i$ . The maximum (or peak) production capacity of the plant at location  $s_i$  in period  $p$  is defined by constraint (6) as the sum of the SKUs capacities installed *up to*  $p$ , and denoted by variable  $y_{p,i}^h$ . By tying the capacity to the cumulative number of newly installed units, we ensure plants can only grow in size.

Second, the plant must be big enough for its peak capacity to meet the offtake rate  $z_{p,i}^k$ . This is ensured by constraint (7), which adds the overhead factor  $\rho^h$  to the normal offtake to take transportation boil-off into account. The boil-off factor is applied differently to different

Network	$\sum_{s_i \in \mathcal{S}} x_{p,i,j}^k = 1 \quad \forall p, d_j, k \text{ where } D_{p,j}^k > 0 \quad (1)$	(1)
	$x_{p,i,j}^k \leq b_{p,i}^k \quad \forall p, s_i, d_j, k \quad (2)$	(2)
	$\sum_{k \in \mathcal{K}} b_{p,i}^k \leq 1 \quad \forall p, s_i \quad (3)$	(3)
Offtake	$\sum_{d_j \in \mathcal{D}} D_{p,j}^{\text{LIQ}} x_{p,i,j}^{\text{LIQ}} = z_{p,i}^{\text{LIQ}} \quad \forall p, s_i \quad (4)$	(4)
	$z_{p,i}^{\text{LIQ}} + \sum_{d_j \in \mathcal{D}} D_{p,j}^{\text{GAS}} x_{p,i,j}^{\text{GAS}} = z_{p,i}^{\text{GAS}} \quad \forall p, s_i \quad (5)$	(5)
Plant capacity	$\sum_{\substack{1 \leq p' \leq p, \\ c \in \mathcal{C}^h}} C^c u_{p',i}^c = y_{p,i}^h \quad \forall p, s_i, h \quad (6)$	(6)
	$(1 + \rho^h) z_{p,i}^{h \rightarrow k} \leq y_{p,i}^h \quad \forall p, s_i, h \in \{\text{ELEC}, \text{COMP}, \text{LIQF}\} \quad (7)$	(7)
	$z_{p,i}^{h \rightarrow k} \geq \nu^h y_{p,i}^h \quad \forall p, s_i, h \in \{\text{ELEC}, \text{COMP}, \text{LIQF}\} \quad (8)$	(8)
	$\rho^{h \rightarrow k} z_{p,i}^{h \rightarrow k} + m^{h \rightarrow k} b_{p,i}^{h \rightarrow k} \leq y_{p,i}^h \quad \forall p, s_i, h \in \{\text{GAS\_STR}, \text{LIQ\_STR}\} \quad (9)$	(9)
	$m^{\text{LIQ}} z_{p,i}^{\text{LIQ}} \leq y_{p,i}^{\text{LIQ\_STR}} \quad \forall p, s_i \quad (10)$	(10)
Production	$\nu^h y_{p,i}^{h \rightarrow k} \leq \hat{p}_{p,i}^{h \rightarrow k} \quad \forall p, s_i, h \in \{\text{ELEC}, \text{COMP}, \text{LIQF}\} \quad (11)$	(11)
	$\hat{p}_{p,i}^{h \rightarrow k} + p_{p,i,t}^{h \rightarrow k} + \rho^h z_{p,i}^{h \rightarrow k} \leq y_{p,i}^h \quad \forall p, s_i, t, h \in \{\text{ELEC}, \text{COMP}, \text{LIQF}\} \quad (12)$	(12)
	$p_{p,i,t+1}^{\text{LIQ}} - p_{p,i,t}^{\text{LIQ}} \leq \mu h_{i,t} y_{p,i}^{\text{LIQF}} \quad \forall p, s_i, t \quad (13)$	(13)
	$p_{p,i,t}^{\text{LIQ}} - p_{p,i,t+1}^{\text{LIQ}} \leq \mu h_{i,t} y_{p,i}^{\text{LIQF}} \quad \forall p, s_i, t \quad (14)$	(14)
	$p_{p,i,0}^{\text{LIQ}} - p_{p,i, \mathcal{T}_i }^{\text{LIQ}} \leq \mu h_{i,t} y_{p,i}^{\text{LIQF}} \quad \forall p, s_i, t \quad (15)$	(15)
	$p_{p,i, \mathcal{T}_i }^{\text{LIQ}} - p_{p,i,0}^{\text{LIQ}} \leq \mu h_{i,t} y_{p,i}^{\text{LIQF}} \quad \forall p, s_i, t \quad (16)$	(16)
Storage	$s_{p,i,t}^{h \rightarrow k} \leq y_{p,i}^h \quad \forall p, s_i, t, h \in \{\text{GAS\_STR}, \text{LIQ\_STR}\} \quad (17)$	(17)
	$s_{p,i,t+1}^{\text{GAS}} = s_{p,i,t}^{\text{GAS}} + \frac{h_{i,t}}{24} \left( \hat{p}_{p,i}^{\text{GAS}} + p_{p,i,t}^{\text{GAS}} - \hat{p}_{p,i}^{\text{LIQ}} - \frac{p_{p,i,t}^{\text{LIQ}} + p_{p,i,t+1}^{\text{LIQ}}}{2} - z_{p,i}^{\text{GAS}} + z_{p,i}^{\text{LIQ}} \right) \quad \forall p, i, t \quad (18)$	(18)
	$s_{p,i,t+1}^{\text{LIQ}} = s_{p,i,t}^{\text{LIQ}} + \frac{h_{i,t}}{24} \left( \hat{p}_{p,i}^{\text{LIQ}} + \frac{p_{p,i,t}^{\text{LIQ}} + p_{p,i,t+1}^{\text{LIQ}}}{2} - z_{p,i}^{\text{LIQ}} \right) \quad \forall p, i, t \quad (19)$	(19)
	$s_{p,i,1}^k = s_{p,i, \mathcal{T}_i }^k = 0 \quad \forall p, s_i, k \quad (20)$	(20)
Electricity usage	$24e_{p,i,t} = e^{\text{ELEC}} \left( \hat{p}_{p,i}^{\text{GAS}} + p_{p,i,t}^{\text{GAS}} \right) + e^{\text{COMP}} \left( \hat{p}_{p,i}^{\text{GAS}} + p_{p,i,t}^{\text{GAS}} + \rho^{\text{GAS}} z_{p,i}^{\text{GAS}} \right) + e^{\text{LIQF}} \left( \hat{p}_{p,i}^{\text{LIQ}} + \frac{p_{p,i,t}^{\text{LIQ}} + p_{p,i,t+1}^{\text{LIQ}}}{2} + \rho^{\text{LIQ}} z_{p,i}^{\text{LIQ}} \right) \quad \forall p, i, t \quad (21)$	(21)
	$\sum_{\text{so} \in \mathcal{SO}} e_{p,i,t}^{\text{so}} = e_{p,i,t} \quad \forall p, s_i, t \quad (22)$	(22)
	$\hat{c}_{p,i,t}^{\text{so}} e_{p,i,t}^{\text{so}} \leq \hat{c}_{p,i,m}^{\text{so}}[t] \quad \forall p, s_i, t, \text{so} \quad (23)$	(23)
	$\left( \sum_{t \in \mathcal{T}_i} (h_{i,t} e_{p,i,t}^{\text{so}}) > 0 \right) \implies \left( \sum_{t \in \mathcal{T}_i} (h_{i,t} e_{p,i,t}^{\text{so}}) \geq \text{LB}_i^{\text{so}} \right) \quad \forall p, s_i, \text{so} \quad (24)$	(24)
	$u_{p,i}^c \in \mathbb{N}_0; b_{p,i}^k \in \{0, 1\}; 0 \leq x_{p,i,j}^k \leq 1; y, z, p, s, e \in \mathbb{R}_{\geq 0} \quad (25)$	(25)
Obj.	$\min \sum_{p,i,j,k} (T_{p,i,j}^k x_{p,i,j}^k) + \sum_{p,i,c} (K_p^c u_{p,i}^c) + \sum_{p,i,t,\text{so}} (c_{p,i,t}^{\text{so}} e_{p,i,t}^{\text{so}}) + \sum_{p,i,m,\text{so}} (\hat{c}_{p,i,m}^{\text{so}}) \quad (26)$	(26)

■ **Figure 2** Objective and constraints for the A1t model of the HyNetOS decision system.

types of hardware  $h \in \mathcal{H}$  because boil-off comes in uncompressed gas form. Thus, it has to pass through the compressor and the liquefactor but not the electrolyser. We use  $h \rightarrow k$  to indicate a mapping from hardware to product. For example, since the electrolyser  $h = \text{ELEC}$  produces  $k = \text{GAS}$ , the electrolyser must be scaled to meet gas demand  $z_{p,i}^{h \rightarrow k}$ . Third, the plant should not be so big that running at minimum capacity yields more product on average than is demanded. Constraint (8) captures this requirement, where the minimum production rate depends on the minimum turndown capabilities  $\nu^h$  of the hardware. Finally, we impose two kinds of minimum storage size constraints. The first constraint ensures two things: 1) that every plant has at least some gas storage for buffering boil-off during transport, which is captured as a fraction  $\rho^{h \rightarrow k}$  of daily production rate  $z_{p,i}^k$ , and 2) that every plant meets a global minimum storage amount  $m^{h \rightarrow k}$  if it is built ( $b_{p,i}^{h \rightarrow k}$ ) (constraint (9)).

The second constraint ensures enough liquid storage is built to sustain liquid hydrogen offtake for a minimum number of days  $m^{\text{LIQ}}$  (constraint (10)).

The construction of any SKU  $c \in \mathcal{C}^h$  of hardware  $h \in \mathcal{H}$  incurs CAPEX and OPEX costs. These are aggregated into the period-specific component cost  $K_p^c$  (in millions of \$ per unit) and accumulated via the number of SKUs newly installed in  $p$  (given by  $u_{p,i}^c$ ), yielding the objective term  $K_p^c u_{p,i}^c$ . The sum of these objective terms forms the second component of the objective, shown in expression (26).

**Production constraints.** For the plant operation, we model the average operating costs through a representative year per period. This means we create a production schedule for one year in each (constant demand) period, which then repeats for however many years the period is long. The production rate of hardware  $h \in \mathcal{H}$  for product  $k \in \mathcal{K}$  at supply location  $s_i \in \mathcal{S}$  during period  $p \in \mathcal{P}$  is allowed to change during each of the time steps  $t \in \mathcal{T}_i$  defined for that location. The production rate is measured in tonnes per day and modelled via two terms:  $\hat{p}_{p,i}^{h \rightarrow k} + p_{p,i,t}^{h \rightarrow k}$ . Here,  $\hat{p}^{h \rightarrow k}$  is the constant *baseline* production amount, and  $p_{p,i,t}^{h \rightarrow k}$  the component that is variable in time step  $t$ . Constraint (11) ensures the baseline exceeds the minimum turndown production rate. Constraint (12) ensures the flexible component never exceeds the installed capacity, while also keeping sufficient headroom for the boil-off fraction  $\rho$  that has to be re-compressed and re-liquefied. Constraints (13) and (14) deal with  $h = \text{LIQF}$  having a slow ramping speed. They constrain the ramp up and down rate, respectively, between two consecutive production rates (where the consecutive steps wrap around the year, via constraints (15) and (16)), such that the change in rate does not exceed the ramping capability  $\mu$  of liquefaction (0.1 of total capacity, i.e., 10%) by the number of hours  $h_{i,t}$  for which the ramping is maintained.

**Storage constraints.** Variable production is buffered via storage: overproduction causes the storage level to increase, while underproduction causes the constant offtake to drain the storage. At every time step  $t \in \mathcal{T}_i$  of supply location  $s_i \in \mathcal{S}$  in period  $p \in \mathcal{P}$ , we track the storage level of product  $k \in \mathcal{K}$  in tonnes via variable  $s_{p,i,t}^k$ . Constraint (17) ensures we never store more than the plants' installed storage capacity. Changes in storage levels are captured by constraints (18) for  $k = \text{GAS}$ , and (19) for  $k = \text{LIQ}$ . In both, the storage in  $t+1$  is an offset from the storage in  $t$ , plus hourly production (baseline plus flexible), minus offtake (either by the liquefaction unit, or by the demand locations constant offtake factor). In addition, we anchor the storage to a baseline with a wrap-around constraint in (20). Adding a baseline of 0 at each year's end removes some of the symmetries in storage schedule solutions and ensures the storage level is implementable across period changes.

**Electricity cost constraints.** The last set of constraints connects the daily hydrogen production to the electricity cost required to produce it. To this end, at every time step  $t \in \mathcal{T}_i$  for the plant of supply location  $s_i \in \mathcal{S}$  in period  $p \in \mathcal{P}$ , we represent the power consumption in MW of the plant via variable  $e_{p,i,t}$ . This variable is defined by constraint (21) as the sum of the power consumed by the plant’s hardware  $h \in \{\text{ELEC}, \text{COMP}, \text{LIQF}\}$  to produce the required hydrogen in  $t$  of  $p$ , i.e., as the electricity usage  $e^h$  multiplied by the production rate at that time which, as before, is formed by a baseline  $\hat{p}_{p,i,t}^k$  component, plus a flexible and possibly a boiloff one. Since our production rates are daily, we downscale the power consumption by factor 24 to the per-hour rate. Constraint (22) distributes the power consumption among sources via variables  $e_{p,i,t}^{so}$ , which represent the power consumed from each electricity source  $so \in \mathcal{SO}$  available at that location. These variables are used to derive the cost of this power, which is accumulated in the third sum of the objective. Constraint (23) defines variable  $\hat{e}_{p,i,m[t]}^{so}$  representing the monthly demand charge for the month  $m[t] \in \{1, \dots, 12\}$  where time step  $t$  falls. It is the maximum demand charge (cost  $c_{p,i,t}^{so}$  of each MW consumed  $e_{p,i,t}^{so}$ ) that can be obtained due to the power consumed at each  $t$  of that month. This accounts for the last sum of the objective. Finally, constraint (24) ensures electricity source  $so \in \mathcal{SO}$  is selected only if its minimum annual energy usage  $\text{LB}_i^{so}$  is met.

### 3.3 Effective modelling and solving in practice

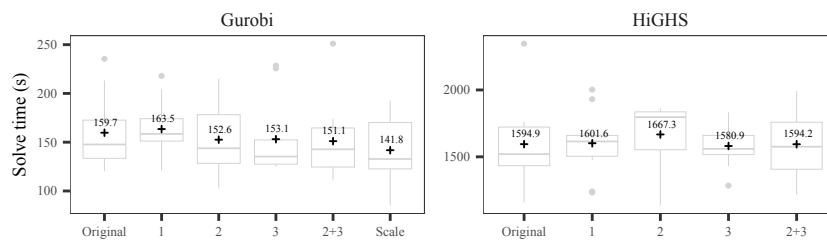
One of the primary benefits of using a *Constraint Modelling Language* such as MINIZINC is that it takes care of the details of mapping models to solvers efficiently (e.g., linearisation). This is important because writing good (i.e. efficiently solvable) models can otherwise be a process of significant trial-and-error, as much an art form as a science. While there are important strategies for good modelling [22], the “last mile” of good practice involves rules of thumb, such as minimizing the number of integer variables in favour of floats, or avoiding equalities if possible. MINIZINC takes care of these during compilation, allowing the modeller to focus on semantic changes. Nevertheless, because of its high level nature, it is equally well-suited to rapid prototyping of model changes. This is why for our industry partner’s quantitative evaluation of the system, we were able to use MINIZINC to try to find model improvements that reduce the total runtime. To do this we took a reference implementation of the model and evaluated the runtime of several model changes. The progression of these changes is shown as a box plot in Figure 3 and discussed below.

**1. Split power equality.** The power equality (21) involves variables  $e_{p,i,t}^{so}$  that are directly minimized in the objective. As such, it is tempting to remove the equality constraint by replacing it with  $\geq$ , such that the model will always demand at least as much power as needed to produce the hydrogen. However, this change allows overconsumption of power to meet the PPA minimum requirement (24). Therefore, we split the power equality constraint into a greater than and a *conditionally applied* less than part, which is only applied if the source has a minimum usage requirement through a conditional constraint:

```
forall (i) if (MIN_USE[I]) then  $\sum_{so \in \mathcal{SO}} e_{p,i,t}^{so} = e_{p,i,t}$  else  $\sum_{so \in \mathcal{SO}} e_{p,i,t}^{so} \geq e_{p,i,t}$  endif;
```

**2. Tighter constraints.** Plant production boolean  $b_{p,i}^k$  is tied to many floating point supply indicator variables. Instead of defining them separately, we can define the state activity indicator with a single constraint, which will be active if any demand is supplied, i.e.:

$$\sum_{d_j \in \mathcal{D}} x_{p,i,j}^k \leq |\mathcal{D}| b_{p,i}^k, \quad \forall p, s_i, k$$



■ **Figure 3** Response of model solve time as a result of model changes; mean time annotated.

**3. Direct objective formulation.** In our reference implementation, the objective terms are captured by intermediate variables for each of the sums that make up the objective. We can rewrite the objective directly in terms of the base variables, allowing the compiler to group terms together, and furthermore helping the MIP solver to prove optimality faster.

We evaluated the time-to-optimality with each model change over 10 runs with different seeds for the solver, to average out effects of solver-internal randomized branching choices and heuristics, and capture statistically meaningful averages on the total runtime. We used a Linux machine with AMD Ryzen 9 3950X 16-Core Processor (3.5-4.7 GHz) and the input data file A shown in Table 1, which has 9 supply locations, 100 demand locations, 3 SKUs per hardware type, and an average period length of 642 tiers (min 365, max 1338). Figure 3 tracks the distributions of total runtime as the changes are implemented, for the Gurobi 9.5.1 and HiGHS 1.5.0 [8] solvers. We also evaluated the combination of 2 and 3 which individually seemed to produce improvements. Nevertheless, the combination of compilation and pre-solving means that most changes have no significant effect. The biggest improvement comes from adjusting how Gurobi scales the objective terms, by changing the SCALEFLAG parameter. We believe this is due to the nature of the hydrogen facility location problem, where many candidate integer assignments are inherently feasible: we can always redistribute the supply network and adjust production rates. The optimisation is thus primarily guided by the (large) objective. We hypothesise that a core-guided search [6], which always assumes constraints can be satisfied and iteratively tightens the model upon detecting an infeasibility, would be a better strategy to solve this kind of model. However, due to the prevalence of floating point data, no core-guided solver available to us can be applied to this problem.

## 4 From modellers to users

To integrate our models into a useful decision system we extended them with functionality that its users (engineers and business experts) could access via the system’s user interface (see Section 5). This includes minor changes to allow many decisions to be set to true/false by the user to explore different scenarios, as well as the more significant ones discussed here.

### 4.1 Run-time versus accuracy – approximations and warm-starts

The large number of variables involved in the hourly operational decisions, can make solving our models to optimality too time consuming for the user. Further, approximate solutions may be sufficient whenever users are interactively exploring what-if scenarios. Our system gives users several ways to control the time versus optimality trade-off, by means of the following two approximation options and warm-start strategy.



**Constraint approximation – annual constant production.** This approximation (referred to as **App**) allows each plant to run at a constant daily production rate sufficient to exactly meet the daily demand that the plant supplies. This means no intra-day or seasonal storage is required and plants can simply build the minimum storage capacity. As a result, in this version of the model all the production and storage constraints (11–20) are removed, and the electricity usage constraints are reformulated to operate on the total annual energy demand under a constant production. As such, the (hourly) energy cost coefficients  $c_{p,i,t}^{\text{so}}$  are rescaled to the total annual cost for a given offtake rate in tonnes per day:

$$c_{p,i}^{\text{so}} = (e^{\text{E}} + e^{\text{C}} + e^{\text{L}}) \sum_{t \in \mathcal{T}_i} \left( c_{p,i,t}^{\text{so}} \cdot \frac{h_{i,t}}{24} \right)$$

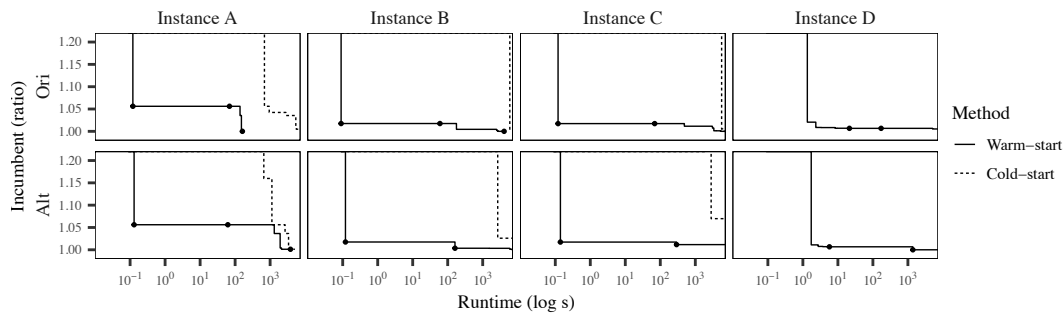
**Data approximation – price-tier grouping.** In practice, many hours in a year often have identical prices for electricity. Since the only driver for operational decisions is a reduction in electricity price, we can group consecutive time steps with equal price. We use the utility market to inform the grouping operation, as utility prices often stay constant for hours, e.g. a daytime tariff and a cheaper night-time tariff from 11pm to 6am. For the other markets in the same compressed time step, we take the median price as the constant price. This approximation (referred to as **Tie**) reduces the cardinality  $\tau$  of the set  $\mathcal{T}_i$ , thereby significantly reducing the total number of floating point variables in the instance (see Table 1).

**Warm-starting strategy.** Both **App** and **Tie** approximations are *valid*, i.e., if their solutions are fed to the non-approximated model (referred to as **Exa** and **Hou**, respectively) they also yield a solution: we can always set the production schedule at a finer time granularity to be equal to the constant-production (rate) assumptions at the coarser level. In addition, they can be combined to further speed up the search at the cost of accuracy. This observation informs our three-stage warm-starting strategy:

1. Solve using an approximate version of the model (i.e., **App-Hou** or **App-Tie**) to optimality;
  2. Assign the integer decisions of that solution to the equivalent variables using an exact version of the model (i.e., **Exa-Hou** or **Exa-Tie**, resp) and resolve;
  3. Warm-start that last used model with the solution from step 2 and resolve to optimality.
- The final step arrives at the same objective as a traditional execution (referred to as *cold-start*) using the same model; however, it will find a good quality initial solution much faster.

■ **Table 1** Size, solving time and objective value of the instances obtained with four data files.

File	Input data size					Instance size for Gurobi solver				Time(s)	Ratio
	S	D	N	M	t/day	Model	Float	Int	Cons.		
A	9	100	9	18	[40]	Exa-Hou	597.7 k	144	835.6 k	5114.85	1.000
						Exa-Tie	96.6 k	144	65.3 k	34.96	1.020
						App-Hou	2.0 k	144	3.4 k	0.22	1.055
						App-Tie	2.0 k	144	3.4 k	0.19	1.068
B	7	100	9	14	[278]	Exa-Hou	457.1 k	112	642.2 k	20982.06	1.000
						Exa-Tie	74.7 k	112	56.5 k	38.06	1.008
						App-Hou	1.6 k	112	2.7 k	0.19	1.017
						App-Tie	1.6 k	112	2.7 k	0.17	1.016
C	9	100	9	18	[278]	Exa-Hou	597.7 k	144	835.6 k	22605.64	1.000
						Exa-Tie	96.6 k	144	65.3 k	84.38	1.009
						App-Hou	2.0 k	144	3.4 k	0.31	1.017
						App-Tie	2.0 k	144	3.4 k	0.31	1.018
D	9	709	9	18	[283, 293]	Exa-Hou	1217.4 k	360	1706.7 k	—	—
						Exa-Tie	215.2 k	360	166.0 k	1309.47	> 1.016
						App-Hou	26.1 k	360	42.2 k	7.81	> 1.027
						App-Tie	26.1 k	360	42.2 k	5.37	> 1.027



■ **Figure 4** Quality of incumbent solution (ratio over optimum) as a function of log runtime obtained by the warm- and cold-start strategies on each instance. Stage solutions are annotated with points.

**Experimental snapshot.** Table 1 shows a snapshot of the trade-offs obtained by the above methods instantiated with four data files (A-D). For each file, it shows the number of supply locations  $|\mathcal{S}|$ ; demand locations  $|\mathcal{D}|$ ; SKUs over all hardware types,  $N = \sum_h N^h$ ; markets  $M = \sum_i |\mathcal{SO}_i|$ , and total demand per period in tonnes/day. It then shows the number (in thousands) of floating-point variables, integer variables, and constraints in the instance resulting from compiling each data file with either an Exa-Hou, Exa-Tie, App-Hou or App-Tie model for the GUROBI solver. The last two columns show the time (where – indicates a timeout) to find an optimal solution and prove optimality when executing each instance in cold-start mode using MINIZINC 2.7.6 and GUROBI 10.0.0.2, and the ratio between the best objective value found using the most accurate Exa-Hou model and the others.

Figure 4 compares the improvement of feasible solutions over time obtained with a cold-start run of the Exa-Hou model instantiated with the data files of Table 1, and with our warm-start strategy, using MINIZINC 2.7.6 and GUROBI 10.0.0.2. Typically, by the time the cold-start finds its first feasible solution (15+ minutes), the warm-start is in its third stage and up to 5% gap to optimality.

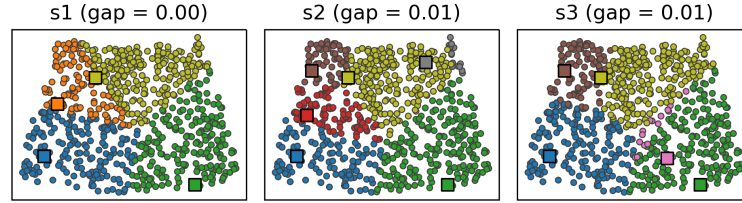
## 4.2 Diversity of solutions

As is common in optimisation, every instance of our problem has many optimal (and even more close-to-optimal) solutions. It is thus useful for decision-makers to obtain a number of close-to-optimal alternative solutions that are meaningfully different in terms of the components being optimised (e.g., transport vs electricity cost) or some major decisions (e.g., plants locations or capacities). Ingmar et al. [11] proposed strategies to find  $N$  diverse solutions of a model instance where the model includes the required user-defined diversity measures. One of these strategies iterates at most  $N - 1$  times from an optimal solution looking for a new solution that is as close to optimality as desired by the user, and is maximally diverse w.r.t. all previously found solutions.

This strategy is now implemented<sup>1</sup> in MINIZINC and used in our system by simply importing from our models a set of diversity measures our users can choose from. The measure most used currently is the Manhattan distance between the vectors of the Boolean variables  $b_{p,i}^k$  indicating there is a plant at supply location  $i \in \mathcal{S}$  in period  $p \in \mathcal{P}$ . Figures 5 and 7(D) show the results of one of the implemented diversity metrics, i.e. maximum diverse set of supply locations. To highlight the differences between diverse solutions our

<sup>1</sup> It will be released as part of the MiniZinc Python package [3] before the end of the year.

system allows users to compare solutions in terms of their supply/demand network, the configuration of the plants they selected, their cost components, etc. Such comparisons have already been useful in identifying, for example, supply locations that appear in all optimal and close-to-optimal solutions due to their favourable cost coefficients (e.g. green location in Figure 5) and are thus prime candidates for construction.



■ **Figure 5** Three maximally diverse solutions within a maximum optimality gap of 0.02, using as distance measure the Manhattan distance of the vector of supply locations built by each solution.

### 4.3 Robustness against plant shutdown

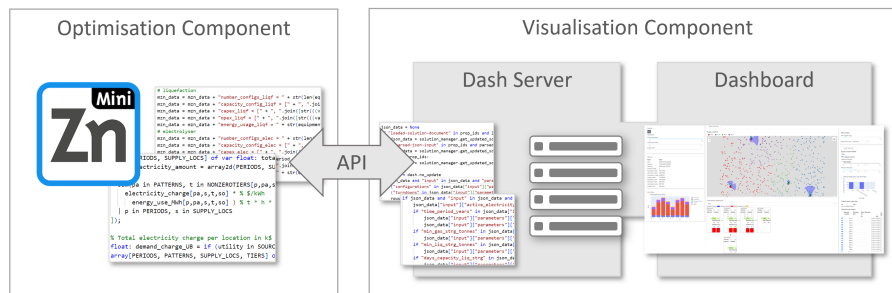
In practice, optimal solutions may lack robustness against uncertain events. There are many sources of uncertainty in our problem, from changes to input data (e.g., costs, demand or hardware technology) to loss of production due to plant shutdowns. Our system is not yet required to deal with the former, as our users are still determining how to build reliable future scenarios and/or probability distributions. It can however be used to find solutions that, in the event of any plant failure, guarantee the supply of a user-defined minimum percentage of the total demand of the network (denoted as new parameter  $GS \in [0..1]$ , set to 0 as default). This allows users to compare optimal but non-robust solutions against robust but non-optimal ones. We achieved this by adding constraint (27) to the models to ensure that for every period  $p \in \mathcal{P}$ , supply location  $s_i \in \mathcal{S}$ , and product  $k \in \mathcal{K}$ , the user-defined percentage  $GS$  of the total demand  $\sum_{d_j \in \mathcal{D}} D_{p,j}^k$  for product  $k$  in period  $p$  is covered by the sum of the peak production capacity of the plants in other supply locations  $s_{i'} \in \mathcal{S} \setminus \{s_i\}$ .

$$GS \sum_{d_j \in \mathcal{D}} D_{p,j}^k \leq \sum_{s_{i'} \in \mathcal{S} \setminus \{s_i\}} y_{p,i'}^h \quad \forall p, s_i, k, h \in \{\text{ELEC, COMP, LIQF}\} \quad (27)$$

## 5 User's view

Our system (shown in Figure 6) connects two main components – optimisation and visualisation – via an application program interface (API). The optimisation calls MINIZINC with the model, input data and configuration to find and return its solutions. The visualisation calls the optimisation via the API and visualises the returned solutions on an interactive web-based dashboard application, referred to as the User Interface (UI). It was implemented using *Plotly Dash* [18], as requested by our industry partner.

Users experience our system mostly via its powerful UI, part of which appears in Figure 7. The UI allows domain-experts and non-expert users (e.g., decision makers) to load different input data, interactively turn some model constraints on/off, obtain solutions, compare them and explore them in detail. This not only helps them make decisions but also gain confidence and trust in the system's output. In addition, domain-experts and external tools can access the system's functionality independently of the UI via the API and the command line interface we implemented in Python. This allows them to automate the solving of instances, i.e. explore many scenarios with different sets of parameters.



■ **Figure 6** The two components integrated by our system and connected by an API.

## 5.1 Configuring an execution run

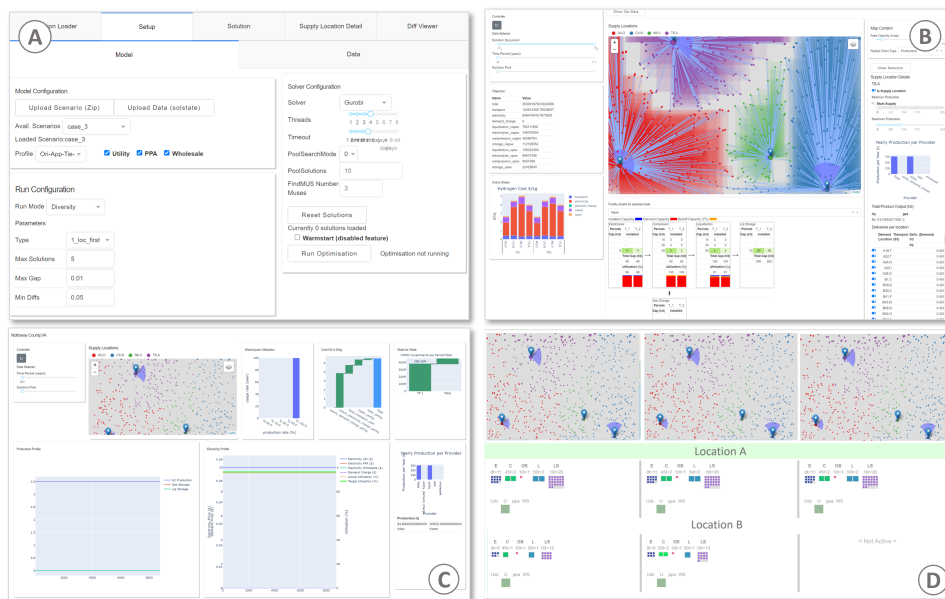
As described above, our system supports a wide range of execution modes (e.g., warm-starts, diversity, and robustness), which users can control via the UI. Fig.7(A) shows one of the panes provided for this. In it, users can select a pre-configured scenario (list of supply and demand locations, their demand, and electricity markets) or upload a new one; modify the scenario by turning on/off the allowed markets (Utility, PPA, Wholesale); select a model profile (e.g., **Ori-Exa-Tie**) from a list of available ones; determine whether to run in diversity, robustness or traditional mode; and modify the default configuration values for each of those. In addition, users can select one of the available solvers; set a timeout and number of threads; and set various solver specific parameters. Once users finish configuring the run, solving can be triggered by pressing a button. If solving time is expected to be long, users can leave the dashboard and load the solution later using the session manager, which keeps track of each execution and reports whether a solution was found, no solution was found due to infeasibility, or the solver is still running.

## 5.2 Visualising and interacting with solutions

The UI displays solutions via the two tabs shown in Figures 7(B) and (C). Figure 7(B) focuses on overview information and the supply/demand network, for which it combines different sub-windows (or *cards*). Figure 7(C) focuses on the operational parts of one plant.

**Overview and Network Tab.** This tab (Figure 7(B)) comprises cards (frames) that contain controller and visualisation elements. With the controller element a specific solution from a set of multiple solutions (for example, solutions from diversity, robustness, or different set of model parameters) and time period can be loaded and visualised. Below the controller a stacked barchart summarises the \$/kg cost (i.e. each cost component) for all active supply locations for all periods. The map in the centre prominently shows the supply/demand network, with markers representing supply locations and coloured circles for each demand location, with the colour matching the supply location they are linked to. For each supply location a radial chart with 4 segments (north, west, south east) shows the amount of hydrogen delivery for each direction, and an area overlay highlights the covered area using semi-transparent rectangles drawn between each linked supply/demand locations, indicating compactness, spread or amount of overlap of supply areas. Supply locations can be selected and its details viewed in a separate card. To the right of the map it shows details about the delivery to each demand location; below it shows the plant configuration represented as a flow diagram with details about installed hardware capacity and number of SKUs, including a stacked bar-chart for each type of hardware showing utilisation to installed capacity. For a

## 21:14 Exploring Hydrogen Supply/Demand Networks



■ **Figure 7** UI's four main tabs: A) input tab, which allows detailed configuration of the model and data before execution; B) supply network as an interactive map with details of selected supply location; C) operational details of each supply location; and D) comparison view of multiple solutions.

selected supply location, interactive elements allow changes to maximum and minimum daily production amount, disabling that particular supply location, or force a non-active location to be active. A re-solve of an instance containing the new data can then be triggered.

**Operation Tab.** For a supply location selected from the map card, this tab (Figure 7(C)) shows further details with focus on the operation side. The two cards in the middle show for each hour of the year (8760 hours) component usage and gas storage levels on the left, and electricity source utilisation (Utility, PPA, Wholesale) on the right. Other cards show the overall hydrolyser utilisation and waterfall charts of the investment costs. On the top left the same controller card is shown to select specific solutions and time periods.

### 5.3 Comparing solutions

Fig 7(D) shows part of the solution comparison pane for three solutions (one per column). For each solution, the pane shows a summary of the configuration of the run that created it, a breakdown of its objective value, and many other useful information, including the two kinds shown in the figure: a map of the plants it built and their hardware as a box-matrix plot (one plant per row). Each box represents one SKU of a particular type of hardware (identified by its colour) and capacity (identified by its size). This enables easy visual comparison of significant changes in plant locations and in their associated hardware. If a location appears in all solutions it is highlighted with a light green colour. Users can use the loading manager to compare any set of solutions obtained, for example, by selecting different model profiles, available sources, and diversity definitions; by modifying the robustness parameter, or by forcing a supply location to produce a given product.

## 5.4 Detecting and resolving conflict

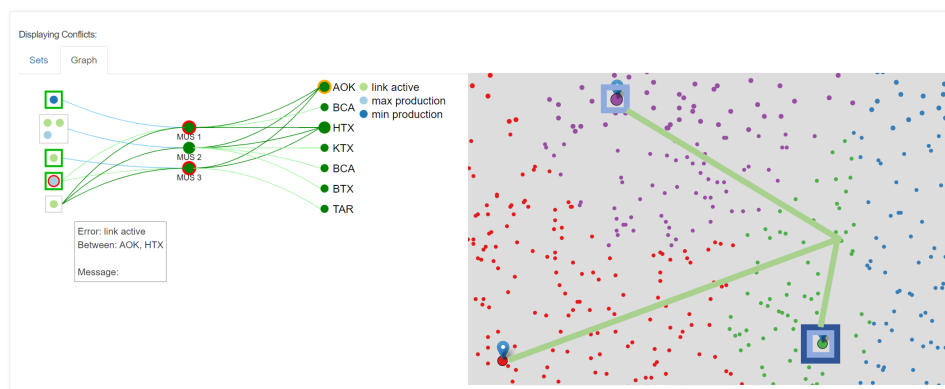
It is not uncommon for users to create infeasible instances while exploring solutions by, for example, setting the minimum daily production for a plant to a higher value than its maximum production. This is a significant roadblock for the usability of any optimisation system. To address it, we use the conflict resolution method of [17], which allows users to obtain a visual representation of the conflicts that is easy to understand, select which of the conflicts to relax for a solution to be found, and obtain a solution to this relaxed instance that quantifies the minimum changes needed to restore feasibility for the original instance.

To implement this method the model needs to be changed to a) provide information about any constraint that can cause conflict, and b) soften these constraints to quantify the minimum changes needed. Both can be achieved in MINIZINC: a) by adding an annotation to each constraint that gives meaningful names to the constraint and its objects, and that tracks its values; b) by adding a slack variable to each constraint that quantifies the minimum changes (see [17] for details). The method also requires computing Minimum Unsatisfiable Sets (MUSes), where a MUS is a set of infeasible constraints in the instance such that removal of any one of the constraints makes the set feasible, and/or Minimum Correction Sets (MCS), where an MCS is a minimum set of constraints that if eliminated from the infeasible instance makes it feasible. Note that any minimum set that intersects all MUSes is an MCS. We have currently implemented one of the pathways proposed in [17] which, upon failure, uses FINDMUS (a MUS enumerator available in MINIZINC) to obtain all MUSes, shows them to the user in different formats, and allows them to manually select an MCS, thus ensuring there is a solution if the constraints in the MCS are relaxed. Currently, users must directly modify these constraints to obtain a feasible instance. We are in the process of implementing the automatic relaxation and resolution process of [17].

Figure 8 shows two of the formats proposed in [17] to show MUSes, which we have implemented. The pane on the left shows the MUS-graph, a compact way of showing the conflicting constraints (left of graph), the MUSes they appear in (centre), and the objects they involve (right). Each circle on the left of the graph represents one conflicting constraint, whose colour connects it to the name shown in the legend on the top right of the graph. For the example shown in the figure these include *minimum production* and *maximum production*. Constraints in the same box appear in the same MUSes. The links connect each group of constraints to the MUSes they appear in, and each MUS to the objects in any of its constraints, which in this case are particular supply and demand locations. If users select a constraint on the left of the graph, the constraint and all the MUSes in which it appears are highlighted with a red frame. Users know they have selected an MCS if the selected constraints highlight all MUSes. A geo-network view of the conflicts is given by showing them on a map (Figure 8 right) of the supply/ demand locations. We overlay conflicts specific to a location with a frame in the colour of the conflict (e.g., light blue to show that this location is involved in a maximum production conflict), and with a coloured link for those involving a supply and demand location. Conflict overlays disappear when any of its constraints is selected in the MUS-graph, thus showing no conflict overlays once an MCS is selected.

## 6 Literature review

The problem of hydrogen production facility location and supply chain optimisation is widely studied; see Riera, Lima and Knio [16] for a recent survey of the field, both in terms of modelling the problem itself, and of optimisation strategies used. Our key takeaway from this survey is that there are myriads of different contexts in which to study the problem, from the



■ **Figure 8** Two conflict visualisations: MUS-graph on the left, Geo-network on the right.

“micro” perspective of a single plant, to the “macro” decisions around the energy delivery system for entire countries (including hydrogen as one of many types of renewable fuels). As such, it is difficult to identify any two papers that study exactly the same mathematical model (e.g., for optimisation benchmarking purposes).

Nevertheless, several papers stand out as closely related to the problem studied here: Ingason, Ingolfsson and Jensson [10] study a electrolysis-based facility location problem for Iceland, although they ignore the operational scheduling of the plant thanks to the assumption of constant-rate renewable sources of electricity (hydro and geothermal). Likewise, Štádlerová et al. [20] study a facility location problem in Norway including uncertainty in the demand and operational considerations such as minimum turn-down rates, although not at an hourly resolution. Demirhan et al. [4] study a multi-fuel, multi-chemical (hydrogen, methanol and ammonia) facility location problem with hourly resolution on the operation of the production plant to capture variability in renewable energy generation from wind and solar. Finally, Kim and Kim [13] also study a joint facility location and hourly operation problem for green hydrogen, although they consider only one time period for facility location without planning the facility’s expansion pathway. All these papers, however, focus mostly on the modelling expert’s view and ignore the domain expert’s one.

## 7 Conclusions and Future Work

This paper describes a Hydrogen Network Optimisation System (HyNetOS) designed to support energy companies in solving the complex combinatorial optimisation problem of producing and supplying hydrogen at minimum cost. HyNetOS integrates two supply network and facility operation models implemented in the high-level constraint modelling language MINIZINC, which natively supplies a range of tools to support the *human* decision-making process, such as finding a diverse sets of near-optimal solutions to present possible alternatives, and conflict resolution technology to explain infeasible instances. Further, HyNetOS supports our industry partner’s decision-making process by means of a powerful and interactive user interface that allows them to view, change, and compare solutions, and rapidly iterate on “what-if” scenarios with efficiently solvable approximate versions of our models. To increase confidence in the quality of the model, we applied a strategy of redundancy and rapid prototyping of model changes to identify the most efficient formulation, and provide robustness measures, to produce resilient solutions against hydrogen production failures.

The HyNetOS system was quantitatively and qualitatively evaluated against an alternative implementation using a direct modelling approach. It was selected as the preferred system based mainly on its significantly higher scores in qualitative criteria such as maintainability, extensibility, user interface, code quality and technology stack, as well as its additional features (diversity and robustness).

While deployed, the system is in constant evolution with many avenues for future work. One of the most pressing and complex is extending the system to produce robust solutions against uncertainties in electricity prices and availability of resources using techniques such as stochastic programming [19], sensitivity analysis [2] and Predict+Optimise [14]. Others include the integration of extra functionality, such as the production of ammonia or the consideration of carbon intensity, the integration of a simulation system for detailed operational modelling on a high resolution time scale, and the integration of a plant layout optimisation system such as [1] to generate optimal hydrogen facility layouts.

---

## References

- 1 Gleb Belov, Tobias Czauderna, Maria Garcia de la Banda, Matthias Klapperstueck, Ilankaikone Senthoooran, Mitch Smith, Michael Wybrow, and Mark Wallace. Process Plant Layout Optimization: Equipment Allocation. In John Hooker, editor, *Principles and Practice of Constraint Programming*, pages 473–489. Springer, 2018. doi:10.1007/978-3-319-98334-9\_31.
- 2 M. W. Dawande and J. N. Hooker. Inference-based sensitivity analysis for mixed integer/linear programming. *Operations Research*, 48(4):505–660, 2000. doi:10.1287/opre.48.4.623.12420.
- 3 Jip J. Dekker. MiniZinc Python, 2023. URL: <https://minizinc-python.readthedocs.io/en/latest/>.
- 4 C. Doga Demirhan, William W. Tso, Joseph B. Powell, and Efstratios N. Pistikopoulos. A multi-scale energy systems engineering approach towards integrated multi-product network optimization. *Applied Energy*, 281:116020, 2021. doi:10.1016/j.apenergy.2020.116020.
- 5 Ibrahim Dincer and Canan Acar. Review and evaluation of hydrogen production methods for better sustainability. *International Journal of Hydrogen Energy*, 40(34):11094–11111, 2015.
- 6 Graeme Gange, Jeremias Berg, Emir Demirović, and Peter J. Stuckey. Core-guided and core-boosted search for CP. In Emmanuel Hebrard and Nysret Musliu, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 205–221, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-58942-4\_14.
- 7 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: <https://www.gurobi.com>.
- 8 Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, March 2018. doi:10.1007/s12532-017-0130-5.
- 9 IEA. Net zero by 2050. Technical report, IEA, Paris, May 2021. URL: <https://www.iea.org/reports/net-zero-by-2050>.
- 10 Helgi Thor Ingason, Hjalti Pall Ingolfsson, and Pall Jensson. Optimizing site selection for hydrogen production in Iceland. *International Journal of Hydrogen Energy*, 33(14):3632–3643, 2008. TMS07: Symposium on Materials in Clean Power Systems. doi:10.1016/j.ijhydene.2008.04.046.
- 11 Linnea Ingmar, Maria Garcia de la Banda, Peter J Stuckey, and Guido Tack. Modelling diversity of solutions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1528–1535, 2020.
- 12 IPCC. Climate change 2022: Impacts, adaptation, and vulnerability. Technical report, Cambridge University Press, Cambridge, UK and New York, NY, USA, 2022. Contribution of Working Group II to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change. doi:10.1017/9781009325844.



- 13 Minsoo Kim and Jiyong Kim. Optimization model for the design and analysis of an integrated renewable hydrogen supply (IRHS) system: Application to Korea's hydrogen economy. *International Journal of Hydrogen Energy*, 41(38):16613–16626, 2016. doi:10.1016/j.ijhydene.2016.07.079.
- 14 Jayanta Mandi, Emir Demirović, Peter J. Stuckey, and Tias Guns. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1603–1610, 2020. doi:10.1609/aaai.v34i02.5521.
- 15 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming—CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23–27, 2007. Proceedings 13*, pages 529–543. Springer, 2007.
- 16 Jefferson A. Riera, Ricardo M. Lima, and Omar M. Knio. A review of hydrogen production and supply chain modeling and optimization. *International Journal of Hydrogen Energy*, 48(37):13731–13755, 2023. doi:10.1016/j.ijhydene.2022.12.242.
- 17 Ilankaikone Senthoran, Matthias Klapperstueck, Gleb Belov, Tobias Czauderna, Kevin Leo, Mark Wallace, Michael Wybrow, and Maria Garcia de la Banda. Human-Centred Feasibility Restoration in Practice. *Constraints*, 2023. (in publication). doi:10.1007/s10601-023-09344-5.
- 18 Shammamah Hossain. Visualization of Bioinformatics Data with Dash Bio. In Chris Calloway, David Lippa, Dillon Niederhut, and David Shupe, editors, *Proceedings of the 18th Python in Science Conference*, pages 126–133, 2019. doi:10.25080/Majora-7ddc1dd1-012.
- 19 Lawrence V. Snyder. Facility location under uncertainty: a review. *IIE Transactions*, 38(7):547–564, 2006. doi:10.1080/07408170500216480.
- 20 Šárka Štádlerová, Trygve Magnus Aglen, Andreas Hofstad, and Peter Schütz. Locating hydrogen production in Norway under uncertainty. In Jesica de Armas, Helena Ramalhinho, and Stefan Voß, editors, *Computational Logistics*, pages 306–321, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-16579-5\_21.
- 21 Mark van den Brand and Jan Friso Groote. Software engineering: Redundancy is key. *Science of Computer Programming*, 97:75–81, 2015. Special Issue on New Ideas and Emerging Results in Understanding Software. doi:10.1016/j.scico.2013.11.020.
- 22 H. Paul Williams. *Model building in mathematical programming*. John Wiley & Sons, 5th edition, 2013.

# Binary Constraint Trees and Structured Decomposability

Petr Kučera  

Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

---

## Abstract

A binary constraint tree (BCT, Wang and Yap 2022) is a normalized binary CSP whose constraint graph is a tree. A BCT constraint is a constraint represented with a BCT where some of the variables may be hidden (i.e. existentially quantified and used only for internal representation). Structured decomposable negation normal forms (SDNNF) were introduced by Pipatsrisawat and Darwiche (2008) as a restriction of decomposable negation normal forms (DNNF). Both DNNFs and SDNNFs were studied in the area of knowledge compilation. In this paper we show that the BCT constraints are polynomially equivalent to SDNNFs. In particular, a BCT constraint can be represented with an SDNNF of polynomial size and, on the other hand, a constraint that can be represented with an SDNNF, can be represented as a BCT constraint of polynomial size. This generalizes the result of Wang and Yap (2022) that shows that a multivalued decision diagram (MDD) can be represented with a BCT. Moreover, our result provides a full characterization of binary constraint trees using a language that is well studied in the area of knowledge compilation. It was shown by Wang and Yap (2023) that a CSP on  $n$  variables of domain sizes bounded by  $d$  that has treewidth  $k$  can be encoded as a BCT on  $O(n)$  variables with domain sizes  $O(d^{k+1})$ . We provide an alternative reduction for the case of binary CSPs. This allows us to compile any binary CSP to an SDNNF of size that is parameterized by  $d$  and  $k$ .

**2012 ACM Subject Classification** Theory of computation → Automated reasoning; Theory of computation → Constraint and logic programming

**Keywords and phrases** Binary CSP, Binary Constraint Tree, Structured Decomposability, Structured DNNF, Polynomial Equivalence

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.22

## 1 Introduction

Constraint satisfaction problems (CSPs) offer an expressive and natural way of formulating problems. A CSP is a problem of checking satisfiability of a conjunction of constraints on variables with finite domains. Constraints can be represented in various ways, which include tables (see e.g. [2, 17, 18]) or multivalued decision diagrams (MDD, see e.g. [1, 5, 6]).

The representation using binary constraint trees was introduced in [27]. A BCT constraint is a constraint  $c$  defined on a set of variables  $\mathbf{x}$  that is represented with a normalized binary CSP  $P$  whose constraint graph is a tree. The CSP  $P$  itself is defined on a set of variables  $\mathbf{z}$  which may include some *hidden* variables in addition to all the *original* variables from  $\mathbf{x}$ . BCTs have a nice property that an arc consistency propagator can be used to check their consistency [12]. Any CSP can be turned into a binary one with an encoding such as dual encoding [11], hidden variable encoding [22], double encoding [24], or bipartite encoding [25].

Decomposable negation normal forms (DNNFs) were introduced in [7] as a tractable language for knowledge representation. Structured DNNFs (SDNNF) were introduced in [20]. The definition of SDNNFs is based on the notion of a *v-tree* which is a rooted binary tree whose leaves are in one-to-one correspondence with the constraint variables (both original and hidden). The conjunction gates in an SDNNF are then required to respect a particular v-tree (see definitions 5 and 6 in Section 2.3 for more details). The structural requirements imposed



© Petr Kučera;

licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 22; pp. 22:1–22:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

on SDNNFs allow for instance a polynomial time construction of an SDNNF representing the conjunction of two SDNNFs that respect the same v-tree – something that is not possible for DNNFs without any structural requirements. Although DNNFs were introduced as a representation of functions on boolean variables, they were also considered as a representation of constraints on variables with finite domains [1, 14].

Both BCT constraints and SDNNFs have a structure based on a tree. We use this similarity to show the main result of this paper: BCT constraints and SDNNF constraints are polynomially equivalent. In particular, we show a polynomial time transformation of a BCT constraint into an SDNNF and also a polynomial time transformation of an SDNNF into a BCT constraint. The polynomial equivalence of BCTs with SDNNFs offers a characterization of BCTs by a language of SDNNFs which has been extensively studied in the area of knowledge compilation [3, 20, 21, 23]. Our result also generalizes the previous construction of a BCT constraint for an MDD described in [27]. It was shown in [26] that BCTs are strictly more succinct than MDDs. This also follows from a combination of our result with the fact that SDNNFs are strictly more succinct than MDDs by [20].

Recently, [28] studied BCTs from the perspective of knowledge compilation together with several other languages that are being used to represent ad-hoc constraints. The authors studied BCTs with respect to the queries and transformations considered in the knowledge compilation map [10] and showed that BCTs allow answering consistency, clausal entailment and model enumeration queries in polynomial time which is (unsurprisingly) the same as in the case of structured DNNFs [20]. The authors of [28] also studied BCTs with respect to transformations. If the input BCTs or SDNNFs are required to have the same tree structure, then they allow polynomial-time bounded conjunction, unbounded disjunction, forgetting any number of variables, and conditioning [20, 28]. Interestingly, [20] only considers the case of combining SDNNFs that respect the same v-tree while [28] also considers the case of combining BCTs that are not required to have the same tree structure. In this case BCTs do not allow polynomial time bounded conjunction, they do not allow an unbounded disjunction, and the case of bounded disjunction is unresolved in [28]. We believe that our result might help to resolve the case of bounded disjunction for BCTs, because it might be easier to reason about a disjunction of two SDNNFs than BCTs. It is also worth mentioning that according to [20], AOMDDs introduced in [19] are strictly less succinct than SDNNFs and thus also strictly less succinct than BCTs. This already answers one of the questions posed in [28].

Our transformation of a BCT constraint into an SDNNF leads to a smooth SDNNF. It is thus possible to use a domain consistency propagator for smooth DNNFs described in [14] as a domain consistency propagator for BCT constraints. An encoding of BCT constraints with propagation complete CNF formulas was described in [26]. Various CNF encodings of DNNF theories were considered in [1] and a propagation complete encoding of smooth DNNFs was introduced in [16]. Our result thus offers an alternative way of reducing BCT constraints to a CNF encoding.

If CNF  $\varphi$  has treewidth  $k$ , then it can be compiled to an SDNNF of size that is parameterized by  $k$  by the construction described in [21]. In particular, if  $\varphi$  has  $n$  variables and  $m$  clauses, then an equivalent SDNNF can be constructed in time  $O(nm2^k)$ . We can obtain a similar result also for binary CSPs, but we have to take into account also the domain sizes. It was shown in [28] that if  $P$  is a CSP on  $n$  variables of domain size  $d$  that has treewidth  $k$ , then it can be encoded as a BCT with  $O(n)$  variables with domain size  $d^{k+1}$ . The construction in [28] uses the encoding described in [11]. In addition, if  $P$  is a binary CSP, its consistency can be checked in time  $O(nd^{k+1})$  by [13]. To have a complete compilation procedure of a binary CSP into an SDNNF, we provide a direct reduction of a binary CSP

to a BCT parameterized by the treewidth and the domain size. The bound we obtain is the same as in [28], so our result is not really new in this sense, but we obtain a slightly better bound on the size of an SDNNF constructed for the given binary CSP than if we would simply combine the bound of [28] with our construction of an SDNNF.

The paper is organized as follows. We introduce the necessary notation in Section 2, including the definitions of BCTs and structured DNNFs. In Section 3, we show that a BCT constraint can be represented with an SDNNF. The transformation of an SDNNF to a BCT is described in Section 4. A transformation of a binary CSP with bounded treewidth into an SDNNF is described in Section 5. Section 6 closes the paper with a few concluding remarks.

## 2 Definitions

In this section, we shall recall the necessary notation and notions used in the paper.

### 2.1 BCT Constraint

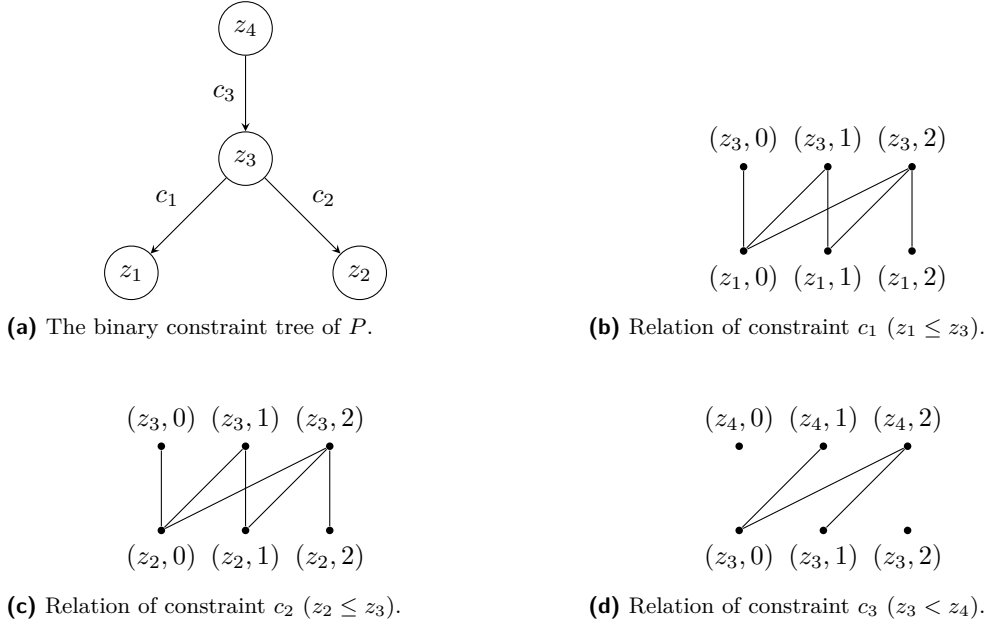
We use a notation adapted from [27] where binary constraint trees were introduced.

A CSP  $P$  is a pair  $(\mathbf{x}, C)$  where  $\mathbf{x}$  is a set of variables and  $C$  is a set of constraints. Each variable  $x$  has a finite domain denoted  $dom(x)$ . A *literal* on a variable  $x$  is a variable value assignment  $(x, a)$ . A *tuple* over a set of variables  $\{x_{i_1}, x_{i_2}, \dots, x_{i_r}\}$  is a *set of literals*  $\{(x_{i_1}, a_1), (x_{i_2}, a_2), \dots, (x_{i_r}, a_r)\}$ . Each constraint  $c_j$  has a constraint scope  $scp(c_j) \subseteq \mathbf{x}$  and a relation  $rel(c_j)$  defined by a set of tuples over  $scp(c_j)$ . A constraint  $c$  is a *binary constraint* if  $|scp(c)| = 2$  and it is a *unary constraint* if  $|scp(c)| = 1$ . A CSP  $P$  is called a *binary CSP* if it consists of binary and unary constraints. A binary CSP is *normalized* if its constraints have pairwise different scopes. Given any set of variables  $\mathbf{z}$  and literals  $\tau$ , we use  $\tau[\mathbf{z}] = \{(x, a) \in \tau \mid x \in \mathbf{z}\}$  to denote a subset of  $\tau$ , while  $T[\mathbf{z}] = \{\tau[\mathbf{z}] \mid \tau \in T\}$  is the *projection* of a set of tuples  $T$  on  $\mathbf{z}$ . A tuple  $\tau$  over  $\mathbf{x}$  is a solution of  $P$  if  $\tau[scp(c)] \in rel(c)$  for all constraints  $c \in C$  and  $a \in dom(x)$  for all  $(x, a) \in \tau$ . We use  $sol(\mathbf{x}, C)$  (or  $sol(P)$ ) to denote the set of all solutions of  $P$ . We also say that  $P$  is *satisfied* by its solution and that a solution of  $P$  *satisfies* all constraints in  $C$ . A *support of a value*  $a \in dom(x)$  in a constraint  $c$  is a tuple  $\tau \in rel(c)$  such that  $(x, a) \in \tau$  and  $b \in dom(y)$  for all  $(y, b) \in \tau$ .

► **Definition 1** ([27]). A Binary Constraint Tree (BCT) is a normalized binary CSP whose constraint graph is a tree. A BCT constraint  $c$  is a pair  $(\mathbf{x}, P)$  such that  $P = (\mathbf{z}, C)$  is a BCT,  $scp(c) = \mathbf{x} \subseteq \mathbf{z}$ , and  $rel(c) = sol(\mathbf{z}, C)[\mathbf{x}]$ . A tree binary encoding (TBE) of a constraint  $c^*$  is a BCT  $P = (\mathbf{z}, C)$  such that the BCT constraint  $(scp(c^*), P)$  has the same constraint relation as  $c^*$  where the variables in  $scp(c^*)$  and  $\mathbf{z} \setminus scp(c^*)$  are called the original and hidden variables, respectively.

► **Example 2.** Let us consider a BCT constraint  $c^* = (\mathbf{x}, P)$  on three variables  $\mathbf{x} = \{x_1, x_2, x_3\}$  where  $P = (\mathbf{z}, C)$  is a BCT described as follows. We have one hidden variable  $y$  in  $P$ , i.e.  $\mathbf{z} = \{x_1, x_2, x_3, y\}$ , and three constraints  $C = \{c_1, c_2, c_3\}$  with  $scp(c_i) = \{x_i, y\}$ ,  $i = 1, 2, 3$ . The domain of all variables (original and hidden) is  $\{1, 2, 3\}$ . For  $i = 1, 2, 3$ , we set  $rel(c_i) = \{((x_i, a), (y, b)) \mid a \neq b\}$ . That is,  $c_i$  enforces that  $y$  has a different value from  $x_i$  in any solution to  $c^*$ . Altogether,  $c^*$  is equal to the negation of the *alldifferent* constraint over the variables  $x_1, x_2, x_3$ .

A general construction of a BCT representing the negation of the *alldifferent* constraint over variables  $x_1, \dots, x_r$  with domains  $D = \{1, \dots, r\}$  was described in [27] where the authors also noted that the size of the MDD representing the constraint is exponential in  $r$ .



■ **Figure 1** A binary constraint tree  $P = (\mathbf{z}, C)$  from Example 3.

To demonstrate the techniques described in the paper, we shall consider a simple constraint that is a bit less symmetrical than the negation of the *alldifferent* constraint on three variables.

► **Example 3.** Figure 1 shows a BCT  $P = (\mathbf{z}, C)$  that is defined on variables  $\mathbf{z} = (z_1, z_2, z_3, z_4)$  with domains  $\text{dom}(z_i) = \{0, 1, 2\}$  for all  $i = 1, \dots, 4$ .  $C$  consists of three constraints. Constraints  $c_1$  and  $c_2$  represent inequalities  $z_1 \leq z_3$  and  $z_2 \leq z_3$  respectively and its relation is shown in figures 1b and 1c. Constraint  $c_3$  represents inequality  $z_3 < z_4$  and its relation is shown in Figure 1d. Note that literals  $(z_4, 0)$  and  $(z_3, 2)$  do not have support in  $c_3$ .

Let us now consider a constraint  $c^*$  with scope  $\text{scp}(c^*) = \{x_1, x_2, x_3\}$  where  $\text{dom}(x_i) = \{0, 1, 2\}$  for  $i = 1, 2, 3$  and the set of tuples  $\text{rel}(c^*)$  represents inequality  $\max(x_1, x_2) < x_3$ . If we identify variables  $z_1, z_2$ , and  $z_4$  with  $x_1, x_2$ , and  $x_3$  respectively, then  $P$  is a tree binary encoding of  $c^*$  in which  $z_1, z_2$ , and  $z_4$  are original variables and  $z_3$  is a hidden variable.

Note that the hidden variable  $z_3$  is not actually needed for the constraint representation. We keep it to demonstrate how a hidden variable can be later forgotten in an SDNNF. We may also observe that literals  $(z_4, 0)$  and  $(z_3, 2)$  do not have a support in constraint  $c_3$ . We shall see later how this situation is dealt with during the construction of an SDNNF representing  $c^*$ .

## 2.2 DNNF

The notion of a DNNF was introduced in [7] as a restriction of NNF. We consider a multivalued variant that was used for instance in [14, 16]. This form is suitable for using DNNFs to represent constraints.

Consider a set of variables  $\mathbf{x} = \{x_1, \dots, x_n\}$  with a finite domain  $\text{dom}(x_i)$  for each  $x_i \in \mathbf{x}$ . A sentence in *negation normal form* (NNF)  $D$  is a rooted DAG with vertices  $V$ , root  $\rho \in V$ , the set of edges  $E$ , and the set of leaves  $L \subseteq V$ . The inner vertices (also called *gates*) are labeled with logical connectives  $\wedge$  or  $\vee$ . Each edge  $(v, u)$  in  $D$  connects an inner vertex  $v$  labeled  $\wedge$  or  $\vee$  with one of its inputs  $u$ . The edge is directed from  $v$  to  $u$ , so the inputs of a

vertex are its successors (or child vertices). The leaves are labeled with literals of variables  $\mathbf{x}$ , i.e. each leaf is labeled with a literal  $(x_i, a)$  on a variable  $x_i \in \mathbf{x}$  and a value  $a \in \text{dom}(x_i)$ . We assume that each literal  $(x_i, a)$  is used as a label of at most one leaf. Some of the literals may be missing in  $D$ , however, we assume that for each  $i = 1, \dots, n$  at least one leaf of  $D$  is labeled with a literal on variable  $x_i$ . For technical reasons, we also allow to label leaves with constants 0 or 1. If a NNF is nonempty, constants can always be propagated and these constants are thus needed only on a NNF without any variables.

If all the constraint variables  $x_i \in \mathbf{x}$  are boolean (i.e.  $\text{dom}(x_i) = \{0, 1\}$ ) and we identify literal  $(x_i, 1)$  with the propositional literal  $x_i$  and literal  $(x_i, 0)$  with the propositional literal  $\neg x_i$ , then we obtain the usual definition of a NNF for representing a boolean function.

Assume that  $c$  is a constraint with the scope  $\text{scp}(c) = \mathbf{x}$  and that  $D$  is a NNF defined on the variables  $\mathbf{x}$ . We say that  $D$  represents constraint  $c$  if for every tuple  $\tau$  over variables  $\mathbf{x}$  we have that  $\tau \in \text{rel}(c)$  if and only if  $D$  evaluates to true on the tuple  $\tau$ . Evaluating  $D$  on  $\tau$  is done in a straightforward manner, we simply set the leaves  $(x_i, a) \in \tau$  to true and the remaining leaves to false, then we use the usual semantic of the circuit  $D$  to get the value on this assignment.

Following [14], we define the decomposability and smoothness properties with respect to constraint variables  $x_1, \dots, x_n$ . For a vertex  $v \in V$ , let us denote  $\text{var}(v) \subseteq \mathbf{x}$  the set of variables in the subcircuit of  $D$  rooted at  $v$ . More precisely, a variable  $x_i \in \mathbf{x}$  belongs to  $\text{var}(v)$  if and only if there is a directed path from  $v$  to a leaf labeled with a literal  $(x_i, a)$  for a value  $a \in \text{dom}(x_i)$ . We have by assumption that  $\text{var}(\rho) = \mathbf{x}$ .

► **Definition 4.** We define the following structural restrictions of NNFs.

- We say that NNF  $D$  is decomposable (DNNF), if for every vertex  $v = v_1 \wedge \dots \wedge v_k$  the sets of variables  $\text{var}(v_1), \dots, \text{var}(v_k)$  are pairwise disjoint.
- We say that DNNF  $D$  is smooth if for every vertex  $v = v_1 \vee \dots \vee v_k$  we have  $\text{var}(v) = \text{var}(v_1) = \dots = \text{var}(v_k)$ .

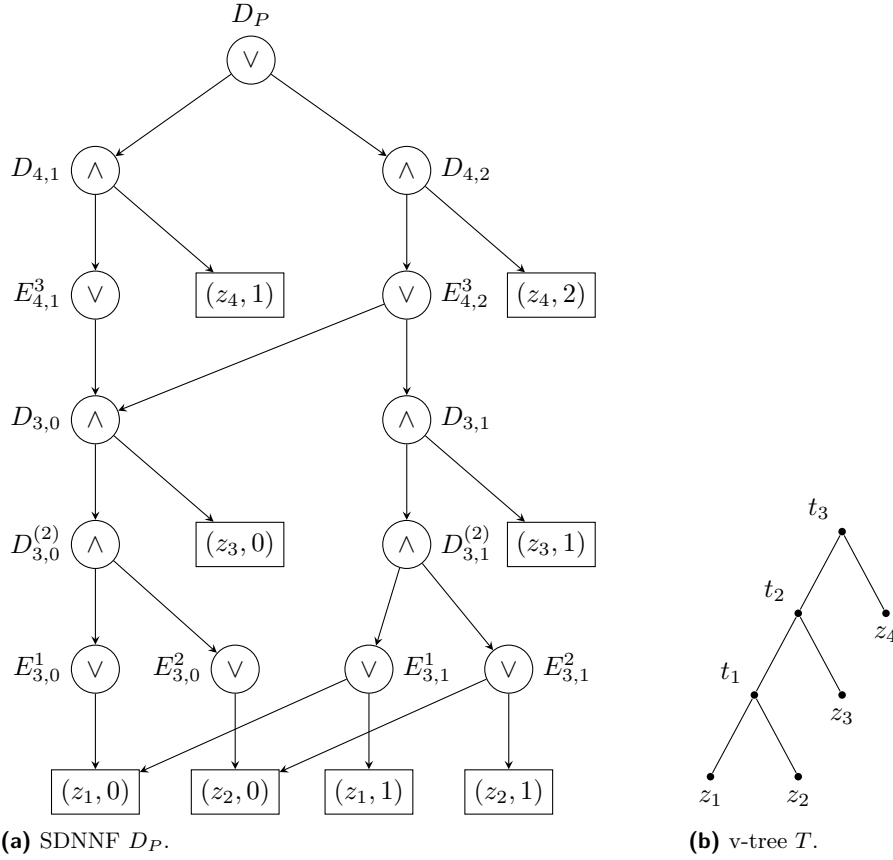
Assume that  $D$  is a DNNF representing constraint  $c$  with scope  $\text{scp}(c) = \mathbf{z}$ . Let  $\mathbf{x} \subseteq \mathbf{z}$  and  $\mathbf{y} = \mathbf{z} \setminus \mathbf{x}$ . By *forgetting* variables  $\mathbf{y}$  in  $D$  we mean the construction of a DNNF  $D'$  that represents the constraint  $c'$  which is a projection of  $c$  on variables  $\mathbf{x}$ . In particular,  $\text{scp}(c') = \mathbf{x}$  and  $\text{rel}(c') = \text{rel}(c)[\mathbf{x}]$ . Forgetting can be done efficiently on a DNNF, we simply replace every literal  $(y, a)$  with constant 1 for all  $y \in \mathbf{y}$  and  $a \in \text{dom}(y)$  [8].

## 2.3 Structured DNNF

Structured DNNFs were introduced in [20]. Structured decomposability is based on the notion of a v-tree defined as follows.

► **Definition 5** ([20]). A v-tree for a set of variables  $\mathbf{x}$  is a full, rooted binary tree whose leaves are in one-to-one correspondence with the variables in  $\mathbf{x}$ .

Given a node  $t$  of a v-tree  $T$ , we denote  $\text{var}(t)$  the set of variables associated with the leaves in the subtree of  $T$  rooted at  $t$ . We also denote  $\text{var}(T) = \text{var}(\sigma)$  where  $\sigma$  is the root of  $T$ . For a non-leaf node  $t$ , we use  $t^l$  ( $t^r$ ) to denote the left (right) child node of  $t$ . For the rest of this paper, we will assume that each conjunction in a DNNF has exactly two non-constant inputs, while a disjunction can have any number of inputs. This is a technical assumption used in [20] mainly to simplify the definition of a SDNNF, since with this assumption, it is enough to consider only binary v-trees. Note also that we can make this assumption without loss of generality due to the associativity and commutativity of conjunction.



■ **Figure 2** An example of an SDNNF  $D_P$  and the corresponding v-tree  $T$ . This particular SDNNF is the result of our construction on the BCT  $P$  from Example 3. The labels of the nodes mark the steps of our construction.

► **Definition 6** ([20]). A DNNF  $D$  respects a v-tree  $T$  if for every conjunction  $v = v_1 \wedge v_2$  in  $D$ , there is a node  $t$  in  $T$ , such that  $\text{var}(v_1) \subseteq \text{var}(t^l)$  and  $\text{var}(v_2) \subseteq \text{var}(t^r)$ .

Let  $v$  be a node in a DNNF  $D$  that respects a v-tree  $T$ . The *decomposition node* ( $d$ -node) of  $v$  is defined as the deepest node  $d$  in  $T$  such that  $\text{var}(v) \subseteq \text{var}(d)$ .

► **Definition 7** ([20]). A DNNF that respects a given v-tree  $T$  is denoted as  $\text{DNNF}_T$ . Moreover, the language of structured DNNFs (SDNNF) consists of all  $\text{DNNF}_T$  for any v-tree  $T$ .

Given a  $\text{DNNF}_T$   $D$ , we can construct an equivalent smooth  $\text{DNNF}_T$   $D'$  in quadratic time [23]. It means that we can always assume that the input  $\text{DNNF}_T$  is smooth.

► **Example 8.** Figure 2 shows an example of a smooth  $\text{DNNF}_T$ . In particular,  $\text{DNNF}_T D_P$  on Figure 2a respects v-tree  $T$  on Figure 2b.  $\text{DNNF}_T D_P$  represents the BCT constraint  $(\mathbf{z}, P)$  where  $P$  is the BCT from Example 3.

For the construction of an SDNNF representing a BCT constraint, we will need the following operation for composing two v-trees. Assume  $T_1$  and  $T_2$  are two v-trees on disjoint sets of variables, i.e.  $\text{var}(T_1) \cap \text{var}(T_2) = \emptyset$ . Then  $T = T_1 \circ T_2$  denotes the v-tree with a newly added root  $\sigma$  whose left child node  $\sigma^l$  is set to the root of  $T_1$  and the right child node  $\sigma^r$  is set to the root of  $T_2$ . It follows that  $\text{var}(T) = \text{var}(T_1) \cup \text{var}(T_2)$ .

### 3 Compiling a BCT Constraint into a Structured DNNF

We shall show in this section that we can construct an SDNNF representing a given BCT constraint in polynomial time.

► **Theorem 9.** *Let  $c^* = (\mathbf{x}, P)$  be a BCT constraint where  $P = (\mathbf{z}, C)$  is a BCT. Then there is a smooth SDNNF  $D$  representing  $c^*$  with  $O(md)$  nodes and  $O(md^2)$  edges where  $m = |\mathbf{z}|$  and  $d = \max_{z_i \in \mathbf{z}} |\text{dom}(z_i)|$ .*

We will describe the construction of  $D$  in the rest of this section, thus proving Theorem 9. Assume that  $n = |\mathbf{x}|$  and  $\mathbf{z} = \{z_1, \dots, z_m\}$  where  $\mathbf{x} \subseteq \mathbf{z}$  and  $m \geq n$  is the number of all variables. We assume that  $|\text{dom}(z_i)| \leq d$  for every  $i = 1, \dots, m$ .

The construction proceeds in two steps. First, we describe a construction of a SDNNF  $D_P$  representing  $P$ . A SDNNF  $D$  that represents  $c^*$  then originates from  $D_P$  by forgetting the hidden variables  $\mathbf{y} = \mathbf{z} \setminus \mathbf{x}$ . This step can be done efficiently by [20].

Let  $G$  be the constraint graph of  $P$ .  $G$  is a tree with the set of nodes  $\mathbf{z}$ , each edge corresponds to a single constraint from  $C$ . Let  $G^+$  denote a directed tree that originates from  $G$  by picking an arbitrary node as a root and directing all edges from the root towards the leaves. Let us assume that the nodes  $z_1, \dots, z_m$  are ordered in a reverse topological order with respect to  $G^+$ . It means that  $z_m$  is the root and if  $(z_i, z_j)$  is an edge in  $G^+$ , then  $i > j$ . See Figure 1a for an example.

For every  $i = 1, \dots, m$ , let us consider the subtree  $G_i$  of  $G^+$  rooted at  $z_i$ . Let  $C_i \subseteq C$  denote the set of constraints corresponding to the edges of  $G_i$ . Denote  $\mathbf{z}_i = \bigcup_{c \in C_i} \text{scp}(c)$ . In this way, we have defined BCT  $P_i = (\mathbf{z}_i, C_i)$ . For every value  $a \in \text{dom}(z_i)$ , we also define BCT  $P_{i,a}$  as a restriction of  $P_i$  to the solutions that contain literal  $(z_i, a)$ . This can be best understood as adding a unary constraint with scope  $z_i$  and a single relation  $(z_i, a)$  to  $C_i$ . Another way of looking at it is restricting the relation of every constraint  $c \in C_i$  with  $z_i \in \text{scp}(c)$  to the tuples containing  $(z_i, a)$  and setting  $\text{dom}(z_i)$  to  $\{a\}$ .

The algorithm proceeds for every  $i = 1, \dots, m$  in order and constructs for every  $a \in \text{dom}(z_i)$  a SDNNF  $D_{i,a}$  representing BCT constraint  $(\mathbf{z}_i, P_{i,a})$  and a v-tree  $T_i$  that is respected by  $D_{i,a}$  using the following steps:

- (A1) If  $z_i$  is a leaf (no edges leave  $z_i$  in  $G^+$ ), then  $P_i$  has no constraints and  $P_{i,a}$  has the domain of  $z_i$  restricted to the single value  $a$ . DNNF  $D_{i,a}$  is a single node labeled with literal  $(z_i, a)$  and v-tree  $T_i$  is a single node labeled with variable  $z_i$ .
- (A2) Assume that  $z_i$  is not a leaf and it has  $k$  outgoing edges  $(z_i, z_{i_1}), \dots, (z_i, z_{i_k})$  associated with constraints  $c_1, \dots, c_k \in C$ . For every  $p = 1, \dots, k$  we have that  $i_p < i$ , because the nodes are processed in a reverse topological order, and thus we have already constructed  $D_{i_p,b}$  and  $T_{i_p}$  for each  $b \in \text{dom}(z_{i_p})$ . Let us now construct  $D_{i,a}$  for  $a \in \text{dom}(z_i)$  in the following two steps.

(A2a) For every  $p = 1, \dots, k$ , define  $E_{i,a}^{i_p}$  as follows:

$$E_{i,a}^{i_p} = \bigvee_{\{(z_i,a),(z_{i_p},b)\} \in \text{rel}(c_p)} D_{i_p,b}.$$



**(A2b)** We construct  $D_{i,j}$  as a conjunction of literal  $(z_i, a)$  with all DNNFs  $E_{i,a}^{i_p}$  for  $p = 1, \dots, k$ . However, since we only allow conjunctions with two inputs, we construct  $D_{i,a}$  in  $k + 1$  steps as follows.

$$D_{i,a}^{(1)} = E_{i,a}^{i_1} \quad (1)$$

$$D_{i,a}^{(p)} = D_{i,a}^{(p-1)} \wedge E_{i,a}^{i_p} \quad \text{for } p = 2, \dots, k \quad (2)$$

$$D_{i,a} = (z_i, a) \wedge D_{i,a}^{(k)} \quad (3)$$

In addition, we define  $T_i$  as follows:

$$T_i^{(1)} = T_{i_1} \quad (4)$$

$$T_i^{(p)} = T_i^{(p-1)} \circ T_{i_p} \quad \text{for } p = 2, \dots, k \quad (5)$$

$$T_i = z_i \circ T_i^{(k)} \quad (6)$$

Variable  $z_i$  is identified with a tree consisting of a single leaf labeled with  $z_i$  in step (6).

Once we have constructed  $D_{m,a}$  for every  $a \in \text{dom}(z_m)$ , we compose them to obtain  $D_P$  that respects v-tree  $T = T_m$  as follows:

$$D_P = \bigvee_{a \in \text{dom}(z_m)} D_{m,a}. \quad (7)$$

Intuitively,  $E_{i,a}^{i_p}$  represents the fact that  $(z_i, a)$  has a support  $((z_i, a), (z_{i_p}, b))$  in  $c_p$ . Moreover, the constraints below  $z_{i_p}$  in  $G^+$  can be satisfied with the value of  $z_{i_p}$  set to  $b$ . SDNNF  $D_{i,a}$  represents the models of all constraints that correspond to the edges in the subtree of  $G^+$  rooted at  $z_i$  and that contain literal  $(z_i, a)$ . If  $(z_i, a)$  does not have a support in  $c_p$  for some  $p = 1, \dots, k$ , then the empty disjunction  $E_{i,a}^{i_p}$  is equal to constant 0 and so is  $D_{i,a}$ . However, it is possible that  $E_{i,a}^{i_p}$  is inconsistent even if  $(z_i, a)$  has a support  $((z_i, a), (z_{i_p}, b))$  in  $c_p$ , but  $D_{i_p,b}$  is inconsistent for every such  $b$ .

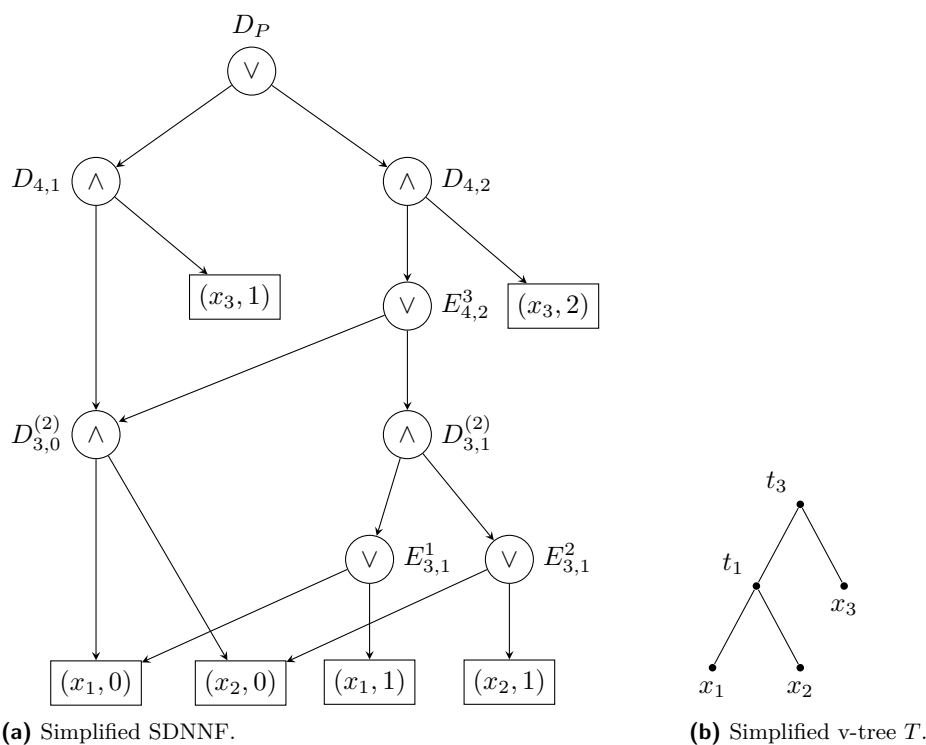
► **Example 10.** Figure 2 shows the result of the construction when applied to the BCT  $P$  from Example 3. Note that there is no leaf labeled with literal  $(z_4, 0)$ , because  $(z_4, 0)$  has no support in constraint  $c_3$ . For the same reason, there is no leaf labeled with literal  $(z_3, 2)$ . Consequently, there are no leaves labeled with literals  $(z_1, 2)$  and  $(z_2, 2)$ . It is worth noting that value 2 would be removed from the domains of variables  $z_1, z_2$ , and  $z_3$  and value 0 would be removed from the domain of  $z_4$  when enforcing arc consistency. In this way, enforcing arc consistency is part of the construction.

If variable  $z_3$  would be forgotten from  $D_P$ , we would obtain a SDNNF representing the constraint  $c^*$  from Example 3. This would amount to replacing leaves labeled with literals  $(z_3, 0)$  and  $(z_3, 1)$  with constant 1. In this case, it just means removing these leaves altogether. Afterwards, we could simplify the SDNNF by removing the trivial gates with a single input, the result of this simplification can be seen in Figure 3.

We will now show that  $D_P$  is a smooth SDNNF of polynomial size that represents  $P$ . We will start by showing that  $D_P$  is a smooth SDNNF.

► **Lemma 11.**  $D_P$  is a smooth SDNNF that respects v-tree  $T_m$ .

**Proof.** Let us first show that  $D_P$  is an SDNNF that respects v-tree  $T_m$ . We will proceed by induction on  $i = 1, \dots, m$ . If  $z_i$  is a leaf of  $G^+$ , then by step (A1),  $D_{i,a}$  consists of a single leaf node for every  $a \in \text{dom}(z_i)$ . It follows that  $D_{i,a}$  respects  $T_i$  which also consists of a single leaf node.



**Figure 3** Simplified SDNNF for the BCT constraint  $c^*$  from Example 3 representing constraint  $\max(x_1, x_2) < x_3$ . The SDNNF originated from the SDNNF in Figure 2a by forgetting  $x_3$ , identifying  $x_1 = z_1, x_2 = z_2, x_3 = z_4$ , and removing gates with a single input.

Let us now assume that  $z_i$  is not a leaf, in particular  $i > 1$ . Let us assume that  $z_i$  has  $k$  outgoing edges to nodes  $z_{i_1}$  to  $z_{i_k}$ . Assume a value  $a \in \text{dom}(z_i)$ . By induction hypothesis, for every  $p = 1, \dots, k$  and every  $b \in \text{dom}(z_{i_p})$  we have that  $D_{i_p,b}$  is a SDNNF respecting v-tree  $T_{i_p}$ . It follows that  $E_{i_p,a}^{i_p}$  constructed in step (A2a) is a SDNNF respecting  $T_{i_p}$ . We have that  $\text{var}(E_{i_p,a}^{i_p}) = \mathbf{z}_{i_p}$ . Since the subtrees rooted at nodes  $z_{i_1}, \dots, z_{i_k}$  are pairwise disjoint, the same is true for sets  $\mathbf{z}_{i_1}, \dots, \mathbf{z}_{i_p}$ . Moreover, variable  $z_i$  is not in any of these sets. Therefore,  $D_{i,a}$  constructed in step (A2b) is a DNNF. The construction of the tree  $T_i$  proceeds in a way similar to the construction of  $D_{i,a}$ , and thus  $D_{i,a}$  respects  $T_i$ . In particular, the node of  $T_i$  introduced in (5) is the d-node of the conjunction (2) for the same value of  $p$ , and the node introduced in (6) is the d-node of the conjunction (3).

$D_P$  is constructed in step (7) as a disjunction of  $D_{m,a}$ ,  $a \in \text{dom}(z_m)$ . As each of these SDNNFs respects  $T_m$ , the same is true for  $D_P$ .

Let us now show the smoothness. Assume that  $D_{i,a}$  is nontrivial, i.e. it is not just a single leaf labeled with 0. We show by induction that then  $D_{i,a}$  is a SDNNF that depends on all variables in  $\mathbf{z}_i$ . This is true for the leaves. If  $z_i$  is not a leaf,  $D_{i,a}$  is constructed in step (A2b). By induction hypothesis used on each  $D_{i_p,b}$  we get that  $E_{i_p,a}^{i_p}$  is a smooth disjunction that depends on all variables in  $\mathbf{z}_{i_p}$ . Thus also  $D_{i,a}$  depends on all variables in  $\mathbf{z}_i = \{z_i\} \cup \bigcup_{p=1}^k \mathbf{z}_{i_p}$ . It follows that also the disjunction introduced in the final step (7) is smooth. ◀

Now, let us estimate the size of  $D_P$ .

► **Lemma 12.** *SDNNF  $D_P$  has  $O(md)$  nodes and  $O(md + s)$  edges where  $s = \sum_{c \in C} |\text{rel}(c)|$ .*

**Proof.** For every  $i = 1, \dots, m$ , we add one disjunction for each value  $a \in \text{dom}(z_i)$  in step (A2a) and  $k$  conjunctions in step (A2b) assuming  $z_i$  is not a leaf. One more disjunction gate is added in the final step (7). Altogether, we thus add  $O(md)$  gates to  $D_P$ . Each conjunction gate has at most two inputs, thus  $O(md)$  edges are leaving the conjunction gates. For every constraint  $c_p$  with scope  $\{z_i, z_{i_p}\}$ , every tuple  $((z_i, a), (z_{i_p}, b))$  adds one edge leaving disjunction gate  $E_{i,a}^{i_p}$ . The total number of edges leaving the disjunction gates added in step (A2a) is thus at most  $s$ . The disjunction gate added in the final step has at most  $|\text{dom}(z_i)| \leq d$  inputs. Altogether, we have  $O(md + s)$  edges in  $D_P$ . ◀

It remains to show that  $D_P$  represents  $P$ .

► **Lemma 13.** *SDNNF  $D_P$  represents  $P$ .*

**Proof.** We shall first show by induction on  $i$  that  $D_{i,a}$  represents  $P_{i,a}$  for every  $i = 1, \dots, m$  and  $a \in \text{dom}(z_i)$ . This is true for leaves (and thus also for  $i = 1$ ), because  $P_{i,a}$  does not have any constraints, it depends only on  $z_i$ , and the domain of  $z_i$  is restricted to the value  $a$ . A single node labeled with literal  $(z_i, a)$  added in step (A1) is thus a correct representation of  $P_{i,a}$  in this case.

Let us now consider a variable  $z_i$  with outgoing edges  $(z_i, z_{i_1}), \dots, (z_i, z_{i_k})$  associated with constraints  $c_1, \dots, c_k \in C$  where  $k \geq 1$ .  $C_i$  is thus a disjoint union of  $C_{i_p}$ ,  $p = 1, \dots, k$  with  $\{c_1, \dots, c_k\}$ . Let us also consider a value  $a \in \text{dom}(z_i)$ . Let us assume by induction hypothesis that each  $D_{i_p,b}$  represents  $P_{i_p,b}$  for every  $b \in \text{dom}(z_{i_p})$ . Recall that the scope of  $P_{i,a}$  is the set  $\mathbf{z}_i$  of variables in the subtree of  $G^+$  rooted at  $z_i$ . Let  $\tau$  be a tuple of variables  $\mathbf{z}_i$  and let us fix some  $a \in \text{dom}(z_i)$ .

Let us first assume that  $\tau \in \text{sol}(P_{i,a})$ . It follows that  $(z_i, a) \in \tau$ . We will show that  $\tau$  satisfies  $D_{i,a}$ . Let us consider literals  $(z_{i_1}, b_1), \dots, (z_{i_k}, b_k) \in \tau$ . For every  $p = 1, \dots, k$  we have that  $\tau$  satisfies  $P_{i_p}$ . It satisfies  $P_{i_p,b_p}$  as well since  $(z_{i_p}, b_p) \in \tau$ . By induction hypothesis, circuit  $D_{i_p,b_p}$  represents  $P_{i_p,b_p}$  and thus it evaluates to true on  $\tau$ . By definition of  $P_{i,a}$ ,  $\tau$  satisfies  $P_i$  and thus also constraint  $c_p$ . It follows that  $\{(z_{i_p}, b_p), (z_i, a)\} \in \text{rel}(c_p)$  and thus, by step (A2a), also  $E_{i,a}^{i_p}$  evaluates to true. Since this holds for every  $p = 1, \dots, k$  and  $(z_i, a) \in \tau$ , we have by step (A2b) that  $D_{i,a}$  evaluates to true on  $\tau$ .

Let us now assume that  $D_{i,a}$  evaluates to true on  $\tau$  and let us show that  $\tau \in \text{sol}(P_{i,a})$ . We have  $(z_i, a) \in \tau$  by (3), it remains to show that  $\tau \in \text{sol}(P_i)$ . Let  $p \in \{1, \dots, k\}$  be arbitrary. We have by (1) to (3) that  $E_{i,a}^{i_p}$  evaluates to true. By (A2a), we have that  $D_{i_p,b_p}$  evaluates to true on  $\tau$  for some  $\{(z_i, a), (z_{i_p}, b_p)\} \in \text{rel}(c_p)$ . Induction hypothesis implies  $\tau[\mathbf{z}_{i_p}] \in \text{sol}(P_{i_p,b_p})$  and thus also  $(z_{i_p}, b_p) \in \tau$ . Together with  $(z_i, a) \in \tau$  we obtain that  $c_p$  is satisfied by  $\tau$ . In addition, all constraints in  $P_{i_p}$  are satisfied by  $\tau$ . Since this holds for every  $p = 1, \dots, k$ , we get that all constraints of  $P_i$  are satisfied and thus  $\tau \in \text{sol}(P_i)$ .

Let us now show that  $D_P$  represents  $P$ . If  $\tau$  is a tuple satisfying  $P$  and  $(z_m, a) \in \tau$ , then  $\tau$  satisfies  $P_{m,a}$ . It follows that  $D_{m,a}$  evaluates to true and that  $D_P$  evaluates to true as well. If, on the other hand,  $D_P$  evaluates to true on  $\tau$ , then  $D_{m,a}$  evaluates to true for some  $a \in \text{dom}(z_m)$ . Thus  $\tau$  is a solution of both  $P_{m,a}$  and  $P = P_m$ . ◀

Theorem 9 now follows from the above propositions.

**Proof of Theorem 9.** The SDNNF  $D$  representing  $c^*$  originates from  $D_P$  by forgetting variables  $\mathbf{y} = \mathbf{z} \setminus \mathbf{x}$ . This step can be performed in polynomial time by [20] by replacing the literals on variables from  $\mathbf{y}$  with constants 1 and then propagating these constants. Note that in  $D_P$ , this is equivalent to removing the corresponding leaves which were added in (3). In particular, this step preserves smoothness which is ensured for  $D_P$  by Lemma 11. The size bound on  $D$  follows from Lemma 12 using the fact that  $|C| \leq m$  and  $|\text{rel}(c)| \leq d^2$  for every  $c \in C$ . ◀

## 4 Compiling an SDNNF to a BCT Constraint

In this section, we shall show the following theorem.

► **Theorem 14.** *Let  $c^*$  be a constraint represented by a smooth SDNNF. Then there is a tree binary encoding of  $c^*$  of polynomial size.*

It is useful to look at a DNNF in terms of *certificates* [4], also called *minimal satisfying subtrees* in [16]. A certificate for a satisfying assignment is simply a minimal satisfied sub-DNNF that contains the output gate. Due to decomposability, the certificates of a DNNF are trees whose leaves are some of the leaves of the DNNF. In addition, no two leaves of a certificate are labeled with a literal of the same variable. Assume  $D$  is a smooth DNNF on variables  $\mathbf{x} = (x_1, \dots, x_n)$  and  $S$  its certificate. We shall also assume that  $D$  has no leaves labeled with constants 0 or 1 (as these can always be propagated). Then for each  $i = 1, \dots, n$ , we have that  $S$  contains exactly one leaf associated with a literal of variable  $x_i$  (see also [16] for more details). The leaves of  $S$  thus determine a tuple  $\tau$  on which  $D$  evaluates to true. We say in this case that the leaves of  $S$  are *associated with the literals in tuple  $\tau$* . The certificates are thus in one-to-one correspondence with the satisfying assignments of  $D$ .

Let us now fix a constraint  $c^*$  with  $\text{scp}(c^*) = \mathbf{x}$ . Let us assume that  $c^*$  is represented by a smooth SDNNF  $D$  that respects a  $\vee$ -tree  $T$  and let  $\rho$  denote the root of  $D$ . In particular,  $\text{var}(T) = \text{var}(\rho) = \mathbf{x}$ . Then the certificates of  $D$  also respect  $T$ . In fact, we will show that if  $S$  is a certificate of  $D$ , then the conjunction gates of  $S$  are in one-to-one correspondence with the inner nodes of  $T$ . This property lies at the basis of our construction of a tree binary encoding  $P = (\mathbf{z}, C)$  of  $c^*$ . The idea is to introduce a hidden variable for each inner node  $t$  of  $T$  with the domain being the  $\wedge$ -gates whose d-node is  $t$ . The constraints make sure that the models of  $P$  are in one-to-one correspondence with the certificates of  $D$ .

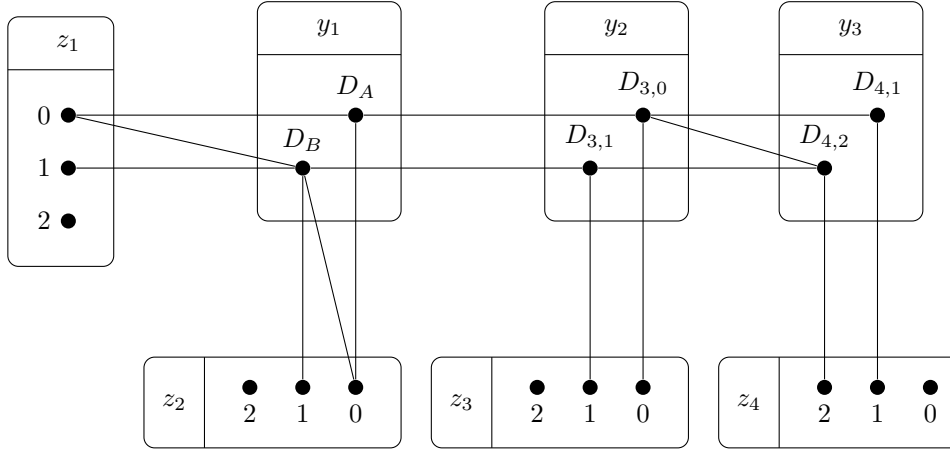
We will construct a BCT  $P = (\mathbf{z}, C)$  satisfying  $\mathbf{x} \subseteq \mathbf{z}$  and  $\text{rel}(c^*) = \text{sol}(P)[\mathbf{x}]$ . For each inner node  $t$  of  $T$ , we introduce a hidden variable  $y_t$ . The set of all these hidden variables will be denoted as  $\mathbf{y}$ . We then define  $\mathbf{z} = \mathbf{x} \cup \mathbf{y}$ . The domain of an original variable  $x_i \in \mathbf{x}$  is  $\text{dom}(x_i)$  as given by the constraint  $c^*$ . For a hidden variable  $y_t \in \mathbf{y}$ , we set  $\text{dom}(y_t) = \Lambda(t)$  where  $\Lambda(t)$  denotes the set of conjunction gates of  $D$  that have d-node  $t$ .

The constraints of  $C$  correspond to the edges of  $T$ . In particular, for each edge  $(t, t')$  of  $T$  where  $t'$  is a child node of  $t$ , we add a constraint  $c_{t,t'}$  to  $C$  whose definition differs slightly depending on whether  $t'$  is a leaf or an inner node of  $T$ . A sequence of vertices  $v_0, \dots, v_k$  of  $D$  is called an  $\vee$ -path if it is a path, nodes  $v_1, \dots, v_{k-1}$  are  $\vee$ -gates,  $v_0$  is a conjunction gate and  $v_k$  is either a conjunction gate, or a leaf node.

- (C1) Assume  $t'$  is a leaf labeled with variable  $x_i$ . Then  $\text{scp}(c_{t,t'}) = \{y_t, x_i\}$  and  $\text{rel}(c_{t,t'})$  is defined as a set of tuples  $\{(y_t, v), (x_i, a)\}$  such that  $D$  contains a  $\vee$ -path from  $v$  to the leaf labeled with literal  $(x_i, a)$ .
- (C2) Assume  $t'$  is an inner node of  $T$ . Then  $\text{scp}(c_{t,t'}) = \{y_t, y_{t'}\}$  and  $\text{rel}(c_{t,t'})$  is defined as a set of tuples  $\{(y_t, v), (y_{t'}, v')\}$  such that  $D$  contains a  $\vee$ -path from  $v$  to  $v'$ .

► **Example 15.** Figure 4 shows the result of the application of the construction to the SDNNF from Figure 2. Recall that the SDNNF itself was constructed as a representation of the BCT  $P$  from Example 3. BCT in Figure 4 differs from  $P$  in that it has three hidden variables  $y_1$ ,  $y_2$ , and  $y_3$ . Note that  $y_1$  and  $y_2$  are basically equivalent to  $z_3$  and  $y_3$  is equivalent to  $z_4$ . The auxiliary variables can thus be easily eliminated by which we obtain the constraints of  $P$ .

## 22:12 Binary Constraint Trees and Structured Decomposability



■ **Figure 4** Example of the construction of the tree binary encoding of a constraint represented by the SDNNF  $D$  in Figure 2.

The size of  $P$  defined in this way is clearly polynomial in the size of  $D$ . The rest of this section is devoted to showing the correctness of the construction. We will start with a few technical propositions on the structure of the certificates of  $D$ .

- **Lemma 16.** *Assume  $v$  is a node of  $D$  with d-node  $t$  in  $T$ . Then  $\text{var}(v) = \text{var}(t)$ . Moreover,*
1. *if  $v = v_1 \wedge v_2$ , then  $t^l$  is the d-node of  $v_1$  and  $t^r$  is the d-node of  $v_2$ , and*
  2. *if  $v = v_1 \vee \dots \vee v_k$ , then  $t$  is the d-node of all input nodes  $v_1, \dots, v_k$ .*

**Proof.** We will proceed by the induction on the structure of  $D$ . If  $v = \rho$  is the root of  $D$ , then  $\text{var}(v) = \mathbf{x}$ . It follows that its d-node  $t$  is the root of  $T$  and thus  $\text{var}(t) = \mathbf{x} = \text{var}(v)$ .

Let us assume that  $v = v_1 \wedge v_2$  and that  $\text{var}(v) = \text{var}(t)$ . Since  $D$  does not contain leaves labeled with constants, we have that both  $\text{var}(v_1)$  and  $\text{var}(v_2)$  are nonempty and thus  $\text{var}(v_i) \subsetneq \text{var}(v)$  for  $i = 1, 2$  and  $\text{var}(v) = \text{var}(v_1) \cup \text{var}(v_2)$ . Let  $t_i$  be the d-node of  $v_i$ ,  $i = 1, 2$ . By the definition of structured DNNFs, both  $t_1$  and  $t_2$  are descendants of  $t$  in  $T$  and thus  $\text{var}(t_i) \subseteq \text{var}(t)$ ,  $i = 1, 2$ . By the definition of d-nodes, we also have that  $\text{var}(v_i) \subseteq \text{var}(t_i)$ ,  $i = 1, 2$ . It follows that  $\text{var}(t) = \text{var}(v) = \text{var}(v_1) \cup \text{var}(v_2) \subseteq \text{var}(t_1) \cup \text{var}(t_2) \subseteq \text{var}(t)$  and thus  $\text{var}(t) = \text{var}(t_1) \cup \text{var}(t_2)$ . The only possibility is that both  $t_1$  and  $t_2$  are the child nodes of  $t$  and thus  $t_1 = t^l$ ,  $t_2 = t^r$ , and  $\text{var}(v_i) = \text{var}(t_i)$ ,  $i = 1, 2$ .

Assume that  $v = v_1 \vee \dots \vee v_k$  and  $\text{var}(v) = \text{var}(t)$ . By smoothness we get that  $\text{var}(t) = \text{var}(v) = \text{var}(v_1) = \dots = \text{var}(v_k)$ . It follows that  $t$  is the d-node of all input nodes  $v_1, \dots, v_k$ .

We have shown that if  $\text{var}(v) = \text{var}(t)$  and  $v'$  is a child node of  $v$  with d-node  $t'$ , then  $\text{var}(v') = \text{var}(t')$  which also holds for the leaves. ◀

- **Lemma 17.** *Assume that  $v_0, \dots, v_k$  is a  $\vee$ -path with  $k > 0$ . Assume that  $t$  is the d-node of  $v_0$  and  $t'$  is the d-node of  $v_k$ . Then  $v_1, \dots, v_{k-1}$  have d-node  $t'$  and  $t'$  is a child node of  $t$ .*

**Proof.** By smoothness,  $\text{var}(v_1) = \text{var}(v_2) = \dots = \text{var}(v_k)$  and thus all gates  $v_1, \dots, v_{k-1}$  have the same d-node as  $v_k$ . In particular,  $t'$  is the d-node of  $v_1$  which is an input to the conjunction gate  $v_0$ . By Lemma 16 we have that  $t'$  is a child node of  $t$ . ◀

Based on Lemma 17, we can show the following proposition.

- **Lemma 18.** *Assume  $S$  is a certificate and  $t$  is an inner node of  $T$ . Then  $S$  contains exactly one conjunction gate  $v$  from  $\Lambda(t)$ .*

**Proof.** Consider a variable  $x_i \in \text{var}(t)$  and the path  $v_0, \dots, v_k$  in  $S$  that leads from the root  $v_0 = \rho$  to the leaf  $v_k$  of  $S$  labeled with a literal on  $x_i$ . It follows that  $\text{var}(v_j) \subseteq \text{var}(v_{j-1})$  for every  $j = 1, \dots, k$ . Moreover  $\text{var}(v_0) = \mathbf{x}$  and  $\text{var}(v_k) = \{x_i\}$ . Let  $v_{i_1}, \dots, v_{i_p}$  be the subsequence of  $v_0, \dots, v_k$  formed only by conjunction nodes. Then by Lemma 17 we get that  $t_{i_1}, \dots, t_{i_p}, t_k$  form a path in  $T$  from the root to the leaf labeled with  $x_i$ . For some index  $i_j$  we thus have that  $t_{i_j} = t$  and it follows that  $v_{i_j}$  is a conjunction gate in  $S$  with d-node  $t$ .

Let us now assume that  $S$  contains two  $\wedge$ -gates  $v_1$  and  $v_2$  with the same d-node  $t$ , thus  $\text{var}(v_1) = \text{var}(v_2) = \text{var}(t)$  by Lemma 16. However, Lemma 16 also implies that there is no path from  $v_1$  to  $v_2$  or from  $v_2$  to  $v_1$ . If we take the paths from the root  $\rho$  to  $v_1$  and  $v_2$  in  $S$ , they have to split in a  $\wedge$ -gate  $v$  (by minimality of  $S$ ), but then  $v$  is not decomposable.

It follows that  $v_{i_j}$  is the only conjunction gate in  $S$  that belongs to  $\Lambda(t)$ . ◀

Note that each literal on a variable from  $\mathbf{z} = \mathbf{x} \cup \mathbf{y}$  is associated with a node of  $D$ . In particular, for  $x_i \in \mathbf{x}$ , literal  $(x_i, a)$  is associated with the leaf of  $D$  labeled with  $(x_i, a)$ . To this end, we need to assume that every such literal has a leaf labeled with it. However, if  $D$  does not contain any leaf associated with literal  $(x_i, a)$ , then this literal does not have a support in  $c^*$  and  $a$  can be removed from  $\text{dom}(x_i)$ . We may thus assume that no such value is in  $\text{dom}(x_i)$ . For an inner node  $t$  of  $T$ , a literal  $(y_t, v)$  is associated with the node  $v \in \Lambda(t)$ .

► **Lemma 19.** *Let  $\tau \in \text{sol}(P)$  be a tuple that is a solution to  $P$ . Then  $\tau[\mathbf{x}] \in \text{rel}(c^*)$ .*

**Proof.** Since  $D$  represents  $c^*$ , it is enough to show that there is a certificate  $S$  of  $D$  whose leaves are associated with the literals in  $\tau[\mathbf{x}]$ .

Tuple  $\tau$  associates a node  $v$  of  $D$  with every node  $t$  of  $T$ . We proceed by induction on the structure of  $T$  to describe a certificate  $S_t$  for the sub-DNNF of  $D$  rooted at  $v$ .

Assume first that  $t$  is a leaf of  $T$  labeled with variable  $x_i$ . Consider the literal  $(x_i, a) \in \tau$  and set the certificate  $S_t$  to a single node labeled with this literal.

Assume now that  $t$  is an inner node of  $T$ . Since  $t$  is an inner node of  $T$ , we have that  $(y_t, v) \in \tau$  for some  $v \in \Lambda(t)$ . Tuple  $\tau$  also contains literals associated with  $t^l$  and  $t^r$ . These literals associate a nodes  $v_l$  and  $v_r$  of  $D$  with  $t^l$  and  $t^r$  respectively. By induction hypothesis, we have constructed certificate  $S_l$  and  $S_r$  for the sub-DNNFs rooted at  $v_l$  and  $v_r$  respectively. Since  $\tau$  satisfies constraints  $c_{t,t^l}$  and  $c_{t,t^r}$ ,  $D$  contains a  $\vee$ -paths from  $v$  to  $v_l$  and from  $v$  to  $v_r$ . Certificate  $S_t$  for the the sub-DNNF rooted at  $v$  is then constructed as a union of  $S_l$ ,  $S_r$ , node  $v$  and the  $\vee$ -paths from  $v$  to  $v_l$  and  $v_r$ .

Let  $\sigma$  be the root of  $T$  and let us assume that  $v$  is the node of  $D$  associated with  $\sigma$  by  $\tau$ . Let  $S_\sigma$  be the certificate of the sub-DNNF rooted at  $v$ . If  $v = \rho$  is the root of  $D$ , then  $S = S_\sigma$  is a certificate of  $D$ . Otherwise,  $D$  contains a path from  $\rho$  to  $v$  that consists only of  $\vee$ -gates and we construct  $S$  by combining this path with  $S_\sigma$ . ◀

► **Lemma 20.** *For every  $\tau^* \in \text{rel}(c^*)$ , there is  $\tau \in \text{sol}(P)$  satisfying  $\tau^* = \tau[\mathbf{x}]$ .*

**Proof.** Since  $\tau^* \in \text{rel}(c^*)$ , there is a certificate  $S$  of  $D$  whose leaves are associated with the literals from  $\tau^*$ . By Lemma 18, the certificate  $S$  contains exactly one conjunction gate  $v_t \in \Lambda(t)$  for each inner node  $t$ . We form  $\tau$  by adding literals  $(y_t, v_t)$  to  $\tau^*$  for all internal nodes  $t$  of  $T$ . Let us check that  $\tau$  satisfies all constraints of  $P$ . Let  $c_{t,t'}$  be a constraint of  $P$  where  $t'$  is a child node of  $t$  in  $T$ . By Lemma 16, one of the child nodes of  $v_t$  in  $D$  has d-node  $t'$ , let us denote it  $v_1$ . Since  $v_t$  is a conjunction gate,  $v_1$  must belong to  $S$ . If  $v_1$  is a disjunction, then by Lemma 16, its child nodes have d-node  $t'$ , too. If we follow the path in  $S$  from  $v_1$  to a leaf or to the next conjunction gate, we get a  $\vee$ -path that ends in the node  $v_k$  whose d-node is still  $t'$  and  $v_k$  is either a leaf or a conjunction gate.

## 22:14 Binary Constraint Trees and Structured Decomposability

If  $t'$  is a leaf of  $T$  labeled with variable  $x_i$ , we must have that  $v_k$  is a leaf of  $S$  labeled with a literal  $(x_i, a)$  for some  $a \in \text{dom}(x_i)$ , it follows that  $(x_i, a) \in \tau^* \subseteq \tau$  and  $\{(y_t, v_t), (x_i, a)\} \in \text{rel}(c_{t,t'})$ , constraint  $c_{t,t'}$  is thus satisfied by  $\tau$ .

If  $t'$  is an inner node, then  $v_k$  is a conjunction gate  $v_{t'}$  associated with  $t'$  in  $S$ . It follows that  $(y_{t'}, v_{t'}) \in \tau$  and  $\{(y_t, v_t), (y_{t'}, v_{t'})\} \in \text{rel}(c_{t,t'})$ , constraint  $c_{t,t'}$  is thus satisfied by  $\tau$ . ◀

Theorem 14 now follows by the above construction from the following proposition.

► **Theorem 21.**  $P = (\mathbf{z}, C)$  is a tree binary encoding of  $c^*$  of polynomial size.

**Proof.**  $C$  consists of  $O(n)$  constraints and the total size of the domains of variables in  $\mathbf{z}$  is bounded by the number of the nodes in  $D$ . Lemmas 19 and 20 imply that  $P$  is a TBE of  $c^*$ . ◀

### 5 Binary Constraint Graphs With Bounded Treewidth

In this section, we shall extend the construction from Section 3 to BCG constraints that naturally generalize BCT constraints.

► **Definition 22.** A BCG constraint  $c$  is a pair  $(\mathbf{x}, P)$  such that  $P = (\mathbf{z}, C)$  is a normalized binary CSP,  $\text{scp}(c) = \mathbf{x} \subseteq \mathbf{z}$  and  $\text{rel}(c) = \text{sol}(\mathbf{z}, C)[\mathbf{x}]$ .

The construction we describe is parameterized by the treewidth of the underlying constraint graph and the domain size. The treewidth of a graph is defined using a tree decomposition.

Given an undirected graph  $G = (V, E)$ , its *tree decomposition* is defined as a pair  $(T, \chi)$  where  $T = (V_T, E_T)$  is a tree and  $\chi : V_T \rightarrow \mathcal{P}(V)$  is a function that assigns each vertex  $t \in V_T$  a subset of  $V$  called a *bag* that satisfies the following conditions:

(d1)  $V = \bigcup_{t \in V_T} \chi(t)$ .

(d2) For each edge  $\{u, v\} \in E$  there is a node  $t \in V_T$  such that  $\{u, v\} \subseteq \chi(t)$ .

(d3) If a node  $v$  is contained in two bags  $\chi(t_1)$  and  $\chi(t_2)$ , then  $v \in \chi(t)$  for every node  $t$  on the path connecting  $t_1$  with  $t_2$ .

The *width* of the tree decomposition is defined as  $\max_{t \in V_T} |\chi(t)| - 1$ . The *treewidth*  $\text{tw}(G)$  of  $G$  is the minimum width among all possible tree decompositions of  $G$ . It should be noted that any tree decomposition of a graph  $G$  on  $n$  vertices can be transformed into a tree decomposition with the same width and  $O(n)$  nodes [15].

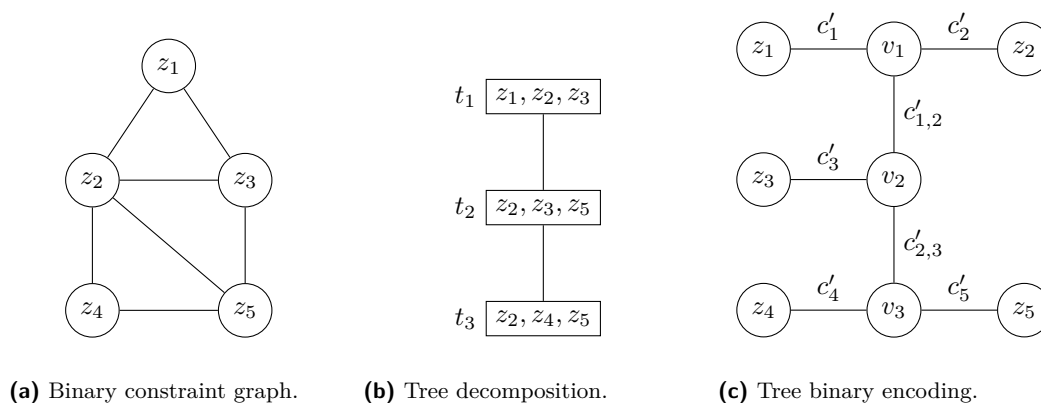
We are now ready to formulate the main result of this section.

► **Theorem 23.** Assume that  $c^* = (\mathbf{x}, P)$  is a BCG constraint defined by a normalized binary CSP  $P = (\mathbf{z}, C)$ . Denote  $G$  the constraint graph of  $P$ . Denote  $m = |\mathbf{z}|$  and  $d = \max_{z_i \in \mathbf{z}} |\text{dom}(z_i)|$ . Then there is an SDNNF  $D$  representing  $c^*$  with  $O(md^{\text{tw}(G)+1})$  nodes and  $O(md^2 \text{tw}(G)+1)$  edges.

The proof of Theorem 23 is based on the following proposition.

► **Theorem 24.** Assume that  $c^* = (\mathbf{x}, P)$  is a BCG constraint defined by a normalized binary CSP  $P = (\mathbf{z}, C)$ . Denote  $G$  the constraint graph of  $P$ . Denote  $m = |\mathbf{z}|$  and  $d = \max_{z_i \in \mathbf{z}} |\text{dom}(z_i)|$ . Then  $c^*$  has a tree binary encoding  $P' = (\mathbf{z}', C')$  with  $|\mathbf{z}'| = O(m)$  and  $|\text{dom}(z'_i)| \leq d^{\text{tw}(G)+1}$  for every  $z'_i \in \mathbf{z}'$ .

Note that Theorem 24 actually follows from Proposition 4 in [28] which is based on the encoding described in [11]. If we would simply combine the bound given by Theorem 24 with the bound given by Theorem 9, we would get an SDNNF  $D$  representing  $c^*$  with



■ **Figure 5** Example of the construction, see the description in Example 25.

$O(md^{tw(G)+1})$  nodes and  $O(md^{2(tw(G)+1)})$  edges. We provide a specific construction that proves Theorem 24 and that can be combined with Lemma 12 to prove a slightly better bound stated in Theorem 23. The construction we describe below is similar to the dual encoding described in [11].

Let us fix a BCG constraint  $c^* = (\mathbf{z}, P)$  where  $P = (\mathbf{z}, C)$  is a normalized binary CSP with constraint graph  $G$ . Let us assume that  $\mathbf{z} = (z_1, \dots, z_m)$  and  $d = \max_{i=1}^m |dom(z_i)|$ . Let us also fix a tree decomposition  $(T, \chi)$  of  $G$ . Let us assume that  $V_T = (t_1, \dots, t_N)$  for some  $N = O(m)$ . We shall describe a BCT  $P' = (\mathbf{z}', C')$  which is a TBE of  $c^*$ . First, let us define the variables in  $\mathbf{z}'$ . We associate a new variable  $v_i$  with every  $t_i$ ,  $i = 1, \dots, N$ . Then we set  $\mathbf{z}' = \mathbf{z} \cup \mathbf{v}$  where  $\mathbf{v} = (v_1, \dots, v_N)$ . The domains of variables in  $\mathbf{z}$  are given by  $c^*$ . For every  $v_i$ ,  $i = 1, \dots, N$ , we set the domain as follows. Let us consider the set of constraints defined on variables from  $\chi(t_i)$  as  $C_i = \{c \in C \mid scp(c) \subseteq \chi(t_i)\}$ . Then the domain of  $v_i$  is defined as the set of solutions to CSP  $(\chi(t_i), C_i)$ , i.e.  $dom(v_i) = sol(\chi(t_i), C_i)$ .

Let us now define the constraints in  $C'$ .

- (T1) For every edge  $\{t_i, t_j\} \in E_T$  we add a constraint  $c'_{i,j}$  into  $C'$  with  $scp(c'_{i,j}) = \{v_i, v_j\}$ . The constraint relation  $rel(c'_{i,j})$  consists of pairs  $\{(v_i, \tau_1), (v_j, \tau_2)\}$  where  $\tau_1 \in dom(v_i)$ ,  $\tau_2 \in dom(v_j)$ , and  $\tau_1[\chi(t_i) \cap \chi(t_j)] = \tau_2[\chi(t_i) \cap \chi(t_j)]$ .
- (T2) For every  $z_i$ ,  $i = 1, \dots, m$ , we pick a representative node  $t_{r_i} \in V_T$  satisfying  $z_i \in \chi(t_{r_i})$ . We then add a constraint  $c'_i$  into  $C'$  with  $scp(c'_i) = \{z_i, t_{r_i}\}$ . The set of tuples  $rel(c'_i)$  consists of pairs  $\{(z_i, a), (v_{r_i}, \tau)\}$  where  $a \in dom(z_i)$ ,  $\tau \in dom(v_{r_i})$ , and  $(z_i, a) \in \tau$ .

► **Example 25.** Let us consider a binary CSP  $P = (\mathbf{z}, C)$  with  $\mathbf{z} = \{z_1, \dots, z_5\}$  whose constraint graph  $G$  is depicted in Figure 5a. We shall use  $c_{i,j} \in C$  to denote the constraint with scope  $\{z_i, z_j\}$ . Figure 5b shows a tree decomposition  $T$  of the graph with the contents of the bags inside the rectangles. The structure of the tree binary encoding  $P'$  of  $P$  is then shown in Figure 5c. The domain of variable  $v_1$  consists of tuples  $\tau$  on variables  $z_1, z_2$ , and  $z_3$  satisfying constraints  $c_{1,2}$ ,  $c_{2,3}$ , and  $c_{1,3}$ . Assume a tuple  $\sigma' \in sol(P')$ . Constraint  $c'_2$  makes sure that if  $(z_2, a) \in \sigma'$ , then  $\sigma'$  contains  $(v_2, \tau)$  satisfying  $(z_2, a) \in \tau$ . Similarly for other variables. Constraints  $c'_{1,2}$  and  $c'_{2,3}$  extend this property also to nodes  $v_2$  and  $v_3$ . The tuples assigned to variables  $v_1, v_2$ , and  $v_3$  are thus consistent with each other and also with constraints  $C$ . We thus have that  $\sigma'[\mathbf{z}] \in sol(P)$ .



The proof of the correctness of the above construction and thus also the proof of Theorem 24 is moved to the appendix. Here, we will describe its application for proving the main result of this section.

**Proof of Theorem 23.** Using Theorem 24, we obtain a TBE encoding  $P' = (\mathbf{z}', C')$  with  $O(m)$  variables and the domain sizes bounded by  $d^{tw(G)+1}$ . We can then apply Theorem 9 to obtain an SDNNF  $D$  that represents  $c^*$ .  $D$  has  $O(md^{tw(G)+1})$  nodes. By Lemma 12, the number of edges of  $D$  is bounded by  $O(md^{tw(G)+1} + s)$  where  $s = \sum_{c' \in C'} |rel(c')|$ . Since  $|C'| = O(m)$ , it is enough to show that  $|rel(c')| \leq d^{2 tw(G)+1}$  for every  $c' \in C'$ .

Assume first a constraint  $c'_{i,j}$  added in step (T1). We may assume that  $G$  is connected (otherwise we process each connected component of  $G$  separately) and therefore  $\chi(t_i) \cap \chi(t_j) \neq \emptyset$ . The number of pairs of tuples  $\tau_1$  and  $\tau_2$  that satisfy  $\tau_1[\chi(t_i) \cap \chi(t_j)] = \tau_2[\chi(t_i) \cap \chi(t_j)]$  is thus at most  $d^{tw(G)+1} \cdot d^{tw(G)} = d^{2 tw(G)+1}$ . Therefore  $|rel(c'_{i,j})| \leq d^{2 tw(G)+1}$ .

Assume now a constraint  $c'_i$  added in step (T2). The number of tuples  $\tau$  satisfying that  $(z_i, a) \in \tau$  for one particular  $a \in dom(z_i)$  is at most  $d^{tw(G)}$  and thus  $|rel(c'_i)| \leq d^{tw(G)+1} \leq d^{2 tw(G)+1}$ . ◀

Note that the size estimate in Theorem 23 is only an upper bound and the real size of  $P'$  and the SDNNF  $D$  depends on the particular tree decomposition and, in particular, on how much the bags intersect. Therefore, there is a space for optimization in a practical setting.

## 6 Conclusion

As the main result of our paper, we have shown that binary constraint trees are polynomially equivalent to structured DNNF circuits. We would like to note that for a given BCT  $P$  the construction in Section 3 leads to a deterministic SDNNF  $D_P$  (thanks to rule 3 in step (A2b)). This means that for every pair of distinct children  $v_1$  and  $v_2$  of a disjunction node, the sub-NNFs rooted at  $v_1$  and  $v_2$  do not share any models (see [9, 20]). This property allows for instance model counting on  $D_P$ . However, forgetting the hidden variables from  $D_P$  does not preserve determinism in general [10] and thus the actual result of the construction is not a deterministic SDNNF. Introducing hidden variables is thus an important part of the construction described in Section 4 since SDNNFs are strictly more succinct than deterministic SDNNFs [20].

Several rules for reducing the number of hidden variables in a BCT constraint were described in [27], it would be interesting to investigate the effect of these rules on a SDNNF that is compiled into a BCT constraint, reduction rules are applied to it and then it is compiled back to a SDNNF. When compiling the BCT constraint back to a SDNNF, we can pick an arbitrary node of the constraint tree as a root which allows us to change the structure of the SDNNF to a different orientation of the v-tree. This, for instance, extends the applicability of the conjoin operation described in [20] to conjoining two SDNNFs whose v-trees differ, but their undirected versions are the same.

---

## References

- 1 Ignasi Abío, Graeme Gange, Valentin Mayer-Eichberger, and Peter J. Stuckey. On CNF encodings of decision diagrams. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 1–17, Cham, 2016. Springer International Publishing.
- 2 Christian Bessiere and Jean-Charles Régin. Arc consistency for general constraint networks: preliminary results. In *IJCAI (1)*, pages 398–404, 1997.

- 3 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. On compiling CNFs into structured deterministic DNNFs. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 199–214, Cham, 2015. Springer International Publishing.
- 4 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Knowledge compilation meets communication complexity. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI'16*, pages 1008–1014. AAAI Press, 2016. URL: <http://dl.acm.org/citation.cfm?id=3060621.3060761>.
- 5 Kenil C. Cheng and Roland H. Yap. Maintaining generalized arc consistency on ad hoc r-ary constraints. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming, CP '08*, pages 509–523, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-85958-1\_34.
- 6 Kenil C. Cheng and Roland H. Yap. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15(2):265–304, April 2010. doi:10.1007/s10601-009-9087-y.
- 7 Adnan Darwiche. Compiling knowledge into decomposable negation normal form. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI'99*, pages 284–289, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. URL: <http://dl.acm.org/citation.cfm?id=1624218.1624260>.
- 8 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, July 2001. doi:10.1145/502090.502091.
- 9 Adnan Darwiche. A compiler for deterministic, decomposable negation normal form. In *Eighteenth National Conference on Artificial Intelligence*, pages 627–634, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. URL: <http://dl.acm.org/citation.cfm?id=777092.777189>.
- 10 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.
- 11 Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, April 1989. doi:10.1016/0004-3702(89)90037-4.
- 12 Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4):755–761, October 1985. doi:10.1145/4221.4225.
- 13 Eugene C. Freuder. Complexity of K-tree structured constraint satisfaction problems. In *Proceedings of the eighth National conference on Artificial intelligence - Volume 1, AAAI'90*, pages 4–9, Boston, Massachusetts, 1990. AAAI Press.
- 14 Graeme Gange and Peter J. Stuckey. Explaining propagators for s-DNNF circuits. In Nicolas Beldiceanu, Narendra Jussien, and Éric Pinson, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 195–210. Springer Berlin Heidelberg, 2012.
- 15 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. doi:10.1007/BFb0045375.
- 16 Petr Kučera and Petr Savický. Propagation complete encodings of smooth DNNF theories. *Constraints*, 27(3):327–359, July 2022. doi:10.1007/s10601-022-09331-2.
- 17 Christophe Lecoutre. STR2: optimized simple tabular reduction for table constraints. *Constraints*, 16(4):341–371, October 2011. doi:10.1007/s10601-011-9107-6.
- 18 Christophe Lecoutre, Chavalit Likitvivatanavong, and Roland H. C. Yap. STR3: A path-optimal filtering algorithm for table constraints. *Artificial Intelligence*, 220:1–27, March 2015. doi:10.1016/j.artint.2014.12.002.
- 19 Robert Mateescu and Rina Dechter. Compiling Constraint Networks into AND/OR Multi-valued Decision Diagrams (AOMDDs). In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006*, Lecture Notes in Computer Science, pages 329–343, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11889205\_25.

- 20 Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1*, AAAI'08, pages 517–522. AAAI Press, 2008. URL: <http://dl.acm.org/citation.cfm?id=1619995.1620079>.
- 21 Knot Pipatsrisawat and Adnan Darwiche. Top-down algorithms for constructing structured DNNF: Theoretical and practical implications. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 3–8, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press. URL: <http://dl.acm.org/citation.cfm?id=1860967.1860970>.
- 22 Francesca Rossi, Charles J. Petrie, and Vasant Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence*, ECAI'90, pages 550–556, USA, 1990. Pitman Publishing, Inc.
- 23 Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: <https://proceedings.neurips.cc/paper/2019/file/940392f5f32a7ade1cc201767cf83e31-Paper.pdf>.
- 24 Kostas Stergiou and Toby Walsh. Encodings of non-binary constraint satisfaction problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 163–168, USA, 1999. American Association for Artificial Intelligence.
- 25 Ruiwei Wang and Roland H. C. Yap. Bipartite encoding: A new binary encoding for solving non-binary CSPs. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20, Yokohama, Yokohama, Japan, 2021.
- 26 Ruiwei Wang and Roland H. C. Yap. CNF Encodings of Binary Constraint Trees. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CP.2022.40.
- 27 Ruiwei Wang and Roland H.C. Yap. Encoding multi-valued decision diagram constraints as binary constraint trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):3850–3858, June 2022. doi:10.1609/aaai.v36i4.20300.
- 28 Ruiwei Wang and Roland H.C. Yap. The expressive power of ad-hoc constraints for modelling CSPs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4104–4114, June 2023. doi:10.1609/aaai.v37i4.25526.

## A Proof of Theorem 24

In this section, we shall prove the correctness of the construction described in Section 5. We shall also prove Theorem 24 that states the properties of the construction. We use the same notation that was used in Section 5. In particular, we assume a fixed BCG constraint  $c^* = (\mathbf{x}, P)$  where  $P = (\mathbf{z}, C)$  is a normalized binary CSP with constraint graph  $G$ . We assume that that BCT  $P' = (\mathbf{z}', C')$  is the result of the construction from Section 5. We shall show that  $P'$  is a TBE of  $c^*$ . The construction of  $C'$  implies the following property.

► **Lemma 26.** *Let  $p \in \{1, \dots, m\}$  be arbitrary and let  $t_i \in V_T$  be such that  $z_p \in \chi(t)$ . Assume that  $\sigma' \in \text{sol}(P')$  and assume that  $(z_p, a), (v_i, \tau) \in \sigma'$ . Then  $(z_p, a) \in \tau$ .*

**Proof.** Let  $t_{r_p}$  be the representative node picked for  $z_p$  in step (T2) and consider the path  $t_{r_p} = t_{j_1}, t_{j_2}, \dots, t_{j_k} = t_i$  in  $T$  connecting  $t_{r_p}$  with  $t_i$ . For every  $q = 1, \dots, k$  we have that  $z_p \in \chi(t_{r_p}) \cap \chi(t_i)$  and thus  $z_p \in \chi(t_{j_q})$  by condition (d3). Denote  $\tau_q \in \text{dom}(v_{j_q})$  the tuple for which  $(v_{j_q}, \tau_q) \in \sigma'$ . We shall show by induction on  $q$  that  $(z_p, a) \in \tau_q$  for every  $q = 1, \dots, k$ . Since  $\tau_k = \tau$ , we then have that  $(z_p, a) \in \tau$ .

Since  $t_1 = t_{r_p}$ , we have that  $(z_p, a) \in \tau_1$  by constraint  $c'_p$  added to  $C'$  in step (T2). Assume now that  $q > 1$  and consider constraint  $c'_{j_{q-1}, j_q}$  added in step (T1). The induction hypothesis implies that  $(z_p, a) \in \tau_{q-1}$ , and by definition of  $rel(c'_{j_{q-1}, j_q})$  we also have that  $(z_p, a) \in \tau_q$ . ◀

We are now ready to prove the correctness of the construction.

► **Lemma 27.**  $(\mathbf{x}, P')$  is a tree binary encoding of BCG constraint  $c^* = (\mathbf{x}, P)$ .

**Proof.** The constraint graph of  $P'$  is a tree that originates from  $T$  by adding leaves corresponding to the constraints  $c'_i$  added in step (T2). We shall show that  $sol(P')[\mathbf{z}] = sol(P)$ . Then  $rel(c^*) = sol(\mathbf{z}, C)[\mathbf{x}] = sol(\mathbf{z}', C')[\mathbf{x}]$  and the proposition follows.

Assume first that we have a solution  $\sigma' \in sol(P')$ . Denote  $\sigma = \sigma'[\mathbf{z}]$  and let us show that  $\sigma$  satisfies all constraints of  $P$ . Let  $c \in C$  be a constraint with  $scp(c) = \{z_p, z_q\}$ . We have  $(z_p, a) \in \sigma$  and  $(z_q, b) \in \sigma$  for some  $a \in dom(z_p)$  and  $b \in dom(z_q)$ . By condition (d2) we have that  $scp(c) \subseteq \chi(t_i)$  for some  $t_i \in V_T$ . Consider literal  $(v_i, \tau) \in \sigma'$ . By Lemma 26, we have that  $(z_p, a) \in \tau$  and  $(z_q, b) \in \tau$ . Since  $\tau \in dom(v_i)$ , we have that  $\{(z_p, a), (z_q, b)\} = \tau[scp(c)] \in rel(c)$ . Since this holds for every constraint  $c \in C$ , we get that  $\sigma \in sol(P)$ .

Assume now that we have a solution  $\sigma \in sol(P)$ . Let us now define a tuple  $\sigma' = \sigma \cup \{(v_i, \sigma[\chi(t_i)]) \mid i = 1, \dots, N\}$ . Since  $\sigma \in sol(P)$ , we have that  $\sigma[\chi(t_i)] \in rel(C_i)$  and thus  $\sigma[\chi(t_i)] \in dom(v_i)$ . Tuple  $\sigma'$  is thus correctly defined. It also satisfies all constraints (T1) and (T2) and thus  $\sigma' \in sol(P')$ . ◀

**Proof of Theorem 24.** Assume that  $P' = (\mathbf{z}', C')$  is constructed as above. Then  $(\mathbf{x}, P')$  is a tree binary encoding of  $c^* = (\mathbf{x}, P)$  by Lemma 27. We may assume by [15] that  $|V_T| = O(m)$  and thus  $|\mathbf{z}'| = m + |V_T| = O(m)$ . For every variable  $z_i \in \mathbf{z}$  we have  $|dom(z_i)| \leq d$  by assumption. For every variable  $v_i \in \mathbf{z}' \setminus \mathbf{z}$  we have that  $|\chi(t_i)| \leq tw(G) + 1$  and thus  $|dom(v_i)| \leq d^{tw(G)+1}$  by the definition of  $dom(v_i)$ . ◀



# Large Neighborhood Beam Search for Domain-Independent Dynamic Programming

Ryo Kuroiwa   

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

J. Christopher Beck   

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

---

## Abstract

Large neighborhood search (LNS) is an algorithmic framework that removes a part of a solution and performs search in the induced search space to find a better solution. While LNS shows strong performance in constraint programming, little work has combined LNS with state space search. We propose large neighborhood beam search (LNBS), a combination of LNS and state space search. Given a solution path, LNBS removes a partial path between two states and then performs beam search to find a better partial path. We apply LNBS to domain-independent dynamic programming (DIDP), a recently proposed generic framework for combinatorial optimization based on dynamic programming. We empirically show that LNBS finds better quality solutions than a state-of-the-art DIDP solver in five out of nine benchmark problem types with a total of 8570 problem instances. In particular, LNBS shows a significant improvement over the existing state-of-the-art DIDP solver in routing and scheduling problems.

**2012 ACM Subject Classification** Computing methodologies → Discrete space search

**Keywords and phrases** Large Neighborhood Search, Dynamic Programming, State Space Search, Combinatorial Optimization

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.23

**Supplementary Material** *Software (Source Code)*: <https://github.com/domain-independent-dp/didp-rs/releases/tag/lpbs-cp23>

archived at [swh:1:dir:b531b883c4c9eda4e0452fe801a5b50f6a3970cc](https://swh.1:dir:b531b883c4c9eda4e0452fe801a5b50f6a3970cc)

**Funding** This research is supported by the Natural Sciences and Engineering Research Council of Canada.

## 1 Introduction

In constraint programming (CP), large neighborhood search (LNS) [34] achieves strong performance in solving combinatorial optimization problems such as routing [24] and scheduling problems [27]. LNS is an algorithmic framework that removes a part of a solution and then performs search in the induced partial search space (neighborhood) to find a better solution. Typically, LNS uses tree search to find a better solution in a partial search space, where each search node represents a partial assignment of decision variables, and a solution of a problem corresponds to a leaf node, where all variables are assigned values.

Dynamic programming (DP) is a powerful method for multiple combinatorial optimization problems [14, 15], and the hybridization of CP, decision diagrams, and DP is a topic of active research [1, 22, 23, 5, 28, 19, 17]. Recently, domain-independent dynamic programming (DIDP), a model-based paradigm for combinatorial optimization based on DP, has been proposed [25]. In DIDP, a model of a problem is represented by a state transition system. A solution corresponds to a path in a state space graph, where each vertex represents a state and each edge represents a transition between two states. The current state-of-the-art DIDP



© Ryo Kuroiwa and J. Christopher Beck;

licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 23; pp. 23:1–23:22

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

solver is complete anytime beam search (CABS) [26], which is based on beam search, an algorithm that searches for a path in a state space graph by maintaining a fixed number of states at a time.

In this paper, we propose large neighborhood beam search (LNBS), a combination of LNS and beam search in a state space graph. LNBS tries to improve a solution path by removing a partial path between two states and then performing beam search to find a better partial path. While LNBS has the freedom to select a neighborhood (i.e., a partial path to remove), we propose a strategy that dynamically adjusts the size of a neighborhood based on a multi-armed bandit problem. With our strategy, LNBS is complete, i.e., it finds and proves an optimal solution given enough time, but, of course, it is aimed at problems where its solution quality is more important than proved optimality. We implement LNBS for DIDP and empirically evaluate its performance. The experimental results show that LNBS outperforms CABS in five out of nine benchmark problem types in terms of solution quality. In addition, LNBS performs better than a commercial CP solver, which uses LNS [27], in seven problems while CABS is better than CP in six problems. Since LNBS performs particularly well in routing and scheduling problems, we also investigate the reason for this performance and gain insight from empirical analysis.

## 2 Background

We first introduce domain-independent dynamic programming and complete anytime beam search. Then, we present large neighborhood search (LNS). We also describe LNS with decision diagrams [18], a recently proposed method for combinatorial optimization that can be considered a combination of LNS and state space search.

### 2.1 Domain-Independent Dynamic Programming

A combinatorial optimization problem is to find a set of discrete decisions, e.g., a permutation, to minimize or maximize an objective function. In dynamic programming (DP), a problem is recursively formulated by decomposing it into subproblems, represented by *states*, and the optimal objective value of each subproblem is represented by the *value function*, which maps a state to a real number.

Domain-independent dynamic programming (DIDP) is a model-based paradigm for combinatorial optimization based on DP [25]. In DIDP, a DP formulation of a combinatorial optimization problem is defined by a state-transition system in Dynamic Programming Description Language (DyPDL). A DyPDL model is a seven-tuple  $\langle \mathcal{V}, S^0, \mathcal{K}, \mathcal{T}, \mathcal{B}, \mathcal{C}, h \rangle$  consisting of *state variables*  $\mathcal{V}$ , the *target state*  $S^0$ , *constants*  $\mathcal{K}$ , *transitions*  $\mathcal{T}$ , *base cases*  $\mathcal{B}$ , *state constraints*  $\mathcal{C}$ , and the *dual bound*  $h$ .

A state variable  $v \in \mathcal{V}$  has a type of *set*, *element*, or *numeric*. Each set and element variable  $v_i$  is associated with a set of objects  $N_i = \{0, \dots, n_i - 1\}$ . The domain of a set variable  $v_i$  is  $2^{N_i}$ , and the domain of an element variable  $v_i$  is  $N_i$ . A numeric variable takes a real value. A constant in  $\mathcal{K}$  is a value independent of the state variables.

A state is a complete value assignment to the state variables, and the target state  $S^0$  is a state. We denote the value of a state variable  $v_i$  in a state  $S$  by  $S[v_i]$ . The value of a state  $S$  is represented by the value function  $V(S)$ , and the objective of a DyPDL model is to compute the value of the target state,  $V(S^0)$ . We can define dominance between states based on state variables. If a state  $S$  dominates another state  $S'$ , denoted by  $S' \preceq S$ , then  $V(S)$  is equal or better (less/greater for minimization/maximization) than  $V(S')$ . For the details of dominance in DyPDL, please refer to previous work [25].

A transition  $\tau \in \mathcal{T}$  is a four-tuple  $\langle \text{eff}_\tau, \text{cost}_\tau, \text{pre}_\tau, \text{forced}_\tau \rangle$  defining how a state  $S$  is transformed to a successor state (a subproblem)  $S[\tau]$ , and how  $V(S)$  is computed based on the value of  $V(S[\tau])$ . The set of *effects*  $\text{eff}_\tau = \{(v_i, e_{v_i}) \mid v_i \in \mathcal{V}\}$  defines how each state variable  $v_i$  is updated by an *expression*  $e_{v_i}$ . The expression  $e_{v_i}$  consists of predefined operations on state variables and constants and returns  $S[\tau][v_i]$  given  $S$ . For example, for a set variable  $U$ ,  $U \setminus \{i\}$  is an expression representing removing an element  $i$  from  $U$ , which can be used as  $e_U$  and results in  $S[\tau][U] = S[U] \setminus \{i\}$ . The *cost expression*  $\text{cost}_\tau$  is an expression that takes  $V(S[\tau])$  in addition to  $S$  and returns a real number  $\text{cost}_\tau(V(S[\tau]), S)$ . The preconditions  $\text{pre}_\tau$  are conditions on the state variables, i.e., expressions returning a binary value  $\top$  or  $\perp$ . The preconditions define when the transition is *applicable*. For example, if  $\text{pre}_\tau = \{i \in U\}$ , then  $\tau$  is applicable in  $S$  iff  $i \in S[U]$ . The flag  $\text{forced}_\tau$  is a boolean value indicating whether the transition is a *forced transition*. Let  $\mathcal{T}(S)$  be the set of applicable transitions in state  $S$ . If forced transitions are applicable, the first defined one  $\tau$  is selected, and all other forced and non-forced transitions are ignored, i.e.,  $\mathcal{T}(S) = \{\tau\}$ . The value of  $V(S)$  is computed by taking the best  $\text{cost}_\tau(V(S[\tau]), S)$  over all  $\tau \in \mathcal{T}(S)$ .

Base cases  $\mathcal{B}$  are sets of conditions. For any  $B \in \mathcal{B}$ , if a state  $S$  satisfies all conditions in  $B$ , denoted by  $S \models B$ , then it is called a *base state*, and  $V(S)$  is defined non-recursively by an expression  $e_B(S)$ . State constraints  $\mathcal{C}$  are conditions that must be satisfied by all states. If one of the state constraints  $c \in \mathcal{C}$  is violated, denoted by  $S \not\models \mathcal{C}$ , then  $V(S) = \infty$  ( $V(S) = -\infty$ ) for minimization (maximization). The dual bound  $h(S)$  is a lower (upper) bound on  $V(S)$  for minimization (maximization).

Overall, if the problem is minimization, the DP formulation is defined as follows.

$$\text{compute } V(S^0) \tag{1}$$

$$\text{s.t. } V(S) = \begin{cases} \min_{\tau \in \mathcal{T}(S)} \text{cost}_\tau(V(S[\tau]), S) & \text{if } S \models \mathcal{C} \wedge \forall B \in \mathcal{B}, S \not\models B \\ e_B(S) & \text{if } S \models \mathcal{C} \wedge \exists B \in \mathcal{B}, S \models B \\ \infty & \text{if } S \not\models \mathcal{C} \end{cases} \tag{2}$$

$$V(S) \leq V(S') \quad \text{if } S' \preceq S \tag{3}$$

$$V(S) \geq h(S). \tag{4}$$

The first line states that the optimal objective value is  $V(S^0)$ . Equation (2) recursively defines the value function  $V$ . Inequalities (3) and (4) are bounds on the value function. For maximization, we replace  $\min$  with  $\max$  in the first line of Equation (2) and swap  $\leq$  and  $\geq$  in Inequalities (3) and (4). A *solution* for the DP formulation is a sequence of transitions that transforms the target state  $S^0$  into a base state. Concretely, for a sequence of transitions  $x = \langle x_1, \dots, x_n \rangle$ , let  $S^{i+1} = S^i[x_{i+1}]$  for  $i = 0, \dots, n-1$ . Then,  $x$  is a solution if  $x_{i+1} \in \mathcal{T}(S^i)$ ,  $S^i \models \mathcal{C}$ , and  $\forall B \in \mathcal{B}, S^i \not\models B$  for  $i = 0, \dots, n-1$ ,  $S^n \models \mathcal{C}$ , and  $\exists B \in \mathcal{B}, S^n \models B$ . The solution is an *optimal solution* if  $\text{cost}_{\tau_i}(V(S^{i+1}), S^i) = V(S^i)$  for  $i = 0, \dots, n-1$  in addition.

## 2.2 Complete Anytime Beam Search for DIDP

Previous research has shown that a subset of DyPDL models can be solved by cost-algebraic heuristic search [11], a generalized version of the shortest path algorithm [25]. Multiple DIDP solvers using cost-algebraic heuristic search algorithms have been proposed [25, 26]. These solvers perform *state space search*, which finds a path from the target state to a base state in a *state space graph*, a directed graph where each vertex is a state. In the state space graph, an edge from state  $S$  to  $S[\tau]$  exists if  $\tau \in \mathcal{T}(S)$ . Following the previous work, we assume that  $\text{cost}_\tau(V(S[\tau]), S)$  is expressed as  $w_\tau(S) \times V(S[\tau])$  where  $w_\tau$  is an expression returning



■ **Algorithm 1** Beam Search for DyPDL.

---

```

1: function BEAMSEARCH( $\bar{f}$ ,  $b$ )
2:    $g(S^0) \leftarrow 0, f(S^0) \leftarrow h(S^0), x(S^0) \leftarrow \langle \rangle.$ 
3:    $O \leftarrow \{S^0\}, \bar{x} \leftarrow \text{NULL}, \text{complete} \leftarrow \top.$ 
4:   while  $O \neq \emptyset$  and  $\bar{x} = \text{NULL}$  do
5:      $G \leftarrow \emptyset.$  ▷ A set of states in the next layer.
6:     for all  $S \in O$  do
7:       if  $\exists B \in \mathcal{B}$  such that  $S \models B$  then ▷ A base state.
8:         if  $g(S) \times e_B(S) < \bar{f}$  then ▷ A better solution.
9:            $\bar{f} \leftarrow g(S) \times e_B(S), \bar{x} \leftarrow x(S).$ 
10:        else
11:          for all  $\tau \in \mathcal{T}(S) : S[\tau] \models \mathcal{C}$  do
12:            if  $\nexists S' \in G$  such that  $S[\tau] \preceq S'$  and  $g(S) \times w_\tau(S) \geq g(S')$  then
13:               $x(S[\tau]) \leftarrow \langle x(S); \tau \rangle.$ 
14:               $g(S[\tau]) \leftarrow g(S) \times w_\tau(S), f(S[\tau]) \leftarrow g(S[\tau]) \times h(S[\tau]).$ 
15:              if  $\exists S' \in G$  such that  $S' \preceq S[\tau]$  and  $g(S[\tau]) \leq g(S')$  then
16:                 $G \leftarrow G \setminus \{S'\}.$  ▷ Remove a dominated state.
17:                if  $f(S[\tau]) < \bar{f}$  then ▷ Pruning by the primal bound.
18:                   $G \leftarrow G \cup \{S[\tau]\}.$ 
19:             $O \leftarrow \{S \in G \mid f(S) < \bar{f}\}.$ 
20:            if  $|O| > b$  then
21:               $O \leftarrow$  the best  $b$  states in  $G$  minimizing  $f$ .
22:            complete  $\leftarrow \perp.$ 
23:          if  $O \neq \emptyset$  then
24:            complete  $\leftarrow \perp.$ 
25:          return  $\bar{x}, \text{complete}.$ 

```

---

a real value and  $\times$  is a binary operator satisfying a cost-algebra. In particular, we focus on nonnegative  $w_\tau$  and binary operators  $+$  or  $\max$ , i.e.,  $\text{cost}_\tau(V(S[\tau]), S) = w_\tau(S) + V(S[\tau])$  or  $\text{cost}_\tau(V(S[\tau]), S) = \max\{w_\tau(S), V(S[\tau])\}$ . The weight of the edge  $(S, S[\tau])$  is defined as  $w_\tau(S)$ , and the cost of a path from the target state  $S^0$ , which corresponds to a sequence of the transitions  $x = \langle x_1, \dots, x_n \rangle$ , is  $\text{cost}_x(S^0) = \prod_{i=0}^{n-1} w_{x_{i+1}}(S^i)$  where  $S^{i+1} = S^i[x_{i+1}]$  for  $i = 0, \dots, n-1$ . As each edge weight is nonnegative, the cost of a path is nonnegative and non-decreasing in length. In this paper, we focus on minimization while DyPDL and cost-algebraic heuristic search can handle both minimization and maximization.

The state-of-the-art cost-algebraic heuristic search solver is complete anytime beam search (CABS) [36, 26]. CABS performs beam search, which searches at most  $b$  states in the *open list*  $O$  at each layer of the state space graph. We show the pseudo-code of beam search in Algorithm 1. In addition to a DyPDL model and  $b$ , beam search takes the primal bound  $\bar{f}$  as an input, which is the best-known objective value and could be infinity. With each state  $S$ , the best path  $x(S)$  from  $S^0$  and its cost  $g(S) = \text{cost}_{x(S)}(S^0)$  (the *g-value*) are maintained in lines 13 and 14. Starting from  $O = \{S^0\}$ , beam search processes a state  $S$  in  $O$ . If  $S$  is a base state, and the best path to  $S$  has a better cost than  $\bar{f}$  in line 8, then  $\bar{f}$  and the solution  $\bar{x}$  are updated. Otherwise,  $S[\tau]$  is added to the candidate set  $G$  for each transition  $\tau \in \mathcal{T}(S)$ , which is called the *expansion* of  $S$ , and we say that  $S$  is expanded (lines 11–18). When expanding  $S \in O$ , if there exists a state  $S' \in G$  that dominates  $S[\tau]$  and  $g(S') \leq g(S) \times w_\tau(S)$ , then  $S[\tau]$  is not added to  $G$ . In addition to  $g(S[\tau])$ , the priority

$f(S[\tau]) = g(S[\tau]) \times h(S[\tau])$  (the *f-value*) is computed in line 14, which is a lower bound on the optimal path cost from  $S^0$  to a base state via  $S[\tau]$ . If  $f(S[\tau]) \geq \bar{f}$ , the corresponding path does not lead to an improved solution, so  $S[\tau]$  is not added to  $G$  in line 18. After expanding all states,  $O$  is updated to the best  $b$  states in  $G$  according to  $f$  in lines 20-21. This procedure is repeated until a solution whose cost is better than the given primal bound is found, or no successor states are generated. The variable `complete` maintains whether the search is complete. If states in  $G$  are pruned due to the beam width  $b$  (line 22), or  $O$  is not empty when a solution is found (line 24), there may exist a better solution, so the search is not complete. If `complete` =  $\top$ , then  $\bar{x}$  is the optimal solution if it is not NULL, or the model is infeasible if  $\bar{x} = \text{NULL}$ . CABS performs a sequence of beam search with exponentially increasing beam width  $b = 1, 2, 4, \dots$  using the best objective value found so far as the primal bound  $\bar{f}$  until the optimality of the best solution or the infeasibility is proved.

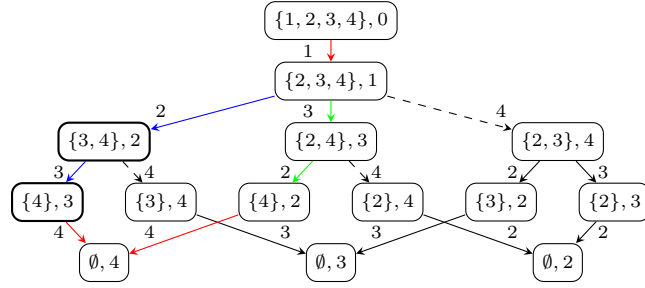
## 2.3 Large Neighborhood Search

Large neighborhood search (LNS) iteratively removes a part of a solution and solves the resulting subproblem (neighborhood) [34]. A solution for a CP problem is represented as a complete value assignment to decision variables. Given a solution, LNS removes a subset of the value assignments and solves the subproblem where the remaining variables are fixed to the values assigned in the original solution. Typically, a tree search algorithm is used. In a tree search algorithm, a search node is a partial value assignment to decision variables, successor nodes are generated by assigning a value to an unassigned variable, and a solution corresponds to a leaf node, where all variables are assigned values.

### 2.3.1 Large Neighborhood Search with Decision Diagrams

For combinatorial optimization, recent work proposed LNS with decision diagrams (DD-LNS) [18]. Although DD-LNS was not explicitly framed as state space search, we interpret it as a state space search algorithm. While DD-LNS is independent of DIDP, it also uses the DP formulation of a problem as input while assuming that the solution has  $n$  transitions. Given a sequence of transitions  $\langle x_1, \dots, x_n \rangle$ , DD-LNS keeps the first  $d$  transitions,  $\langle x_1, \dots, x_d \rangle$ , and searches for the remaining  $n - d$  transitions. To find such a sequence, DD-LNS constructs a decision diagram (DD), a directed graph where nodes are partitioned into layers. In the constructed DD, each vertex corresponds to a state, and each edge corresponds to a transition, so it is a state space graph. The first layer in the DD contains only the node corresponding to  $S^d$ , where  $S^i = S^{i-1} \llbracket x_i \rrbracket$  for  $i = 1, \dots, d$ . DD-LNS iteratively constructs a layer by applying transitions to the states in the current layer until reaching layer  $n - d + 1$ .

Because constructing an exact DD is intractable, DD-LNS constructs a restricted DD, which keeps a subset of states in the exact DD. In each layer, states satisfying certain conditions are kept, some of which are selected randomly. Let the number of such states be  $K$ . If  $K$  is smaller than a parameter  $W$ , from the remaining states, DD-LNS also keeps the best  $W - K$  states that minimize a priority function, the rough lower bound (RLB). The RLB of a state is a lower bound on the cost of a solution path via that state, which is the same as the *f-value*. If the RLB of a state is larger than the best solution cost, the state is removed from the DD as it does not lead to a better solution. Therefore, the procedure of constructing a restricted DD can be considered beam search with randomization. Indeed, the authors acknowledged that DD-LNS is a hybridization of LNS and beam search [18]. DD-LNS decreases  $d$  by 1 if a better solution is not found with  $d$ , starting from  $d = n - 2$  and restarting from  $d = n - 2$  if  $d = 0$ . When  $d = 0$  and the restricted DD keeps all states except



■ **Figure 1** Partial state space graph induced by the prefix  $\langle 1 \rangle$  and the suffix  $\langle 4 \rangle$  (highlighted in red) in Example 1. The current partial path is highlighted in blue and an alternative partial path is highlighted in green. Dashed transitions conflict with the suffix (explained in Section 3.2).

for those removed based on RLB, then it is the exact DD, so DD-LNS proves the optimality of the solution. Since DD-LNS can be interpreted as a state space search algorithm and is designed for combinatorial optimization, we implement it for DIDP and experimentally compare it with our method.

### 3 Large Neighborhood Beam Search

We start with a simple idea of LNS for state space search: given a solution path,  $x = \langle x_1, \dots, x_n \rangle$  which connects states  $\langle S^0, \dots, S^n \rangle$ , we remove a partial path  $\langle x_i, \dots, x_{i+d-1} \rangle$  and search for a better partial path from  $S^{i-1}$  to  $S^{i+d-1}$ . If we find a better solution  $\langle x_1, \dots, x_{i-1}, x'_i, \dots, x'_{i+d-1}, x_{i+d}, \dots, x_n \rangle$ , we repeat this procedure with the new solution. While the overview of the algorithm is simple, there are design choices on how to select a partial path to remove and how to search for a better partial path. The novelty of our method compared to existing methods arises from such choices in addition to the fact that it is used for DIDP. First, we describe the modifications of beam search for DyPDL to search for a partial path from  $S^{i-1}$  to  $S^{i+d-1}$ . Then, we propose strategies to select a partial path to remove.

#### 3.1 Beam Search for DyPDL in a Partial State Space Graph

We want to find a path from  $S^{i-1}$  to  $S^{i+d-1}$  instead of from  $S^0$  to a base state. We could modify line 8 in Algorithm 1 so that it checks if  $S = S^{i+d-1}$  instead of  $\exists B \in \mathcal{B}, S \models B$ . However, in DyPDL, it may not be desirable as shown in the following example.

► **Example 1.** Consider the following DP formulation, where  $U \subseteq \{0, 1, 2, 3, 4\}$  is a set variable,  $k \in \{0, 1, 2, 3, 4\}$  is an element variable, and  $c_{lj}$  for  $l, j \in \{0, 1, 2, 3, 4\}$  is a constant.

compute  $V(\{1, 2, 3, 4\}, 0)$

$$V(U, k) = \begin{cases} \min_{j \in U} c_{kj} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \\ 0 & \text{if } U = \emptyset. \end{cases}$$

Each transition in the DyPDL model has precondition  $j \in U$  and effect  $(U, U \setminus \{j\})$  for some  $j$ , and so we denote each transition by  $j$ . Each solution corresponds to a permutation of the transitions 1, 2, 3, 4. A solution  $\langle 1, 2, 3, 4 \rangle$  connects a sequence of states  $\langle (\{1, 2, 3, 4\}, 0), (\{2, 3, 4\}, 1), (\{3, 4\}, 2), (\{4\}, 3), (\emptyset, 4) \rangle$ . Consider removing  $\langle 2, 3 \rangle$  from the solution. We visualize the partial state space graph in Figure 1. An algorithm tries to find a path from  $(\{2, 3, 4\}, 1)$  to  $(\{4\}, 3)$ . The original one,  $\langle 2, 3 \rangle$ , is only the path. However, a partial path  $\langle 3, 2 \rangle$  from  $(\{2, 3, 4\}, 1)$  to  $(\{4\}, 2)$  also results in a valid solution  $\langle 1, 3, 2, 4 \rangle$ .

■ **Algorithm 2** Beam Search in a partial state space graph.

---

```

1: function BEAMSEARCHFORPARTIALPATH( $\bar{f}$ ,  $b$ , prefix, suffix)
2:    $\hat{S} \leftarrow S^0 \llbracket \text{prefix} \rrbracket$ ,  $g(\hat{S}) \leftarrow \text{cost}_{\text{prefix}}(S^0)$ ,  $f(\hat{S}) \leftarrow g(\hat{S}) \times h(\hat{S})$ ,  $x(\hat{S}) \leftarrow \text{prefix}$ .
3:    $O \leftarrow \{\hat{S}\}$ ,  $\bar{x} \leftarrow \text{NULL}$ , complete  $\leftarrow \top$ .
4:   while  $O \neq \emptyset$  and  $\bar{x} = \text{NULL}$  do
5:      $G \leftarrow \emptyset$ . ▷ A set of states in the next layer.
6:     for all  $S \in O$  do
7:        $S' \leftarrow S$ , success  $\leftarrow \perp$ .
8:       if  $\exists B \in \mathcal{B}$  and  $S' \models B$  then
9:         success  $\leftarrow \top$ . ▷ A base state, success.
10:      else
11:        for  $\tau \leftarrow \text{suffix}_1, \dots, \text{suffix}_{n-i-d+1}$  do ▷ Rollout of the suffix.
12:          if  $S' \not\models \text{pre}_\tau$  then
13:            break. ▷ Preconditions are not satisfied, fail.
14:           $S' \leftarrow S' \llbracket \tau \rrbracket$ ,  $g(S') \leftarrow g(S') \times w_\tau(S')$ ,  $x(S') \leftarrow \langle x(S'); \tau \rangle$ .
15:          if  $S' \not\models \mathcal{C}$  then
16:            break. ▷ State constraints are not satisfied, fail.
17:          if  $\exists B \in \mathcal{B}$  such that  $S' \models B$  then
18:            success  $\leftarrow \top$ . ▷ A base state, success.
19:          break.
20:        if success then
21:          if  $g(S') \times e_B(S') < \bar{f}$  then ▷ A better solution.
22:             $\bar{f} \leftarrow g(S') \times e_B(S')$ ,  $\bar{x} \leftarrow x(S')$ .
23:        else
24:          for all  $\tau \in \mathcal{T}(S) : S \llbracket \tau \rrbracket \models \mathcal{C}$  do
25:            if  $\nexists S' \in G$  such that  $S \llbracket \tau \rrbracket \preceq S'$  and  $g(S) \times w_\tau(S) \geq g(S')$  then
26:               $x(S \llbracket \tau \rrbracket) \leftarrow \langle x(S); \tau \rangle$ .
27:               $g(S \llbracket \tau \rrbracket) \leftarrow g(S) \times w_\tau(S)$ ,  $f(S \llbracket \tau \rrbracket) \leftarrow g(S \llbracket \tau \rrbracket) \times h(S \llbracket \tau \rrbracket)$ .
28:              if  $\exists S' \in G$ ,  $S' \preceq S \llbracket \tau \rrbracket \wedge g(S \llbracket \tau \rrbracket) \leq g(S')$  then
29:                 $G \leftarrow G \setminus \{S'\}$ . ▷ Remove a dominated state.
30:              if  $f(S \llbracket \tau \rrbracket) < \bar{f}$  then ▷ Pruning by the primal bound.
31:                 $G \leftarrow G \cup \{S \llbracket \tau \rrbracket\}$ .
32:             $O \leftarrow \{S \in G \mid f(S) < \bar{f}\}$ .
33:          if  $|G| > b$  then
34:             $O \leftarrow$  the best  $b$  states in  $G$  minimizing  $f$ .
35:          complete  $\leftarrow \perp$ .
36:        if  $O \neq \emptyset$  then
37:          complete  $\leftarrow \perp$ .
38:      return  $\bar{x}$ , complete.

```

---

Considering the above example, instead of focusing on a partial path to a state, we focus on a partial path to a *suffix* of the solution path. Given a solution path  $\langle x_1, \dots, x_n \rangle$ , if we remove a partial path  $\langle x_i, \dots, x_{i+d-1} \rangle$ , then  $\langle x_1, \dots, x_{i-1} \rangle$  is the prefix, and  $\langle x_{i+d}, \dots, x_n \rangle$  is the suffix. For a partial path  $\langle x'_i, \dots, x'_{i+d-1} \rangle$ , we want to check if  $\langle x_1, \dots, x_{i-1}, x'_i, \dots, x'_{i+d-1}, x_{i+d}, \dots, x_n \rangle$  is a valid solution. Therefore, for a state  $S$  found by a search algorithm, we perform a rollout of the suffix from  $S$  and check if each of resulting states satisfies the state constraints and a

---

**Algorithm 3** Large Neighborhood Beam Search (LNBS).
 

---

**Input:** initial feasible solution  $\bar{x}$ .

**Output:** solution  $\bar{x}$  and if the optimality or infeasibility is proved.

```

1: while the time limit is not reached do
2:    $n \leftarrow |\bar{x}|, \bar{f} \leftarrow \text{cost}_{\bar{x}}(S^0)$ .
3:   Select  $d$  such that  $2 \leq d \leq n$ .
4:   Select  $i$  such that  $1 \leq i \leq n - d + 1$ .
5:   Select beam width  $b$ .
6:    $\text{prefix} \leftarrow \langle \bar{x}_1, \dots, \bar{x}_{i-1} \rangle, \text{suffix} \leftarrow \langle \bar{x}_{i+d}, \dots, \bar{x}_n \rangle$ .
7:    $x, \text{complete} \leftarrow \text{BEAMSEARCHFORPARTIALPATH}(\bar{f}, b, \text{prefix}, \text{suffix})$ 
8:   if  $x \neq \text{NULL}$  then
9:      $\bar{x} \leftarrow x$ .
10:  if  $i = 0 \wedge d = n \wedge \text{complete}$  then
11:    return  $\bar{x}, \top$ .
12: return  $\bar{x}, \perp$ .

```

---

base case. We show the modified version of beam search in Algorithm 2. This algorithm takes a prefix  $\text{prefix}$  and a suffix  $\text{suffix}$  as input. In line 2, we denote the state resulting from applying the prefix to the target state by  $\hat{S} = S^0 \llbracket \text{prefix} \rrbracket$  and initialize the open list  $O$  with  $\hat{S}$  in line 3. In lines 8–22, the algorithm performs a rollout of the suffix from  $S$  and checks if it results in a better solution. Other parts are the same as Algorithm 1.

We use this modified version of beam search in large neighborhood beam search (LNBS) as shown in Algorithm 3. In line 2,  $|\bar{x}|$  denotes the length of the current solution  $\bar{x}$ . In lines 3–5, LNBS selects parameters  $d$ ,  $i$ , and  $b$ . In line 7, LNBS performs beam search in the neighborhood. If an improving solution is found, LNBS updates the current solution  $\bar{x}$  in line 9. If the searched neighborhood is the original search space, i.e.,  $i = 1$  and  $d = n$ , and beam search proves the optimality or infeasibility, LNBS terminates in line 11. Therefore, if it is guaranteed to select  $i = 1$  and  $d = n$  with sufficiently large  $b$  given enough time, LNBS is guaranteed to find the optimal solution or prove the infeasibility, i.e., it is complete. CABS can be considered a configuration of LNBS, where  $i = 1$ ,  $d = n$ , and  $b$  increases exponentially. DD-LNS can also be considered a configuration of LNBS, where  $i$  ranges from  $n - 2$  to 1,  $d$  is  $n - i + 1$ , and  $b$  is fixed to  $W$  while beam search is extended with the randomization mechanism. We will describe the strategies that we use to select  $d$ ,  $i$ , and  $b$  below.

### 3.2 Removing Conflicting Transitions

In Example 1, consider finding a partial path from a prefix  $\langle 1 \rangle$ , which results in state  $\hat{S} = (\{2, 3, 4\}, 1)$ , to a suffix  $\langle 4 \rangle$  using beam search. In  $\hat{S}$ , three transitions 2, 3, and 4 are applicable. However, applying transition 4 does not lead to a feasible solution because it is already used in the suffix and cannot be applied twice: it requires  $4 \in U$  and removes 4 from  $U$ , but no other transition adds 4 to  $U$ , so applying 4 makes the suffix inapplicable. Generalizing this example, if we know that a transition  $\tau$  makes a transition  $\tau'$  in the suffix inapplicable, then we can ignore  $\tau$  when searching for a partial path. In particular, we focus on the effects of  $\tau$  that add/remove an element to/from a set variable and the preconditions of  $\tau'$  that require the element to be/not to be in that set variable.

► **Proposition 2.** *Suppose that a DyPDL model  $\langle \mathcal{V}, S^0, \mathcal{K}, \mathcal{T}, \mathcal{B}, \mathcal{C}, h \rangle$  has a set variable  $U \in \mathcal{V}$  whose domain is  $2^N$ , where  $N$  is a set of objects. There does not exist a solution  $\langle x_1, \dots, x_n \rangle$  such that any pair of  $\tau = x_i$  and  $\tau' = x_j$  for  $1 \leq i < j \leq n$  satisfy either of the following*

conditions:

1. There exists  $k \in N$  such that  $(U, U \setminus \{k\}) \in \text{eff}_\tau$ ,  $(k \in U) \in \text{pre}_{\tau'}$ , and each  $\tau'' \in \mathcal{T} \setminus \{x_i, x_j\}$  does not change  $U$  or  $(U, U \setminus \{l\}) \in \text{eff}_{\tau''}$  for some  $l \in N$ .
2. There exists  $k \in N$  such that  $(U, U \cup \{k\}) \in \text{eff}_\tau$ ,  $(k \notin U) \in \text{pre}_{\tau'}$ , and each  $\tau'' \in \mathcal{T} \setminus \{x_i, x_j\}$  does not change  $U$  or  $(U, U \cup \{l\}) \in \text{eff}_{\tau''}$  for some  $l \in N$ .

**Proof.** For the first case,  $x_i$  removes  $k$  from  $U$ , and no other transition adds  $k$  to  $U$ . Since  $x_j$  requires  $k$  to be in  $U$ , once we apply  $x_i$ , we cannot apply  $x_j$  later. The other case is proved similarly.  $\blacktriangleleft$

Before starting beam search in a neighborhood, we remove a transition  $\tau$  from the model if there exists a transition  $\tau'$  in the suffix such that  $\tau$  and  $\tau'$  satisfy one of the conditions in Proposition 2. Detecting such a pair of transitions is done once at the beginning by checking the expression trees representing the preconditions and effects of transitions.

### 3.3 Bandit-Based Depth Selection

Selecting the depth of a neighborhood,  $d$ , in line 3 of Algorithm 3 is non-trivial. If  $d$  is too small, it is unlikely that an improving solution exists. However, if  $d$  is too large, each neighborhood search takes a long time. We want to select  $d$  such that the total cost improvement is maximized within the time limit.

We formulate the depth selection as the budgeted multi-armed bandit problem with continuous random costs [35]. We have the set of depths  $D \subseteq \{2, \dots, n\}$ . If we select a depth  $d \in D$  and perform search in line 7 of Algorithm 3, we obtain a new solution  $x$  by spending search time  $t$ . As  $t$  is assumed to take a value in  $[0, 1]$  in the budgeted multi-armed bandit problem, we divide the actual time by the time limit  $T$ . If the cost  $\text{cost}_x(S^0)$  is smaller than the current best solution cost  $\bar{f}$ , the reward is  $r = (\bar{f} - \text{cost}_x(S^0))/\bar{f}$ . If no better solution is found, the reward is  $r = 0$ . We call this process a *round*, and we repeat rounds until reaching the time limit  $T$ . We do not know the reward  $r$  and time  $t$  before finishing a round, so we use random variables  $r_{dk}$  and  $t_{dk}$  representing the reward and time if depth  $d$  is used at round  $k$ . Let  $a$  be a strategy that selects a depth  $a_k$  in the round  $k$ . The number of rounds performed by  $a$  by the time limit,  $K_{aT}$ , is also a random variable. The objective is to find a strategy  $a$  that maximizes the total expected reward  $\mathbb{E}[\sum_{k=1}^{K_{aT}} r_{a_k k}]$ .

We use Budgeted-UCB [35]. At each round, if some depths in  $D$  have not been selected before, Budgeted-UCB selects one of them. Otherwise, let  $m_{dk}$  be the number of rounds where the depth  $d$  is selected up to round  $k - 1$ , and let  $\bar{r}_{dk}$  and  $\bar{t}_{dk}$  be the average reward and search time for  $d$  up to round  $k - 1$ . Budgeted-UCB selects the depth  $d$  that maximizes

$$\frac{\bar{r}_{dk}}{\bar{t}_{dk}} + \frac{\epsilon_{dk}}{\bar{t}_{dk}} + \frac{\epsilon_{dk}}{\bar{t}_{dk}} \frac{\min\{\bar{r}_{dk} + \epsilon_{dk}, 1\}}{\max\{\bar{t}_{dk} - \epsilon_{dk}, \lambda\}} \quad (5)$$

where  $\epsilon_{dk} = \sqrt{\frac{2 \log(k-1)}{m_{dk}}}$  and  $\lambda$  is a positive lower bound of the search time of each round.

In practice, we initialize  $D = \{2, 4, 8, \dots, 2^l, n\}$ , where  $n$  is the length of the initial feasible solution and  $l$  is the maximum integer such that  $2^l < n$ . If we get a solution whose length  $n'$  is different from  $n$  at round  $k$ , we replace  $n$  with  $n'$  in  $D$  using  $m_{n',k+1} = m_{n,k+1}$ ,  $\bar{r}_{n',k+1} = \bar{r}_{n,k+1}$ , and  $\bar{t}_{n',k+1} = \bar{t}_{n,k+1}$  and ignore depths greater than  $n'$  in  $D$ . If multiple depths have not been selected before or have the same value, we select the minimum depth among them. For  $\lambda$ , we use the time of the first round divided by 10 while there is no guarantee that it is a lower bound. Thus, the theoretical analysis of Budgeted-UCB studied in the original paper [35] does not necessarily apply to our setting. In addition, while Xia et al. [35] assumed that the pairs  $\{(r_{dk}, t_{dk})\}_{k=1}^\infty$  are i.i.d., we do not have such a guarantee.

### 3.4 Start Selection

Once LNBS determines the depth  $d$  to use, it selects a starting point  $i$ , which induces the prefix and the suffix, in line 4 of Algorithm 3. We select  $i$  considering the cost change in a neighborhood. Concretely, the cost change by partial path  $\langle x_i, \dots, x_{i+d-1} \rangle$  is defined as

$$\delta_{di} = \text{cost}_{\langle x_1, \dots, x_{i+d-1} \rangle}(S^0) - \text{cost}_{\langle x_1, \dots, x_{i-1} \rangle}(S^0). \quad (6)$$

Since the path cost is non-decreasing,  $\delta_{di} \geq 0$ . We ignore  $i$  with  $\delta_{di} = 0$ <sup>1</sup> and select one uniformly at random from the remaining options.

Our second approach is to select  $i$  based on the probability biased by  $\delta_{di}$ . As we explain in the next subsection, for each  $d$  and  $i$ , the beam width  $b$  is maintained. Since smaller  $b_{di}$  leads to a shorter search time, we discount the probability of selecting  $i$  by  $b_{di}$ . Concretely, given the depth  $d$ , we can select the starting point  $i$  with the probability

$$p_{di} = \frac{\delta_{di}/b_{di}}{\sum_{j=1}^{n-d+1} \delta_{dj}/b_{dj}}. \quad (7)$$

### 3.5 Beam Width Selection

Given the depth  $d$  and the starting point  $i$ , LNBS selects a beam width  $b$  in line 5 of Algorithm 3. Here, we use a similar strategy to CABS: for each  $d$  and  $i$ , we initialize the beam width  $b_{di}$  to be 1 and update it to  $2b_{di}$  after each round with  $d$  and  $i$ . If we find an improved solution in line 7, we reset  $b_{d'i'}$  = 1 only for  $d'$  and  $i'$  such that  $i' > i$  or  $i' + d' < i + d$ ; if  $i' \leq i$  and  $i' + d' \geq i + d$ , the prefix and the suffix for the neighborhood induced by  $i'$  and  $d'$  do not change, and we know that a better partial path was not found with beam widths smaller than  $b_{d'i'}$ .<sup>2</sup> If the neighborhood is exhausted, i.e., `complete` =  $\top$  in line 7, we ignore the combination of  $d$  and  $i$  in lines 3 and 4 until a new solution is found and  $b_{di}$  is reset to 1. Since the number of neighborhoods is finite, LNBS eventually exhausts all the neighborhoods and finds the optimal solution, which guarantees completeness.

## 4 Experimental Evaluation

We compare LNBS with CABS, DD-LNS, CP, and mixed-integer programming (MIP).

### 4.1 Experimental Settings

As benchmarks, we use the same problems and instances used by previous work [26]: the traveling salesperson problem with time windows (TSPTW), the capacitated vehicle routing problem (CVRP), the multi-commodity pickup and delivery traveling salesperson problem (m-PDTSP), the single machine total weighted tardiness problem ( $1||\sum w_i T_i$ ), the talent scheduling problem (Talent), the simple assembly line balancing problem to minimize the number of stations (SALBP-1), the bin packing problem (BPP), and the graph-clear problem (GCP). We use the same DP, CP, and MIP models as the previous work [26].<sup>3</sup>

<sup>1</sup> Theoretically, a better solution may be found with such  $i$  if the suffix is not empty because a partial path may change the state from which the suffix is applied, which may change the cost of the suffix.

<sup>2</sup> Theoretically, a better solution may be found with beam width smaller than  $b_{d'i'}$  if the updated primal bound changes the search behavior.

<sup>3</sup> <https://github.com/Kurorororo/didp-models>

LNBS, CABS, and DD-LNS are implemented in didp-rs v0.3.2<sup>4</sup> using Rust 1.65.0. In LNBS and DD-LNS, CABS is run first to find a feasible solution, and then LNBS and DD-LNS are run to improve the solution. For DD-LNS, we use  $W = 1000$  and  $p = 0.1$  following the original paper [18]. As we described above, LNBS removes conflicting transitions from a suffix, selects the depth using Budgeted-UCB, and geometrically increases the beam width for each neighborhood. To select the starting point of a partial path, we consider two approaches, uniform and biased sampling. While biased sampling achieves better solution quality in CVRP and m-PDTSP, uniform sampling is better in TSPTW,  $1||\sum w_i T_i$ , SALBP-1, MOSP, and GCP (see Appendix A). In what follows, we only show results from uniform sampling. In Appendix A, we also evaluate the importance of removing conflicting transitions and Budgeted-UCB. We confirm that these mechanisms significantly improve the solution quality. A more comprehensive ablation study is left for future work.

We implemented the DP models using didppy, a Python interface for didp-rs. We use IBM ILOG CP Optimizer 22.1.0 for the CP models and Gurobi Optimizer 9.5.0 for the MIP models. CP Optimizer is known to use LNS [27]. The DP, CP, and MIP models are implemented in Python 3.10.2. All experiments are run on an Intel Xeon Gold 6148 processor with a single thread, an 8 GB memory limit, and a time limit of 1800 seconds. For LNBS and DD-LNS, we take the median of 5 runs.

Following the previous work [26], we use the primal gap and the primal integral to measure the performance [7]. If an algorithm finds a solution  $x^t$  with the cost  $f(x^t)$  at time  $t$ , and the optimal or best-known solution cost is  $f^*$ , the primal gap at  $t$  for the algorithm is

$$p(t) = \begin{cases} 1 & \text{if } f(x^t) \cdot f^* < 0 \\ 0 & \text{if } f(x^t) = f^* = 0 \\ \frac{|f(x^t) - f^*|}{\max\{|f(x^t)|, |f^*|\}} & \text{otherwise.} \end{cases} \quad (8)$$

If the algorithm does not find a solution at time  $t$ , then  $p(t) = 1$ . Let  $t_1, \dots, t_{l-1}$  be time points where a better solution is found,  $t_0 = 0$ , and  $t_l = T$  where  $T$  is the time limit. Then, the primal integral,  $P(T)$  is defined as

$$P(T) = \sum_{i=1}^l p(t_{i-1}) \cdot (t_i - t_{i-1}). \quad (9)$$

We use the primal gap at the time limit,  $p(T)$ , and the primal integral,  $P(T)$ , as the measures.

## 4.2 Experimental Results

We show the number of instances where the optimality of a solution is proved, the average primal gap at the time limit, and the average primal integral for each problem in Table 1. We omit MIP in Table 1 because it is outperformed by CP in the primal gap and the primal integral for all problem types (see Appendix A). In the number of optimally solved instances, MIP is the best in CVRP (with 26 problems solved). As reported by previous work [26], MIP is good at finding an optimal solution for small instances, while it fails to find a feasible solution for larger instances, which results in poor performance in the primal gap and the primal integral on average. In other problems, LNBS solves more instances optimally than MIP except for BPP, where LNBS solves 1139 and MIP solves 1157.

<sup>4</sup> <https://didp.ai>



■ **Table 1** Summary of the experimental result. “#” is the number of optimally solved instances, “gap” is the average primal gap at the time limit, and “p.i.” is the average primal integral.

	CP			CABS			DD-LNS			LNBS		
	#	gap	p.i.	#	gap	p.i.	#	gap	p.i.	#	gap	p.i.
TSPTW (340)	47	0.0259	49.0	<b>257</b>	0.0033	9.0	109	0.0100	23.7	241	<b>0.0016</b>	<b>5.7</b>
CVRP (207)	0	0.3174	601.1	<b>6</b>	0.1772	339.5	0	0.2504	461.6	<b>6</b>	<b>0.1640</b>	<b>316.8</b>
m-PDTSP (1178)	<b>1049</b>	0.0122	25.5	1030	0.0023	5.1	459	0.0102	21.7	1029	<b>0.0022</b>	<b>5.0</b>
$1  \sum w_i T_i$ (375)	150	<b>0.0009</b>	<b>3.5</b>	<b>286</b>	0.0346	74.4	100	0.0409	83.5	275	0.0051	13.0
Talent (1000)	0	0.0081	29.3	<b>232</b>	0.0173	38.2	0	0.0602	114.6	<b>232</b>	<b>0.0041</b>	<b>11.1</b>
SALBP-1 (2100)	1584	0.0046	28.4	<b>1799</b>	<b>0.0003</b>	<b>2.2</b>	1507	0.0067	13.5	1682	0.0022	7.3
BPP (1615)	<b>1234</b>	<b>0.0014</b>	7.7	1159	0.0017	<b>6.1</b>	775	0.0190	35.4	1139	0.0021	8.1
MOSP (570)	437	0.0044	13.0	<b>526</b>	<b>0.0000</b>	<b>0.4</b>	353	0.0203	37.5	523	0.0002	0.7
GCP (135)	1	0.0151	44.3	<b>103</b>	<b>0.0000</b>	<b>0.6</b>	3	0.0009	2.7	102	0.0001	<b>0.6</b>
Larger Instances												
m-PDTSP (240)	77	0.1491	285.9	<b>101</b>	0.0694	153.2	79	0.1345	265.7	98	<b>0.0652</b>	<b>146.6</b>
MOSP (760)	0	0.0676	150.6	<b>150</b>	<b>0.0002</b>	<b>4.4</b>	0	0.0402	72.7	148	0.0025	10.4
GCP (50)	<b>0</b>	0.5289	1268.3	<b>0</b>	<b>0.0013</b>	<b>10.8</b>	<b>0</b>	0.0764	137.8	<b>0</b>	0.0038	19.5

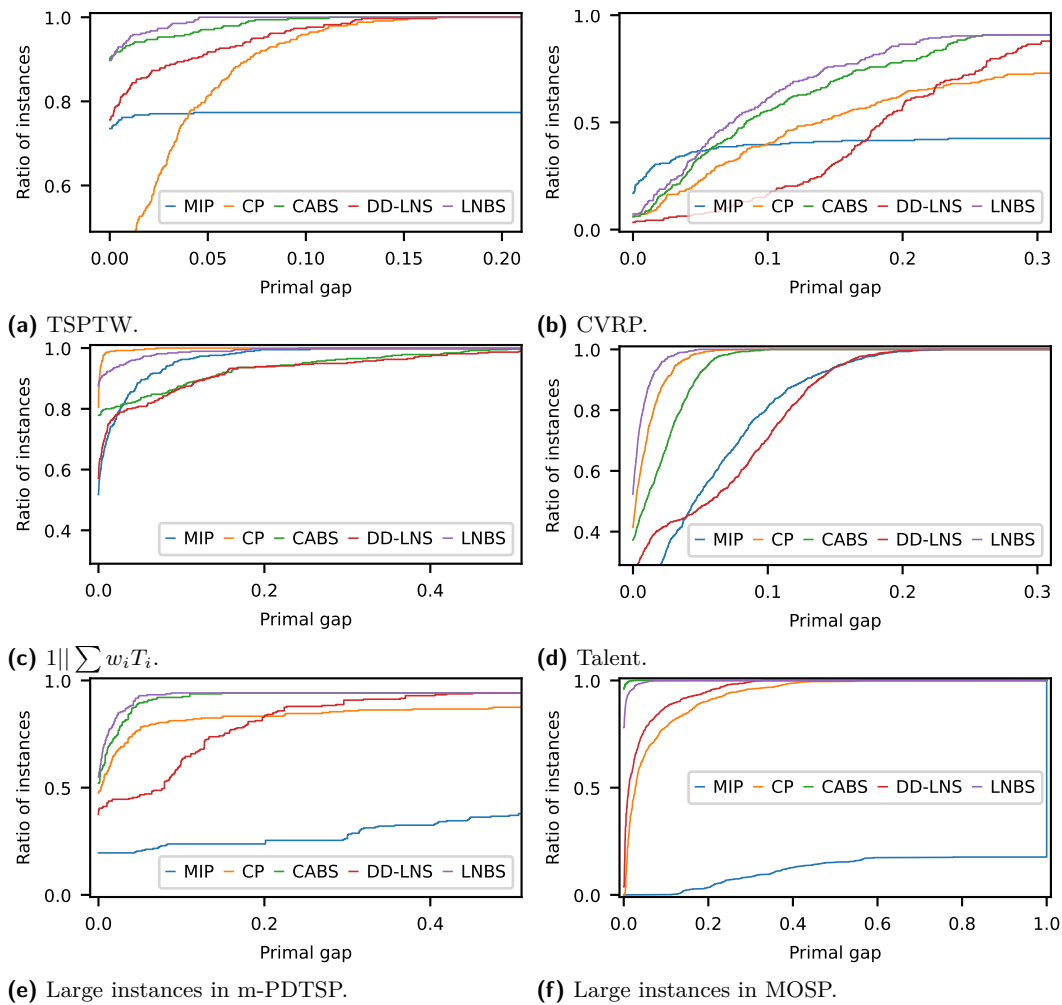
Compared to CABS, LNBS achieves the better primal gap and the primal integral in TSPTW, CVRP, m-PDTSP,  $1||\sum w_i T_i$ , and Talent. We show the distribution of the primal gap in TSPTW, CVRP,  $1||\sum w_i T_i$ , and Talent in Figure 2. The primal integral has a similar trend to the primal gap in these problems (see Appendix A). In contrast, CABS is better than LNBS in SALBP-1, BPP, MOSP, and GCP. These results are consistent with the observation that LNS is effective for routing and scheduling problems in CP [24, 27]. In the routing problems (TSPTW, CVRP, and m-PDTSP), CABS is already better than CP, and LNBS is even better than CABS. In  $1||\sum w_i T_i$ , while LNBS shows a significant improvement from CABS (0.0051 from 0.0346) outperforming MIP (0.0188), CP is still the best. However, in Talent, LNBS outperforms CP while CABS does not. Overall, LNBS is better than CP in seven problems while CABS is better than CP in six problems.

In TSPTW, the difference in the primal gap between LNBS and CABS (0.0016 and 0.0033) seems small, but it is because they achieve almost the same primal gap in many instances: they optimally solve all 135 instances in the Dumas set [10] and achieve almost the same average primal gap in the AFG set [2], which has 50 instances, and GendreauDumas Extended set [16], which has 130 instances. However, in the OhlmannThomas set [31], which has 25 instances, no instance is optimally solved, and LNBS shows a significant improvement in the primal gap (0.0184 from 0.0395).

In the number of optimally solved instances, CABS is equal to or better than LNBS in all problems. While CABS always searches the entire state space graph, LNBS searches multiple neighborhoods, and the entire state space graph is just one of them. Nevertheless, LNBS proves the optimality of more than 93% of instances that are optimally solved by CABS.

DD-LNS performs worse than CABS and LNBS in all the problems. Previous work has reported that DD-LNS is effective for TSPTW [18]. Note however that the DD-LNS and LNBS results in Table 1 are not the results reported by Gillard and Schaus. To validate that our implementation and experimental settings do not handicap DD-LNS, we compare the results of DD-LNS with those of the original paper in TSPTW.<sup>5</sup> Our DD-LNS implementation

<sup>5</sup> [https://github.com/xgillard/ijcai\\_22\\_DDLNS/blob/main/results/tsptw/ddlms/results\\_w1000\\_](https://github.com/xgillard/ijcai_22_DDLNS/blob/main/results/tsptw/ddlms/results_w1000_)



■ **Figure 2** Distribution of the primal gap at the time limit. Higher and left is better.

finds a better solution than the original in all instances with the time limit of 600 seconds. This difference is likely due to the difference between the DP models used by us (from Kuroiwa and Beck [26]) and Gillard and Schaus. In TSPTW, a solution is a tour that starts from a depot, visits each customer  $j$  within the time window  $[a_j, b_j]$ , and returns to the depot, and an optimal solution minimizes the total travel time. In both DP models, state variables are the set of unvisited customers  $U$ , the current customer  $i$ , and the current time  $t$ , and each transition corresponds to visiting a customer or the depot. Each DP model has a dual bound (called RLB by Gillard and Schaus), a lower bound on the optimal solution cost. While Gillard and Schaus use a dual bound based on a minimum spanning tree, Kuroiwa and Beck use a simpler one based on the minimum travel time between customers. In addition, Kuroiwa and Beck use information that was not considered by Gillard and Schaus. First, they use dominance between states based on the current time: a state  $S$  dominates another state  $S'$  if  $S[U] = S'[U]$ ,  $S[i] = S'[i]$ , and  $S[t] \leq S'[t]$ . Furthermore, since the time to visit customer  $j$  is underestimated by  $t + c_{ij}^*$ , where  $c_{ij}^*$  is the shortest travel time from  $i$  to  $j$ , they

t600.txt

define state constraints  $\forall j \in U, t + c_{ij}^* \leq b_j$ . The dominance and the state constraints are useful to prune states, which potentially explains the performance gap. Another difference is whether considering the time window constraints at the depot or not. In the benchmark instances used above<sup>6</sup> (and in Kuroiwa and Beck [26]), a time window  $[a_0, b_0]$  is defined for the depot. Gillard and Schaus explicitly model a required return to the depot within  $[a_0, b_0]$  while Kuroiwa and Beck do not. However, in the benchmark instances,  $b_0 \geq b_j + c_{j0}$  holds for all customers  $j$ , where  $c_{j0}$  is the travel time from  $j$  to the depot. Thus, if all customers are visited within the time windows, the depot can be reached within the time window, and explicit modeling of the depot return window is unnecessary.

### 4.3 Larger Instances

LNBS performs better than CABS in m-PDTSP and worse in MOSP and GCP, but the difference in the average primal gap is small. To evaluate the difference more clearly, we use larger instances for these problems.

In m-PDTSP, a vehicle visits all nodes in a graph, picks up some commodities at some nodes, and delivers them to others. Each commodity has a weight and the total weight of commodities that a vehicle can carry is limited by the capacity. In the benchmark set for m-PDTSP, three types of instances are used: Class 1, Class 2, and Class 3 [21], and Class 1 instances are generated from instances of the sequential ordering problem (SOP) [3]. We generate larger Class 1 instances by using 30 SOP instances in TSPLIB<sup>7</sup> that were not used by the previous work. The original instances have at most 47 nodes, and the new instances have 42 to 378 nodes. We use the same methods as the previous work [21] with the maximum weight  $q \in \{1, 5\}$  and the capacity  $Q \in \{5q, 10q, 20q, 100q\}$ , resulting in 240 instance in total.

In MOSP, an instance is represented by a matrix, and the original set uses at most  $125 \times 125$  matrices. We add instances using  $150 \times 150$  to  $1000 \times 1000$  matrices [8, 12].

In GCP, an instance is represented by a graph. The original instance set uses random and random planar graphs with 20, 30, and 40 nodes. We generate 50 instances using random graphs with 100 and 200 nodes following the method used by previous work [29].

As shown in Table 1, LNBS clearly outperforms CABS in m-PDTSP in the primal gap and the primal integral, but CABS is better in MOSP and GCP. We also show the distribution of the primal gap in m-PDTSP and MOSP in Figure 2. The primal integral has a similar tendency to the primal gap, and the result for GCP is qualitatively similar to that of MOSP.

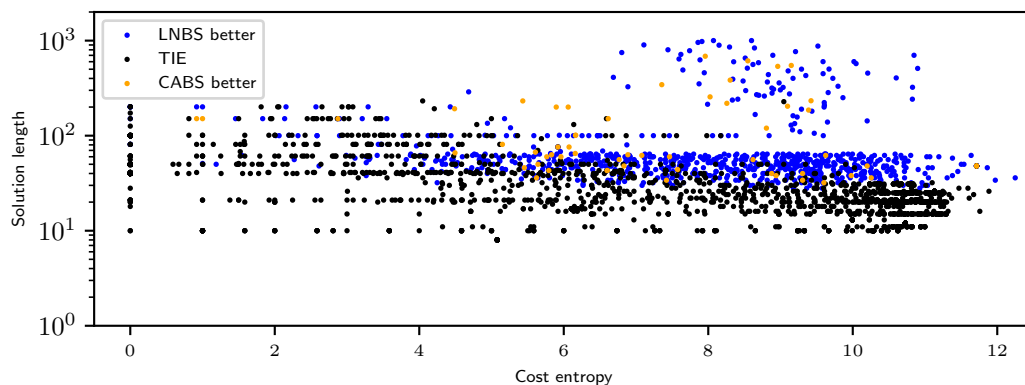
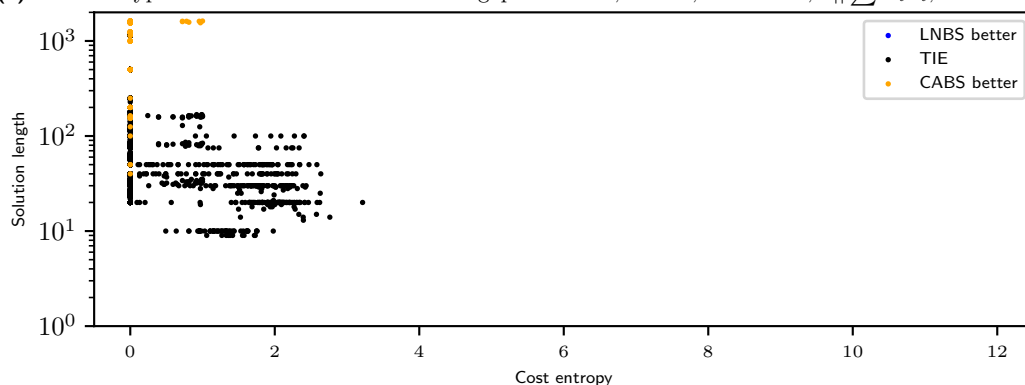
### 4.4 Analysis of Problem Characteristics

In routing problems, a solution is a route visiting all nodes in a graph, and its cost is the length of the route. In the DP models for these problems, each transition corresponds to visiting one node, and the cost of a partial path increases when a transition is applied. We expect that different partial solutions tend to have different costs, and it is relatively easy to find a better partial path; because the path costs are diverse, unless the current partial path is optimal, better partial paths are included in a partial state space graph with high density. In such a case, beam search is likely to find a better partial path although it searches in a fraction of the partial state space graph restricted by the beam width.

In contrast, in SALBP-1 and BPP, the problem is to pack weighted items into capacitated bins while minimizing the number of bins. In the DP models, each transition packs one item into a bin, and the cost increases only when a new bin is opened. In the DP models for

<sup>6</sup> <https://lopez-ibanez.eu/tsptw-instances>

<sup>7</sup> <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/sop/>

(a) Problem types where LNBS has lower mean gap: TSPTW, CVRP, m-PDTSP,  $1||\sum w_i T_i$ , and Talent.

(b) Problem types where CABS has lower mean gap: SALBP-1, BPP, MOSP, and GCP.

**Figure 3** Entropy of the cost distribution over partial paths vs. the solution length in each problem instance. ‘LNBS better’ means LNBS finds a better solution, ‘TIE’ means that LNBS and CABS achieve the same solution cost, and ‘CABS better’ means that CABS finds a better solution.

MOSP and GCP, the cost is computed by taking the maximum weights of edges in a path, and it does not increase unless a new edge has a higher weight than the current maximum. Therefore, we expect that many partial paths tend to have the same cost in these problems, making it difficult to improve a solution by searching only a partial state space graph.

Based on these observations, we hypothesize that LNBS tends to perform better than CABS when path costs in a partial state space graph are diverse. To test this hypothesis, we evaluate the diversity of costs in a partial state space graph using entropy in information theory. Given a solution for a DyPDL model  $x = \langle x_1, \dots, x_n \rangle$ , let  $Y_{di}(x)$  be the set of solution paths whose prefix is  $\langle x_1, \dots, x_{i-1} \rangle$  and the suffix is  $\langle x_{i+d}, \dots, x_n \rangle$ . Let  $C = \{\text{cost}_y(S^0) \mid y \in Y_{di}(x)\}$  be the set of the path costs. Then, the entropy of the path costs is defined as follows:

$$H(Y_{di}(x)) = - \sum_{c \in C} \frac{|\{y \in Y_{di}(x) \mid \text{cost}_y(S^0) = c\}|}{|Y_{di}(x)|} \log_2 \frac{|\{y \in Y_{di}(x) \mid \text{cost}_y(S^0) = c\}|}{|Y_{di}(x)|}. \quad (10)$$

As this value gets larger, the cost distribution becomes more diverse, and we expect that LNBS will perform better than CABS. However, even if the entropy is large, if the problem itself is easy, both CABS and LNBS will find optimal or near-optimal solutions. To consider such cases, we also evaluate the length of the initial solution found by CABS.

We evaluate entropy and the length of the initial solution found for each problem instance used in Section 4.2. We first run CABS until it finds a feasible solution and record the length of the solution. Then, we remove the first eight transitions from the solution and enumerate

all feasible prefixes of the solution, i.e., we use  $d = 8$  and  $i = 1$ . In Figure 3, we show a scatter plot of entropy and the solution length divided into two plots to emphasize the differences between the problem types where LNBS has a lower primal gap on average (Figure 3a) and those where CABS is better (Figure 3b). With low entropy, CABS dominates. For higher entropy, the solution length begins to play a factor: for short solutions, CABS and LNBS perform equally but for longer solutions, LNBS tends to perform better. Indeed, for the problems where CABS performs better on average (Figure 3b), the entropy is quite low (less than 3.5). This result suggests that the entropy of the cost distribution over partial paths is related to the performance of LNBS. Since this analysis is based on path costs, it is not directly applicable to LNS with tree search, where a solution corresponds to a leaf node. However, if we consider the factors of neighborhood size and cost distribution over leaf nodes, we may be able to apply this analysis to LNS for CP and MIP.

## 5 Related Work

As we discussed, LNBS can be considered a generalization of DD-LNS [18], which combines DDs and LNS. DP is closely related to DDs [23], and DDs have been actively used for combinatorial optimization [9]. For example, DDs are used to obtain bounds on the optimal objective value [4, 33], and heuristics based on DDs have been proposed [6]. Moreover, ddo, a general-purpose DD solver for combinatorial optimization has been developed [5, 19]. In CP, DDs are used for constraint propagation [1, 22]. Recently, HADDOCK, a modeling language of a DD based on a state transition system, was proposed for CP [17].

In state space search, there exist several methods that improve a solution path by searching in a partial state space graph, but they were not framed as LNS. In classical planning, plan neighborhood graph search (PNGS) first constructs a partial state space graph by performing local search from each state in a solution path and then finds the shortest path in the graph [30]. In sliding tile puzzles, iterative tunneling search with A\* (ITSA\*) iteratively expands a partial state space graph, which includes states close to a given path, and finds the shortest path in that graph [13]. Unlike the above two algorithms, Joint and local path A\* (LPA\*) [32] try to find a better partial path between two states in a given path using A\* [20]. While Joint and LPA\* fix the length of a partial path to remove and deterministically select a neighborhood, LNBS dynamically adjusts them and uses beam search instead of A\*.

## 6 Conclusion

We proposed large neighborhood beam search (LNBS), a state space search algorithm based on large neighborhood search (LNS) and beam search for domain-independent dynamic programming (DIDP). Our configuration of LNBS exploits the multi-armed bandit problem and random sampling to select a neighborhood. We proved that LNBS is complete. LNBS finds better quality solutions on average than the state-of-the-art DIDP solver, complete anytime beam search (CABS), in five out of the nine benchmark problems. In particular, LNBS performs well in routing and scheduling problems, and our analysis suggests that this performance is related to the diversity of the cost distribution over partial paths. A deeper investigation of the characteristics of the problems that make LNS effective in state space search and tree search is an interesting direction for future work. Based on such analysis, developing better configurations for LNBS may also be possible.

---

**References**

---

- 1 H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In *Principles and Practice of Constraint Programming – CP 2007*, pages 118–132, 2007. doi:10.1007/978-3-540-74970-7\_11.
- 2 Norbert Ascheuer. *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*. PhD thesis, Technische Universität Berlin, 1995.
- 3 Norbert Ascheuer, Michael Jünger, and Gerhard Reinelt. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraint. *Computational Optimization and Applications*, 17:25–42, 2000. doi:10.1023/A:1008779125567.
- 4 David Bergman, Andre A. Cire, Willem-Jan van Hoesve, and J. N. Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2014. doi:10.1287/ijoc.2013.0561.
- 5 David Bergman, Andre A. Cire, Willem Jan Van Hoesve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, December 2016. doi:10.1287/ijoc.2015.0648.
- 6 David Bergman, Andre A. Cire, Willem-Jan van Hoesve, and Tallys Yunes. Bdd-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234, 2014. doi:10.1007/s10732-014-9238-1.
- 7 Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41:611–614, 2013. doi:10.1016/j.orl.2013.08.007.
- 8 Marco Antonio Moreira De Carvalho and Nei Yoshihiro Soma. A breadth-first search applied to the minimization of the open stacks. *Journal of the Operational Research Society*, 66:936–946, June 2015. doi:10.1057/jors.2014.60.
- 9 Margarita P. Castro, Andre A. Cire, and J. Christopher Beck. Decision diagrams for discrete optimization: A survey of recent advances. *INFORMS Journal on Computing*, 34(4):2271–2295, 2022. doi:10.1287/ijoc.2022.1170.
- 10 Yvan Dumas, Jacques Desrosiers, Eric Gelinat, and Marius M Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2):367–371, 1995. doi:10.1287/opre.43.2.367.
- 11 Stefan Edelkamp, Shahid Jabbar, and Alberto Lluch Lafuente. Cost-algebraic heuristic search. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, pages 1362–1367, 2005.
- 12 Rafael de Magalhães Dias Frinhan, Marco Antonio Moreira de Carvalho, and Nei Yoshihiro Soma. A pagerank-based heuristic for the minimization of open stacks problem. *PLoS ONE*, 13(8):1–24, 2018. doi:10.1371/journal.pone.0203076.
- 13 David A Furcy. ITSA\*: Iterative tunneling search with A\*. In *Proceedings of AAAI Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications*, pages 21–26, 2006.
- 14 Maria Garcia de la Banda and Peter J. Stuckey. Dynamic programming to minimize the maximum number of open stacks. *INFORMS Journal on Computing*, 19(4):607–617, 2007. doi:10.1287/ijoc.1060.0205.
- 15 Maria Garcia de la Banda, Peter J. Stuckey, and Geoffrey Chu. Solving talent scheduling with dynamic programming. *INFORMS Journal on Computing*, 23(1):120–137, 2011. doi:10.1287/ijoc.1090.0378.
- 16 Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–346, 1998. doi:10.1287/opre.46.3.330.
- 17 Rebecca Gentzel, Laurent Michel, and W.-J. van Hoesve. HADDOCK: A language and architecture for decision diagram compilation. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming – CP 2020*, pages 531–547, 2020. doi:10.1007/978-3-030-58475-7\_31.

- 18 Xavier Gillard and Pierre Schaus. Large neighborhood search with decision diagrams. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 4754–4760, 2022. doi:10.24963/ijcai.2022/659.
- 19 Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a generic and efficient framework for mdd-based optimization. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 5243–5245, 2020. doi:10.24963/ijcai.2020/757.
- 20 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.
- 21 Hipólito Hernández-Pérez and Juan José Salazar-González. The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196:987–995, August 2009. doi:10.1016/j.ejor.2008.05.009.
- 22 Samid Hoda, Willem-Jan van Hoes, and J. N. Hooker. A systematic approach to MDD-based constraint programming. In *Principles and Practice of Constraint Programming – CP 2010*, pages 266–280, 2010.
- 23 John N. Hooker. Decision diagrams and dynamic programming. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 94–110, 2013.
- 24 Siddhartha Jain and Pascal Van Hentenryck. Large neighborhood search for dial-a-ride problems. In *Principles and Practice of Constraint Programming – CP 2011*, pages 400–413, 2011.
- 25 Ryo Kuroiwa and J. Christopher Beck. Domain-independent dynamic programming: Generic state space search for combinatorial optimization. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2023. doi:10.1609/icaps.v33i1.27200.
- 26 Ryo Kuroiwa and J. Christopher Beck. Solving domain-independent dynamic programming problems with anytime heuristic search. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2023. doi:10.1609/icaps.v33i1.27201.
- 27 Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250, 2018. doi:10.1007/s10601-018-9281-x.
- 28 Shu Lin, Na Meng, and Wenxin Li. Optimizing constraint solving via dynamic programming. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 1146–1154, July 2019. doi:10.24963/ijcai.2019/160.
- 29 Michael Morin, Margarita P. Castro, Kyle E.C. Booth, Tony T. Tran, Chang Liu, and J. Christopher Beck. Intruder alert! Optimization models for solving the mobile robot graph-clear problem. *Constraints*, 23(3):335–354, 2018. doi:10.1007/s10601-018-9288-3.
- 30 Hootan Nakhost and Martin Müller. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 121–128, 2010. doi:10.1609/icaps.v20i1.13402.
- 31 Jeffrey W. Ohlmann and Barrett W. Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 19(1):80–90, 2007. doi:10.1287/ijoc.1050.0145.
- 32 Daniel Ratner and Ira Pohl. Joint and LPA\*: Combination of approximation and search. In *Proceedings of the fifth National Conference on Artificial Intelligence (AAAI)*., pages 173–177, 1986. doi:10.5555/2887770.2887798.
- 33 Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau. Peel-And-Bound: Generating Stronger Relaxed Bounds with Multivalued Decision Diagrams. In *Principles and Practice of Constraint Programming – CP 2022*, pages 35:1–35:20, 2022. doi:10.4230/LIPIcs.CP.2022.35.

- 34 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming – CP98*, volume 1520, pages 417–431, 1998. doi:10.1007/3-540-49481-2\_30.
- 35 Yingce Xia, Xu-Dong Zhang, Nenghai Yu, Geoffrey Holmes, and Yan Liu. Budgeted bandit problems with continuous random costs. In *Proceedings of the Seventh Asian Conference on Machine Learning*, pages 317–332, 2015.
- 36 Weixiong Zhang. Complete anytime beam search. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI-98/IAAI-98)*, pages 425–430, 1998.

## A Additional Experimental Results

■ **Table 2** Comparison of CP, MIP, and two configurations of LNBS. “#” is the number of optimally solved instances, “gap” is the average primal gap at the time limit, and “p.i.” is the average primal integral.

	CP			MIP			LNBS			LNBS/bias		
	#	gap	p.i.	#	gap.	p.i.	#	gap.	p.i.	#	gap.	p.i.
TSPTW (340)	46	0.0275	52.3	227	0.2268	484.1	241	<b>0.0016</b>	<b>5.7</b>	<b>242</b>	0.0019	6.2
CVRP (207)	0	0.3174	601.1	<b>26</b>	0.5845	1157.4	6	0.1640	316.8	6	<b>0.1633</b>	<b>314.8</b>
m-PDTSP (1178)	<b>1050</b>	0.0121	25.4	945	0.0858	180.0	1029	0.0022	5.0	1029	<b>0.0021</b>	<b>4.7</b>
$1  \sum w_i T_i$ (375)	150	<b>0.0003</b>	<b>2.4</b>	109	0.0188	75.6	<b>275</b>	0.0051	13.0	<b>275</b>	0.0056	14.5
Talent (1000)	7	0.0072	27.6	0	0.0573	152.6	<b>232</b>	<b>0.0041</b>	<b>11.1</b>	231	0.0042	<b>11.1</b>
SALBP-1 (2100)	1584	0.0046	28.4	1357	0.3447	634.6	<b>1682</b>	<b>0.0022</b>	<b>7.3</b>	1675	<b>0.0022</b>	7.5
BPP (1615)	<b>1234</b>	<b>0.0014</b>	<b>7.7</b>	1157	0.0385	85.9	1139	0.0021	8.1	1129	0.0021	9.0
MOSP (570)	437	0.0044	13.0	224	0.0394	100.4	<b>523</b>	<b>0.0002</b>	<b>0.7</b>	<b>523</b>	<b>0.0002</b>	<b>0.7</b>
GCP (135)	1	0.0151	44.3	23	0.1102	311.9	<b>102</b>	<b>0.0001</b>	<b>0.6</b>	<b>102</b>	<b>0.0001</b>	0.7
Larger Instances												
m-PDTSP (240)	77	0.1481	284.1	47	0.5811	1096.8	<b>98</b>	0.0652	<b>146.6</b>	97	<b>0.0647</b>	147.0
MOSP (760)	0	0.0675	150.4	0	0.8806	1599.4	<b>148</b>	<b>0.0025</b>	<b>10.4</b>	<b>148</b>	0.0027	10.7
GCP (50)	<b>0</b>	0.5287	1268.1	<b>0</b>	0.5306	977.8	<b>0</b>	<b>0.0038</b>	<b>19.5</b>	<b>0</b>	0.0061	21.2

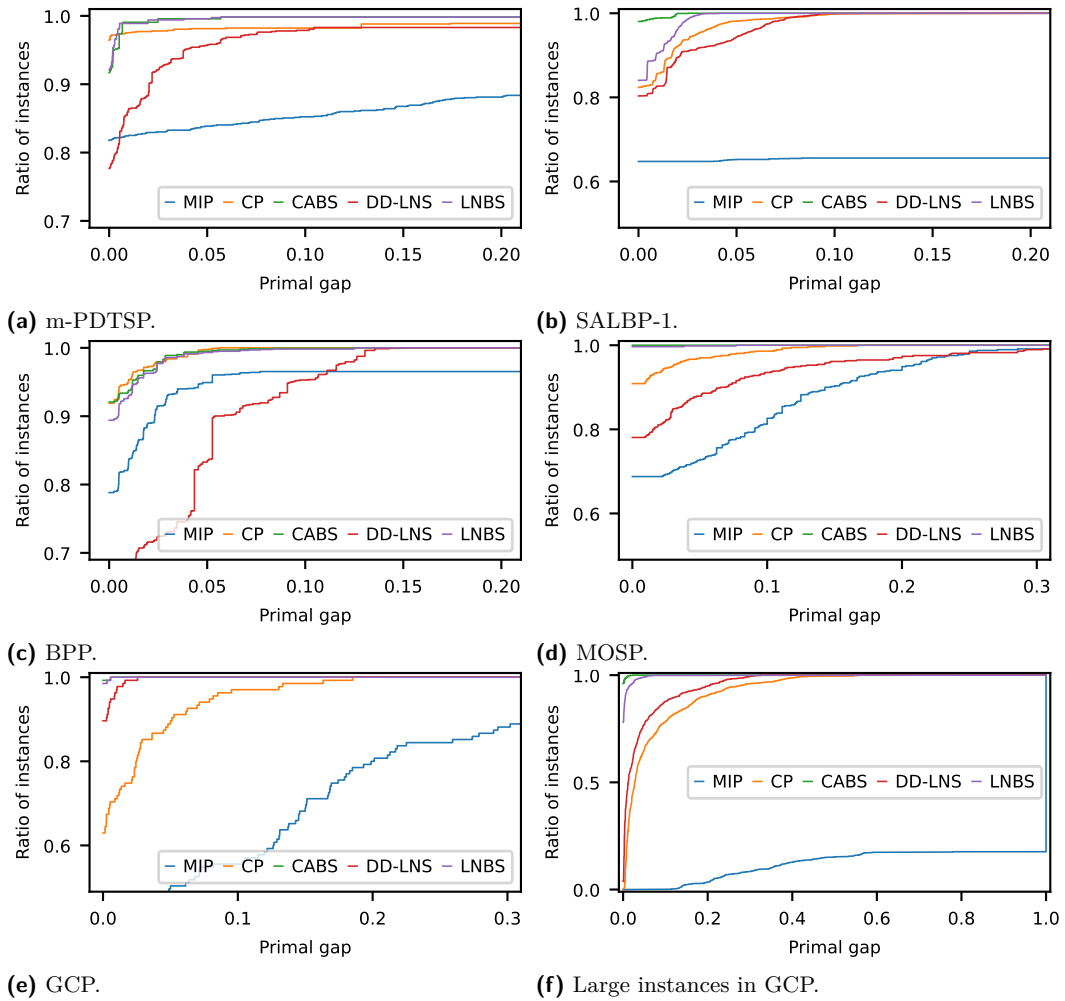
■ **Table 3** Comparison of LNBS variants. “No removing conflicts” does not remove conflicting transitions in the suffix. “No Budgeted-UCB” selects the depth uniformly at random instead of using Budgeted-UCB. “#” is the number of optimally solved instances, “gap” is the average primal gap at the time limit, and “p.i.” is the average primal integral.

	LNBS			No removing conflicts			No Budgeted-UCB		
	#	gap.	p.i.	#	gap.	p.i.	#	gap.	p.i.
TSPTW (340)	241	<b>0.0016</b>	<b>5.7</b>	234	0.0035	10.6	<b>256</b>	0.0034	9.4
CVRP (207)	<b>6</b>	<b>0.1640</b>	<b>316.8</b>	5	0.1682	322.5	<b>6</b>	0.1767	338.7
$1  \sum w_i T_i$ (375)	275	<b>0.0051</b>	<b>13.0</b>	268	0.0224	56.0	<b>287</b>	0.0340	74.1

We show the result of MIP in Table 2. MIP solves more instances than CP in TSPTW, CVRP, and GCP, but CP is better in the primal gap and the primal integral.

In addition, we compare two configurations of LNBS in Table 2. One selects the starting point of a partial path uniformly at random (LNBS), and another selects it according to the probability distribution biased by partial path costs in Equation (7) (LNBS/bias). LNBS/bias solves one more instance in TSPTW and outperforms LNBS in CVRP and m-PDTSP in the primal gap.

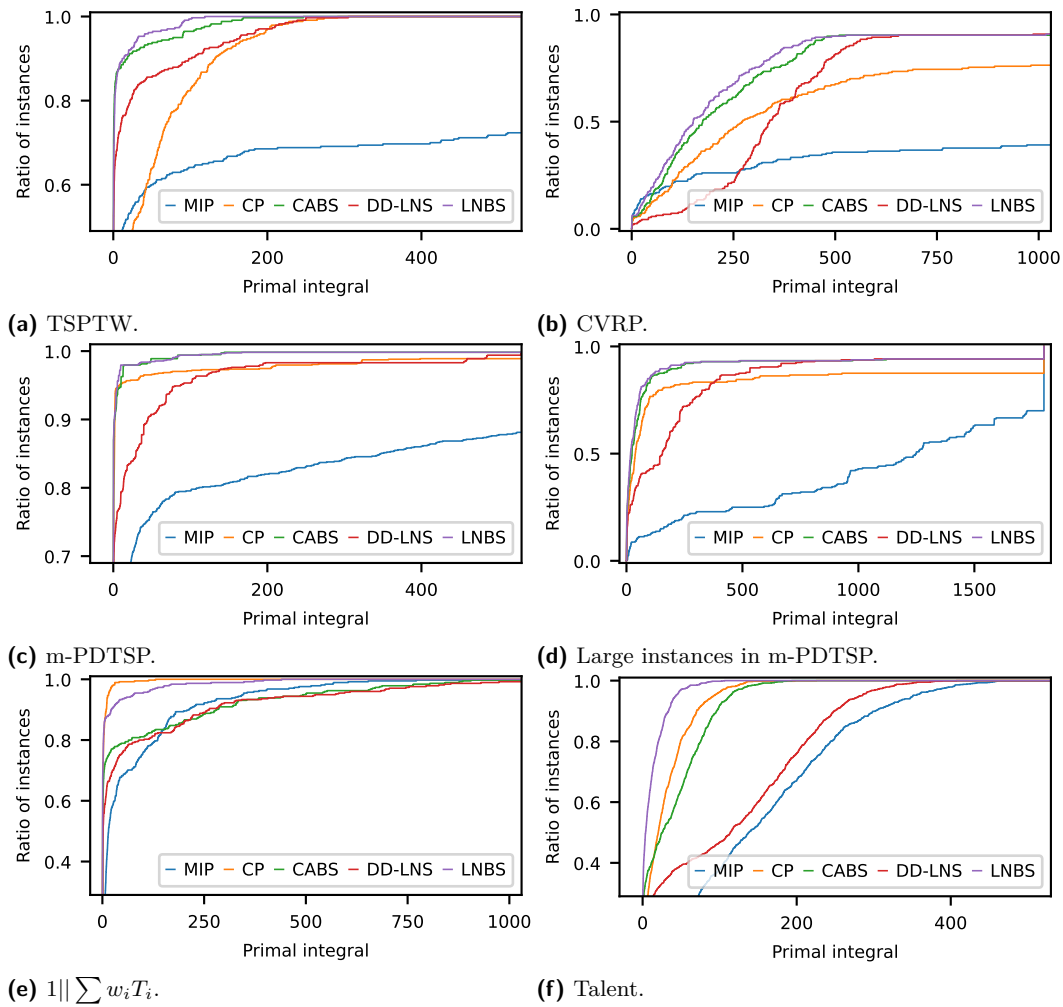




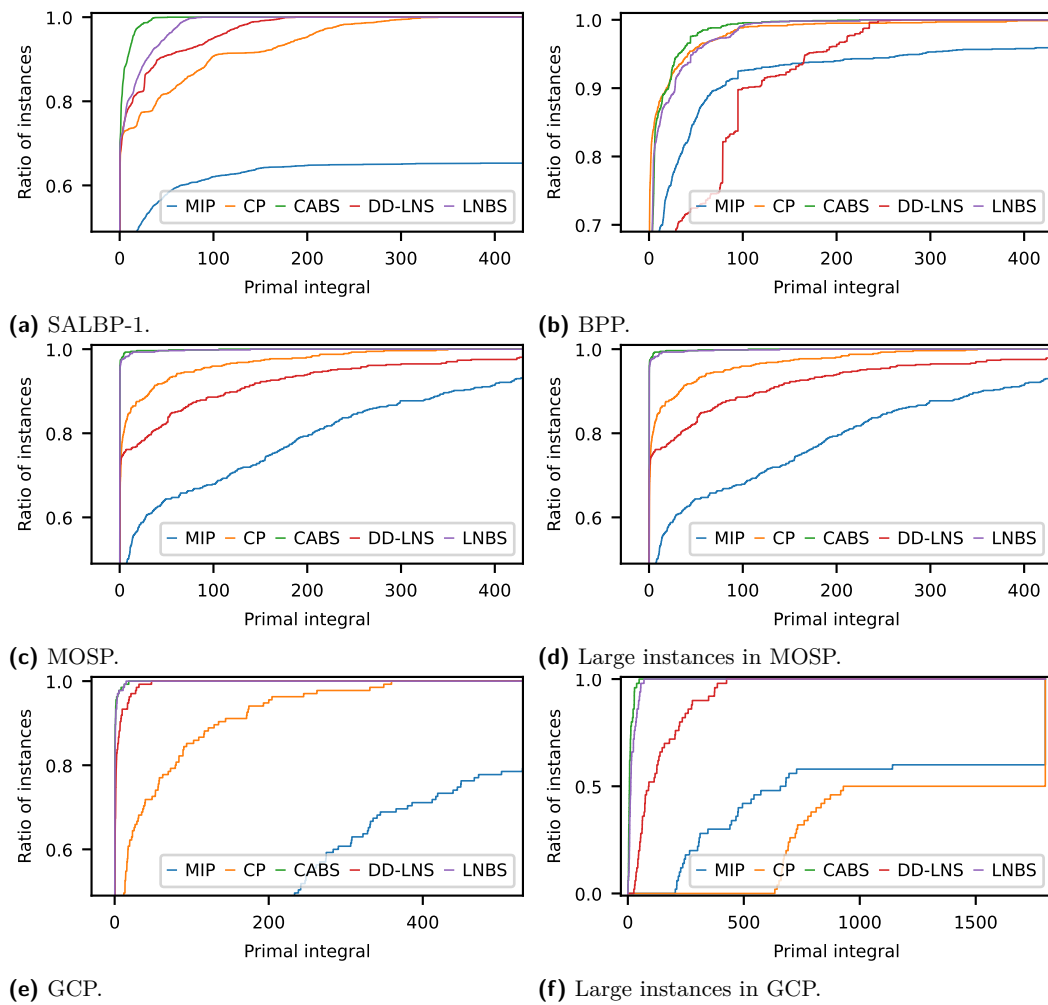
■ **Figure 4** Distribution of the primal gap at the time limit. Higher and left is better.

We evaluate the importance of other components of LNBS using a subset of the problems: TSPTW, CVRP, and  $1||\sum w_i T_i$ . In Table 3, we compare two variants of LNBS, where conflicting transitions in a suffix are not removed (‘No removing conflicts’), and the depth is selected uniformly at random instead of Budgeted-UCB (‘No Budgeted-UCB’). The two variants perform worse in terms of the primal gap and the primal integral. While ‘No Budgeted-UCB’ solves more instances to optimality, it is because the largest depth, which makes LNBS the same as CABS, is more likely to be selected by uniform sampling. Indeed, the primal gap and the primal integral of ‘No Budgeted-UCB’ are close to those of CABS.

In Figure 4, we present the distribution of the primal gap over instances in m-PDTSP, SALBP-1, BPP, MOSP, Graph-Clear, and large instances in GCP. LNBS is slightly better in m-PDTSP, and CABS is better in SALBP-1, BPP, and large instances in GCP. Figures 5 and 6 show the distribution of the primal integral. The tendency is similar to that of the primal gap.



■ **Figure 5** Distribution of the primal integral at the time limit in the problems where LNBS is better. Higher and left is better.



■ **Figure 6** Distribution of the primal integral at the time limit in the problems where CABS is better. Higher and left is better.

# MDD Archive for Boosting the Pareto Constraint

**Steve Malalel**

Université Côte d'Azur, CNRS, I3S, Nice, France

**Arnaud Malapert**

Université Côte d'Azur, CNRS, I3S, Nice, France

**Marie Pelleau**

Université Côte d'Azur, CNRS, I3S, Nice, France

**Jean-Charles Régim**

Université Côte d'Azur, CNRS, I3S, Nice, France

---

## Abstract

Multi-objective problems are frequent in the real world. In general they involve several incomparable objectives and the goal is to find a set of Pareto optimal solutions, i.e. solutions that are incomparable two by two. In order to better deal with these problems in CP the global constraint PARETO was developed by Schaus and Hartert to handle the relations between the objective variables and the current set of Pareto optimal solutions, called the archive. This constraint handles three operations: adding a new solution to the archive, removing solutions from the archive that are dominated by a new solution, and reducing the bounds of the objective variables. The complexity of these operations depends on the size of the archive. In this paper, we propose to use a multi-valued Decision Diagram (MDD) to represent the archive of Pareto optimal solutions. MDDs are a compressed representation of solution sets, which allows us to obtain a compressed and therefore smaller archive. We introduce several algorithms to implement the above operations on compressed archives with a complexity depending on the size of the archive. We show experimentally on bin packing and multi-knapsack problems the validity of our approach.

**2012 ACM Subject Classification** Applied computing → Multi-criterion optimization and decision-making; Theory of computation → Constraint and logic programming; Mathematics of computing → Decision diagrams

**Keywords and phrases** Constraint Programming, Global Constraint, MDD, Multi-Objective Problem, Pareto Constraint

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.24

**Funding** This work has been supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

## 1 Introduction

Multi-objective combinatorial optimization (MOCO) problems are present in many industrial applications [13, 14]. They involve several incomparable objectives represented by objective variables.

For the sake of clarity and without loss of generality, we will consider that we have to solve a problem where all objective variables must be minimized.

A solution  $S_1$  of objective variables is dominated by another solution  $S_2$  if for each objective variable  $obj_i$  the value of  $obj_i$  in  $S_2$  is better than or equal to the value of  $obj_i$  in  $S_1$ . For instance the solution (4, 6, 3, 1) is dominated by (4, 3, 2, 1) but it is not dominated by (1, 1, 1, 3). The set of non dominated solutions defines the set of Pareto optimal solutions. In MOCO, the goal is to compute that set of Pareto optimal solutions. Usually the set of non



© Steve Malalel, Arnaud Malapert, Marie Pelleau, and Jean-Charles Régim;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 24; pp. 24:1–24:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

dominated solutions are saved in an archive that is maintained during the search for solutions. Two operations are involved: *insert* that manages the addition of a new non dominated solution and *delete* that removes the solutions that are dominated by the new solution.

In addition, solvers dealing with MOCO problems have to deal with another question: Is a new solution dominated by an archive solution?

In Constraint Programming, we answer this question by avoiding the generation of dominated solutions. To do this, we add a constraint to the problem that ensures that no dominated solution can be computed. This added constraint is called Pareto constraint and was proposed by Schaus and Hartert [11, 4]. It implements the ideas of Gavanelli [3]. This constraint reduces the bounds of the objective variables such that dominated solutions cannot be produced. This result is obtained by preventing a new solution from being dominated by a solution from the archive. In order to understand this process, let us create a tuple composed of the current minimum of all objective variables. This tuple will dominate all solutions that can be constructed from the current objective variables. Thus, if there is a solution in the archive that dominates this tuple then clearly it will dominate all future solutions and so we can stop the search. Now, consider the objective variable  $obj_i$ . If we replace in our tuple the value of  $obj_i$  by its maximum possible value and if we found  $S$ , a solution of the archive, dominating this tuple, then  $obj_i$  must take a value less than or equal to that of  $S$  otherwise the future solutions will be dominated by  $S$ . By applying this process for each variable, Schaus and Hartert establish the bound consistency of the constraint. We will denote by *filter* this process. The complexity of this operation is not detailed in their paper. A simple implementation will require to traverse  $n$  (the number of objectives) times the archive. It is not straightforward to reach a time complexity linear in the size of the archive.

It may be tempting to use multi-valued decision diagrams (MDDs) for this constraint because MDDs are a compressive data structure for representing solution sets. Perez [7] defined the MDD representing the Pareto constraint, i.e. the set of tuples allowed by the constraint that are the possible future non-dominated solutions. As mentioned by Perez, this approach failed mainly because at the beginning the MDD compresses very strongly the set of solutions since everything is almost possible, then it will decompress because we only delete tuples and the chances of recompression are low.

In this article we propose to use MDDs to represent the archive, that are the current non dominated solutions. Currently the archive is often represented as lists. We can also use quad-trees but this is only efficient if we have few objectives [6], which is not our case of study. With lists (or quad-trees for that matter), the complexity of the operations *insert* and *delete* is linear with the size of the archive (i.e. the number of elements multiplied by the number of objectives). Representing the archive by an MDD will allow parts of common solutions to be merged. As with an MDD, all the solutions are treated globally, we can therefore hope to save time thanks to these groupings.

The operation *delete* will therefore potentially save time. The operation *insert* may be slower because MDDs are a heavier data structure than lists. However, this operation takes much less time than *delete* or *filter*. For this last operation, we will benefit from the global view of the MDD. We propose to improve the algorithm of Schaus and Hartet in two ways: we define an algorithm to find the tightest solutions (i.e. the largest possible value of an objective) faster and we introduce a variant of this algorithm that processes all objective variables at once, and not successively.

Our algorithms are based on the following idea: Consider the objective variable  $obj_i$ . Let us remove from the MDD that represents the archive all the values of the objective variables different from  $obj_i$  that are less than or equal to the minimum of their variable. Then we

perform a reduction of the MDD in order to obtain  $MDD_D$ . If  $MDD_D$  is not empty then it means that there exist paths from root to tt in the MDD, that is solutions in the archive which will dominate any new solution involving some values of  $obj_i$ . More precisely, there exist solutions of the form  $(v_1, v_2, \dots, v_i, \dots, v_n)$  such that  $\forall j = 1 \dots n, j \neq i : v_j \leq \min(obj_j)$ . These solutions dominate (or are equal to) any future solution with  $obj_i \geq v_i$ . Hence, the maximum possible value for  $obj_i$  is the smallest value of  $obj_i$  in  $MDD_D$ . The obtained algorithms have a linear time complexity in the size of the MDD, which improves the algorithm of Schaus and Hartet.

The paper is organized as follows. First, we recall some concepts and definitions of Constraint Programming, Multi-Objective Optimization Problems and Multi-valued Decision Diagrams. Then, we introduce the representation of the archive by an MDD. We present how the *insert* and *delete* operations are implemented. We detail two algorithms for the operation *filter* that establishing the bound consistency of the Pareto Constraint associated with an MDD. Next, we experiment with these methods on bin packing and multi-knapsack problems. At last, we conclude.

## 2 Preliminaries

### 2.1 Constraint Programming

A finite constraint network  $\mathcal{N}$  is defined as a set of  $n$  variables  $X = \{x_1, \dots, x_n\}$ , a set of current domains  $\mathcal{D} = \{D(x_1), \dots, D(x_n)\}$  where  $D(x_i)$  is the finite set of possible values for variable  $x_i$ , and a set  $\mathcal{C}$  of constraints between variables. If  $x$  is a variable, then  $x^{min} = \min(D(x))$  and  $x^{max} = \max(D(x))$ . We introduce the particular notation  $\mathcal{D}_0 = \{D_0(x_1), \dots, D_0(x_n)\}$  to represent the set of initial domains of  $\mathcal{N}$  on which constraint definitions were stated. An element of  $D_0(x_1) \times \dots \times D_0(x_n)$  on the ordered set  $\mathcal{D}$  is called a tuple and is denoted  $\tau$ . In a tuple  $\tau$ , the assignment of the  $i^{th}$  variable is denoted  $\tau_i$ . A solution of  $\mathcal{N}$  is a tuple  $\tau$  that satisfies all the constraints in  $\mathcal{C}$ . A constraint  $C$  on the ordered set of variables  $X(C) = (x_{i_1}, \dots, x_{i_r})$  is a subset  $T(C)$  of the Cartesian product  $D_0(x_{i_1}) \times \dots \times D_0(x_{i_r})$  that specifies the allowed combinations of values for the variables  $x_{i_1}, \dots, x_{i_r}$ .

### 2.2 Multi-Objective Optimization

A multi-objective problem in combinatorial optimization is a problem where several objectives have to be improved while satisfying constraints. For the sake of clarity and without loss of generality, these objectives are represented by integer variables and have to be minimized. We denote by  $O = (obj_1, \dots, obj_m)$  the ordered set of the objective variables in  $X$ , the set of variables of the whole problem. Then, this problem can be modeled as follows:

$$\begin{array}{ll} \text{Minimize} & O \\ \text{Subject to} & \mathcal{C} \end{array} \quad (1)$$

However, the minimization of several objectives simultaneously may seem ambiguous. Indeed, there is no order of priority between the different objectives and improving one often means degrading at least one of the others. This generally introduces the need to make compromises during the solving process: the goal is not to find only one optimal solution but a set of solutions that are considered *Pareto optimal*.

In the rest of the paper, we will focus only on the objective variables. As a consequence, a tuple will always be understood only in relation to the set  $O$ , and the same applies to a solution. Thus, for a tuple  $\tau$ , the assignment of the  $i^{th}$  objective variable is denoted by  $\tau_i$ .

## 24:4 MDD Archive for Boosting the Pareto Constraint

The following definitions are taken from [11]:

► **Definition 1** (Pareto dominance). Let  $\tau$  and  $\tau'$  be two solutions of a multi-objective problem represented by a constraint network  $\mathcal{N}$ .

We say that  $\tau$  dominates  $\tau'$ , denoted  $\tau \prec \tau'$ , if and only if:

$$\begin{aligned} & \forall i \in [1 \dots m] : \tau_i \leq \tau'_i \\ \wedge & \exists i \in [1 \dots m] : \tau_i < \tau'_i \end{aligned} \quad (2)$$

We say that  $\tau$  weakly-dominates  $\tau'$ , denoted  $\tau \preceq \tau'$ , if and only if  $\forall i \in [1 \dots m] : \tau_i \leq \tau'_i$ .

► **Definition 2** (Pareto optimality). Let  $\mathcal{S}$  be the set of all the feasible solutions of a multi-objective problem represented by a constraint network  $\mathcal{N}$ . A solution  $\tau$  is Pareto optimal if and only if there is no solution  $\tau'$  in  $\mathcal{S}$  that dominates  $\tau$ :

$$\nexists \tau' \in \mathcal{S} : \tau' \prec \tau \quad (3)$$

► **Definition 3** (Pareto set). Let  $\mathcal{N}$  be the constraint network representing a multi-objective problem and  $\mathcal{S}$  be the set of all the feasible solutions of  $\mathcal{N}$ . The Pareto set of  $\mathcal{N}$  is the set of all the Pareto optimal solutions in  $\mathcal{S}$ :

$$\{\tau \in \mathcal{S} \mid \nexists \tau' \in \mathcal{S} : \tau' \prec \tau\} \quad (4)$$

The search for the exact Pareto set of a multi-objective problem can be impossible to achieve in a reasonable time. This leads to search for an approximation of the Pareto set: the *archive*.

► **Definition 4** (Archive). An archive  $\mathcal{A}$  is a set of solutions such that there is no solution  $\tau'$  in the archive that dominates another solution  $\tau$  in the archive. This property is known as the *domination-free property*:

$$\tau \in \mathcal{A}, \nexists \tau' \in \mathcal{A} : \tau' \prec \tau \quad (5)$$

A basic way to maintain an archive  $\mathcal{A}$  is to verify if a solution  $\tau$  found during the search is dominated by a solution of the archive:

- If  $\tau$  is dominated by at least one solution, do not add it in  $\mathcal{A}$ .
- If  $\tau$  is not dominated by any solution, add it and remove from  $\mathcal{A}$  all the solutions dominated by  $\tau$ .

### 2.3 Pareto Constraint

We reformulate the definition introduced in [11] in order to avoid the notion of “next discovered solution”.

► **Definition 5** (Pareto Constraint). Let  $X$  be a set of objective variables and  $\mathcal{A}$  be an archive defined on  $O$ . A PARETO constraint is a constraint  $C$  associated with  $\mathcal{A}$  defined by  $\text{PARETO}(O, \mathcal{A}) = \{\tau \text{ s.t. } \tau \text{ is a tuple on } O \text{ and } \nexists \tau' \in \mathcal{A} \text{ with } \tau' \preceq \tau\}$ .

When using this constraint during the search for solutions, newly found solutions must be inserted in the archive. One could notice that this constraint prevents finding a solution  $\tau$  such that  $\tau' \in \mathcal{A}$  and  $\tau = \tau'$ .

$$\begin{array}{l}
D(obj_1) = \{2, 4\} \\
D(obj_2) = \{1, 2, 5\} \\
D(obj_3) = \{1, 3, 4, 5\} \\
D(obj_4) = \{2, 5, 6\}
\end{array}
\wedge
\begin{array}{c}
\mathcal{A} \\
\boxed{\begin{array}{c} (3, 1, 3, 1) \\ (2, 1, 4, 2) \end{array}}
\end{array}
\Rightarrow obj_3 < 4$$

■ **Figure 1** Application of the propagator of the PARETO constraint.

We adapt the definition of ideal point of multi-objective problems to our purpose:

- **Definition 6** (Ideal tuple). Let  $C = \text{PARETO}(O, \mathcal{A})$  be a Pareto constraint with  $O = (obj_1, \dots, obj_n)$ .
- The ideal tuple of  $C$  denoted by  $\tau^*(O)$  is the tuple composed of the best objective values, that is  $(obj_1^{\min}, \dots, obj_n^{\min})$ .
  - The ideal tuple for the value  $a$  of the variable  $obj_i$  denoted by  $\tau^*(O, i, a)$  is the tuple composed of  $obj_i = a$  and the best objective values for the other objective variables, that is  $(obj_1^{\min}, \dots, obj_{i-1}^{\min}, a, obj_{i+1}^{\min}, \dots, obj_n^{\min})$ .

► **Proposition 7.** Let  $C = \text{PARETO}(O, \mathcal{A})$  be a Pareto constraint; the following two properties are equivalent:

- $C$  is consistent;
- $\tau^*(O)$  is not weakly-dominated by any tuple of  $\mathcal{A}$

**Proof.** By definition  $\tau^*(O)$  weakly-dominates any tuple defined on  $O$ , thus if  $\tau^*(O)$  is not weakly-dominated then it is a possible solution and the constraint is consistent. Otherwise, there is no solution and  $C$  is not consistent. ◀

We reformulate the filtering algorithm associated with the Pareto constraint given in [11].

► **Proposition 8.** Let  $C = \text{PARETO}(O, \mathcal{A})$  be a Pareto constraint. The value  $a$  of the objective variable  $obj_i$  is not consistent with  $C$  if and only if  $\exists \tau \in \mathcal{A}$  such that  $\tau \preceq \tau^*(O, i, a)$ .

**Proof.**  $\Rightarrow$  If the value  $a$  of  $obj_i$  is not consistent with  $C$  then every tuple  $\tau$  of  $C$  with  $\tau_i = a$  is weakly-dominated by a tuple of  $\mathcal{A}$ . Therefore  $\tau^*(O, i, a)$  is weakly-dominated by a tuple of  $\mathcal{A}$ .

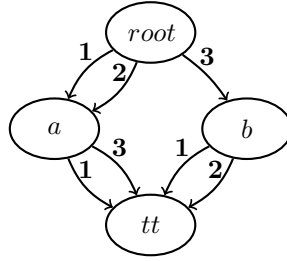
$\Leftarrow$  The tuple  $\tau^*(O, i, a)$  weakly-dominates all the possible tuples  $\tau$  of  $C$  with  $\tau_i = a$ . Thus if this tuple is weakly-dominated then there is no tuple with  $\tau_i = a$  consistent with the constraint and the value  $a$  of  $obj_i$  is not consistent with  $C$ . ◀

From this proposition we can identify all values of all variables that are inconsistent with the constraint and so we can establish the arc consistency of the constraint which is equivalent in our case to the bound consistency. Figure 1 gives an example of domain reduction.

## 2.4 Multi-valued Decision Diagram

The decision diagrams considered in this paper are reduced ordered multi-valued decision diagrams (MDD) [5, 12, 1], which are a generalization of binary decision diagrams [2]. They use a fixed variable ordering for canonical representation and shared sub-graphs for compression obtained by means of a reduction operation. An MDD is a rooted directed acyclic graph (DAG) used to represent some multi-valued functions  $f : \{0 \dots d - 1\}^n \rightarrow \text{true}, \text{false}$ .





■ **Figure 2** An MDD representing the tuple set  $\{(1, 1), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}$ .

Given the  $n$  input variables, the DAG contains  $n + 1$  layers of nodes, such that each variable is represented at a specific layer of the graph. Each node on a given layer has at most  $d$  outgoing arcs to nodes in the next layer of the graph. Each arc is labeled by its corresponding integer. The arc  $(u, a, v)$  is from node  $u$  to node  $v$  and labeled by  $a$ . Sometimes it is convenient to say that  $v$  is a child of  $u$ . The set of outgoing arcs from node  $u$  is denoted by  $\omega^+(u)$ . All outgoing arcs of the layer  $n$  reach  $tt$ , the true terminal node (the false terminal node is typically omitted). There is an equivalence between  $f(a_1, \dots, a_n) = true$  and the existence of a path from the root node to the  $tt$  whose arcs are labeled  $a_1, \dots, a_n$ . Figure 2 shows an example of MDD and the kind of compression it can offer.

The reduction of an MDD is one of the most important operations that may reduce the MDD size by an exponential factor. It consists in removing nodes that have no successor and merging equivalent nodes, i.e., nodes having the same set of neighbors associated with the same labels. This means that only nodes of the same layer can be merged. Other operations used in this paper are the addition and deletion of tuples of an MDD. They can be performed with in-place operations provided by Perez and Régis [8].

The advantage of using MDDs instead of the usual data structures is their compression capability which is useful for reducing memory consumption. Moreover, this compression may also improve the time performance of algorithms computing on a set.

### 3 Pareto Constraint Using MDD

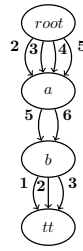
We propose to use  $MDD_{\mathcal{A}}$ , an MDD, to represent the solution archive. Consider  $\tau$  a new solution. Without loss of generality, we assume that  $\tau$  is not weakly-dominated by any tuple of the archive. As mentioned in the introduction we need to implement the operations *insert* and *delete*:

- *insert*:  $\tau$  has to be added to  $MDD_{\mathcal{A}}$ .
- *delete*: All the tuples dominated by  $\tau$  must be removed from  $MDD_{\mathcal{A}}$ .

In addition we need to design algorithms for implementing the *filter* operation of the PARETO constraint.

#### 3.1 Insert and Delete Operations

Adding  $\tau$  to  $MDD_{\mathcal{A}}$  can be done thanks to the in-place addition operation [8]. The deletion from  $MDD_{\mathcal{A}}$  of all the tuples that are dominated by  $\tau$  can be done by creating  $MDD_{dom}(\tau)$ , the MDD of all the tuples weakly-dominated by  $\tau$ .  $MDD_{dom}(\tau)$  is really simple: each layer contains only one node, and for each layer  $i \in [1 \dots (n - 1)]$  there are arcs labeled with all the values that belong to the range  $[\tau_i \dots max(D_0(obj_i))]$  between the node of layer  $i$  and the node of layer  $i + 1$ . Figure 3 shows an example of such an MDD. Then, the operation  $MDD_{\mathcal{A}}$



■ **Figure 3** An MDD representing all the tuples weakly-dominated by  $(2, 5, 1)$  with  $D_0(obj_1) = [1 \dots 5]$ ,  $D_0(obj_2) = [1 \dots 6]$  and  $D_0(obj_3) = [1 \dots 3]$ .

–  $MDD_{dom}(\tau)$  can be performed in-place thanks to the in-place difference operator [8]. It should be noted that this operation also deletes the tuple  $\tau$  from  $MDD_{\mathcal{A}}$ , so it is better to perform first the *delete* operation and then the *insert* operation.

## 3.2 Filtering algorithm of the Pareto Constraint

Let  $C = \text{PARETO}(O, MDD_{\mathcal{A}})$  be a Pareto constraint whose archive is represented by an MDD. We present two methods that eliminate all values of the variables satisfying Proposition 8 (i.e., that are not consistent with  $C$ ). That establishes the bound consistency of  $C$ . The first one has to be executed for each objective variable, and the second one uses the concept of the first method to filter all the objective variables at the same time.

### 3.2.1 Unidirectional Marking

This method operates only on one objective variable  $obj_i$  at a time. It must be repeated for each objective variable to be complete.

$MDD_{\mathcal{A}}$  is traversed using a Depth-First Search (DFS) procedure from *root* following two rules:

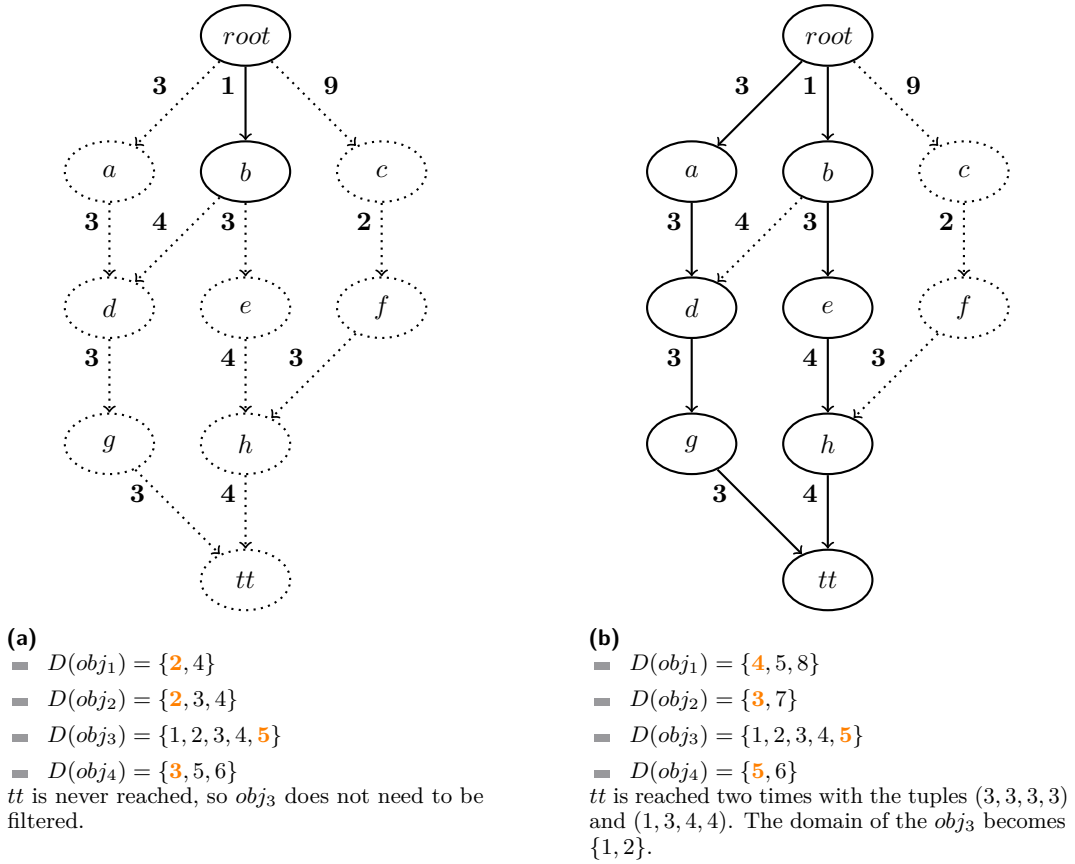
- For each layer  $j \neq i$  it is possible to go only through arcs whose values are less than or equal to  $obj_j^{min}$ .
- For the layer  $i$ , it is possible to go only through arcs whose values are less than or equal to  $obj_i^{max}$ .

Each time the node *tt* is reached, the value of the arc of the layer  $i$  belonging to the current path is memorized if it is less than the previous memorized value. Then,  $obj_i^{max}$  takes the lower value between the current upper bound and the memorized value minus one. Figure 4 shows two examples based on the same archive with different states for the domains. Algorithm 3.1 is a possible implementation of this filtering for all the objective variables.

► **Proposition 9.** *The unidirectional marking method eliminates all the values of  $obj_i$  that are not consistent (c.f. Proposition 8).*

**Proof.** The unidirectional marking method finds paths corresponding to the tuples that weakly-dominate  $\tau^*(O, i, obj_i^{max})$  and it eliminates all the values  $a$  such that  $\tau^*(O, i, a)$  is weakly-dominated, by setting the maximum of  $obj_i$  to  $minV - 1$  where  $minV = \min(\{a \text{ such that } \tau^*(O, i, a) \text{ is weakly-dominated}\})$ . ◀

The time complexity of this method for one objective variable is linear in the size of the MDD, because it traverses the MDD with a DFS. However, as it is repeated for each objective variable and the size of the MDD depends on the number of objective variables, the overall



■ **Figure 4** Application of the unidirectional marking in  $MDD_{\mathcal{A}}$  for the objective variable  $obj_3$ .  $MDD_{\mathcal{A}}$  contains the set of tuples  $\{(3, 3, 3, 3), (1, 4, 3, 3), (1, 3, 4, 4), (9, 2, 3, 4)\}$ . All the nodes and arcs reached with the unidirectional marking method are represented with plain lines while those not reached with dotted lines.

time complexity is then quadratic in the number of objectives. This method takes advantage of the compression offered by MDDs. Nonetheless, we can notice that a large part of the DFS is shared between the filtering of each objective variable, that is to say there are many repetitions. We then propose an improvement of this method that will execute only two DFSs.

### 3.2.2 Bidirectional Marking

The first step is to identify in  $MDD_{\mathcal{A}}$ , for all  $j \in (1 \dots n)$ , all the beginning of tuple  $\tau_{(1 \dots j)} = (\tau_1, \dots, \tau_j)$  such that  $\tau_{(1 \dots j)}$  weakly-dominates  $(obj_1^{min}, \dots, obj_j^{min})$ . This can be done by using a DFS from  $root$  to find the corresponding paths by following one rule: for the layer  $j$  it is possible to go only through arcs whose values are less than or equal to  $obj_j^{min}$ . All the nodes reached with this method are considered as marked from  $root$ .

► **Proposition 10.** *If  $tt$  is reached by the first step of the bidirectional marking method, then the PARETO constraint  $C$  is not consistent.*

**Proof.** If  $tt$  is reached by the first step of the bidirectional marking method it means that there exists a path corresponding to a tuple that weakly-dominates  $\tau^*(O)$ . Then  $C$  is not consistent according to Proposition 7. ◀

■ **Algorithm 3.1** unidirectional marking algorithm.

---

```

// min is passed by reference
recursiveDFS(MDDA,  $\mathcal{D}$ ,  $i$ ,  $l$ ,  $u$ ,  $path[]$ ,  $current$ ,  $min$ )
1 |  $u.isVisited \leftarrow true$ 
2 |  $path[l] \leftarrow u$ 
3 | for each  $arc(u, a, v)$  do
4 |   if  $(l = i \text{ and } a \leq obj_i^{max}) \text{ or } a \leq obj_l^{min}$  then
5 |     if  $l = i$  then  $current \leftarrow a$ 
6 |     if  $v = tt$  then
7 |       for each  $node \in path$  do  $node.reachTt \leftarrow true$ 
8 |       if  $current \leq min$  then  $min \leftarrow current$ 
9 |     else
10 |      if  $v.reachTt$  and  $l \geq i$  and  $current \leq min$  then  $min \leftarrow current$ 
11 |      if not  $v.isVisited$  then
12 |         $recursiveDFS(MDD_A, \mathcal{D}, i, l + 1, v, path, current, min)$ 

oneWayMarkingFiltering( $O$ ,  $\mathcal{D}$ , MDDA)
12 | for each  $obj_i \in O$  do
13 |   for each  $node \in MDD_A$  do
14 |      $node.isVisited \leftarrow false$ 
15 |      $node.reachTt \leftarrow false$ 
16 |    $path[] \leftarrow \emptyset$  for each index
17 |    $currentValue \leftarrow MAX\_INTEGER$ 
18 |    $minValue \leftarrow MAX\_INTEGER$ 
19 |    $recursiveDFS(MDD_A, \mathcal{D}, i, 1, root, path, currentValue, minValue)$ 
20 |    $obj_i^{max} \leftarrow \min(obj_i^{max}, minValue - 1)$ 

```

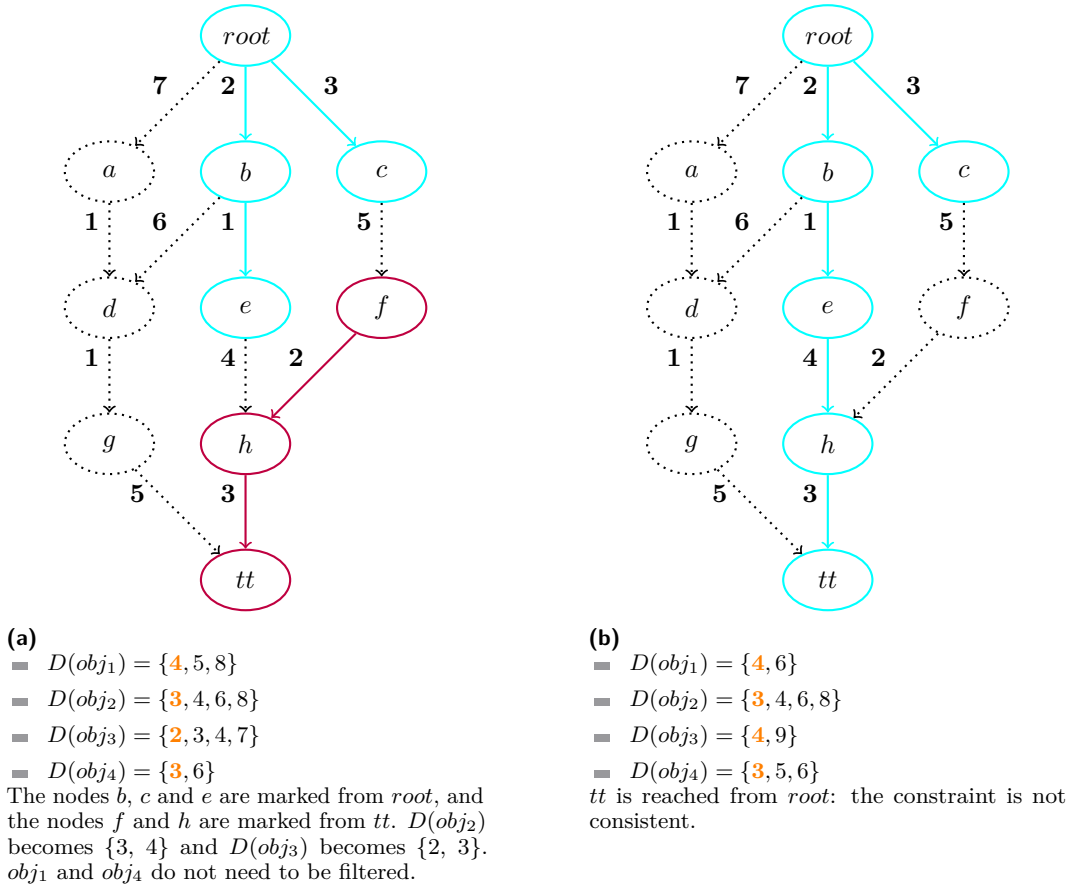
---

The second step is similar to the first one. It consists of identifying in  $MDD_A$ , for all  $j \in (1 \dots n)$ , all the end of tuple  $\tau_{(j \dots n)} = (\tau_j, \dots, \tau_n)$  such that  $\tau_{(j \dots n)}$  weakly-dominates  $(obj_j^{min}, \dots, obj_n^{min})$ . This time the DFS starts from  $tt$ , takes arcs only in reverse, and follows the same rule as for the first step. All the nodes reached during this step are considered as marked from  $tt$ .

The inconsistent edge can now be identified: these are the arcs  $(u, a, v)$  on the layer  $i$ , such that  $u$  is marked from  $root$  and  $v$  is marked from  $tt$ . More formally we have:

► **Proposition 11.** *Let  $\Lambda_i$  be the set of all the arcs  $(u, a, v)$  on the layer  $i$ , such that  $u$  is marked from  $root$  and  $v$  is marked from  $tt$ . The value  $a$  of  $obj_i$  satisfies Proposition 8 if and only if  $\exists (u, a, v) \in \Lambda_i$*

**Proof.** For all the arcs  $(u, l, v)$  in  $\Lambda_i$  there is at least one path going from  $root$  to  $u$  that weakly-dominates  $(obj_1^{min}, \dots, obj_{i-1}^{min})$ , and there is also at least one path going from  $v$  to  $tt$  that weakly-dominates  $(obj_{i+1}^{min}, \dots, obj_n^{min})$ . It means that there is at least one tuple  $\tau$  such that  $\tau_i = l$  and  $\tau \preceq \tau^*(O, i, l)$ . Conversely, if there exists one tuple  $\tau$  such that  $\tau_i = l$  and  $\tau \preceq \tau^*(O, i, l)$ , then there is at least one path going from  $root$  to  $u$  that weakly-dominates  $(obj_1^{min}, \dots, obj_{i-1}^{min})$ , and there is also at least one path going from  $v$  to  $tt$  that weakly-dominates  $(obj_{i+1}^{min}, \dots, obj_n^{min})$ . Therefore  $(u, l, v)$  belongs to  $\Lambda_i$  ◀



■ **Figure 5** Application of the bidirectional marking in  $MDD_A$ . It represents the set of tuples  $\{(7, 1, 1, 5), (2, 6, 1, 5), (2, 1, 4, 3), (3, 5, 2, 3)\}$ . All the nodes and arcs reached with the bidirectional marking method are represented with plain lines while those not reached with dotted lines.

We immediately have:

► **Proposition 12.** *The bidirectional marking method finds and eliminates for each objective variable  $obj_i$  all the value  $a$  such that  $\tau^*(O, i, a)$  is weakly-dominated, by setting the maximum of  $obj_i$  to  $minV - 1$  where  $minV = \min(\{a \text{ such that } \tau^*(O, i, a) \text{ is weakly-dominated}\})$ .*

Algorithm 3.2 is a possible implementation of this method.

Figure 5 shows two examples based on the same archive with different states for the domains. In Figure 5 (a),  $obj_1$  and  $obj_4$  are not filtered because for both cases there is no arc between a node marked from  $root$  and a node marked from  $tt$  at their corresponding layer. Concerning  $obj_2$  there is the arc  $(c, 5, f)$  that satisfies this condition so  $obj_2^{max} = \min(8, 5 - 1)$  and the domain becomes  $\{3, 4\}$ . For  $obj_3$ , the arc  $(e, 4, h)$  is the only one that satisfies this condition so  $obj_3^{max} = \min(7, 4 - 1)$  and the domain becomes  $\{2, 3\}$ . In Figure 5 (b),  $tt$  is reached from  $root$  during the first step of the bidirectional marking. Therefore there is no possible assignment with current domains.

The MDD is traversed two times with DFS and one time with the search of minimal value for each objective variable, then the time complexity of this method is linear in the size of the MDD.

■ **Algorithm 3.2** Bidirectional marking algorithm.

---

```

topDownMarking(MDDA,  $\mathcal{D}$ ,  $l$ ,  $u$ , markedFromRoot)
1  | markedFromRoot[ $l$ ].add( $u$ )
2  | for each arc ( $u$ ,  $a$ ,  $v$ ) do
3  |   | if  $a \leq obj_l^{min}$  and  $v \notin \text{markedFromRoot}[l + 1]$  then
4  |   |   | topDownMarking(MDDA,  $\mathcal{D}$ ,  $l + 1$ ,  $v$ , markedFromRoot)

bottomUpMarking(MDDA,  $\mathcal{D}$ ,  $l$ ,  $v$ , markedFromTt)
5  | markedFromTt[ $l + 1$ ].add( $v$ )
6  | for each arc ( $u$ ,  $a$ ,  $v$ ) do
7  |   | if  $a \leq obj_l^{min}$  and  $u \notin \text{markedFromTt}[l]$  then
8  |   |   | bottomUpMarking(MDDA,  $\mathcal{D}$ ,  $l - 1$ ,  $u$ , markedFromTt)

twoWaysMarkingFiltering( $O$ ,  $\mathcal{D}$ , MDDA)
9  | size  $\leftarrow O.size$ 
10 | for  $i \in 1 \dots size + 1$  do
11 |   | markedFromRoot[ $i$ ]  $\leftarrow \emptyset$ 
12 |   | markedFromTt[ $i$ ]  $\leftarrow \emptyset$ 
13 | topDownMarking(MDDA,  $\mathcal{D}$ , 1, root, markedFromRoot)
    | // If tt si reached, it means that there is no
    | // more possible assignment with current domains.
14 | if  $tt \in \text{markedFromRoot}[size + 1]$  then
15 |   | backtrack
16 | else
17 |   | bottomUpMarking(MDDA,  $\mathcal{D}$ , size, tt, markedFromTt)
18 |   | for  $i \in 1 \dots size$  do
19 |     | minValue  $\leftarrow \text{MAX\_INTEGER}$ 
20 |     | for each  $u \in \text{markedFromRoot}[i]$  do
21 |       | for each arc ( $u$ ,  $a$ ,  $v$ ) do
22 |         |   | if  $v \in \text{markedFromTt}[i + 1]$  and  $a < minValue$  then
23 |         |         |   | minValue  $\leftarrow a$ 
24 |         |   | objimax  $\leftarrow \min(obj_i^{max}, minValue - 1)$ 

```

---

## 4 Experiments

The methods presented in this paper have been implemented in Java 17 using Choco-solver version 4.10.10 [9]. All the experiments were run in sequential on a machine with an Intel(R) Xeon(R) W-2175 CPU @ 2.50GHz using Ubuntu 20.04.6 LTS version 5.4.0-146-generic.

In these experiments, three implementations of the PARETO constraint are compared:

- *List*: the PARETO constraint of Choco, using a list for representing the archive. When a new solution is inserted, all the solutions in the list are compared with this solution to determine if they must be removed from the list. Concerning the filtering, for each objective variable  $obj_i$  the inconsistent values are found by comparing all the solutions in the list with the *dominated point*  $DP_i$ , defined in [11, 3].
- M-U: the PARETO constraint associated with an MDD for representing the archive and using the Unidirectional marking algorithm as filtering algorithm.

■ **Table 1** Time (s) comparison between the use of lists and MDDs for the PARETO constraint with 10 objectives. The search ends when all solutions are found or if 30000 solutions are found, or if it exceeds 30 minutes (TO).

n	Data	# Solutions found	#Solutions in $\mathcal{A}$	Total time			Filtering time			Deletion and insertion time		
				List	M-U	M-B	List	M-U	M-B	List	M-U	M-B
12	b1	6770	1687	11	46	<b>9</b>	<b>3</b>	23	8	6	<b>0.7</b>	0.8
	b2	7443	3004	36	61	<b>11</b>	7	26	<b>6</b>	25	<b>0.8</b>	0.9
	b3	7351	2651	29	55	<b>11</b>	<b>6</b>	23	7	19	<b>0.6</b>	0.7
	b4	7754	3187	40	61	<b>13</b>	<b>7</b>	32	8	29	<b>0.8</b>	0.9
	b5	8502	3284	49	129	<b>18</b>	<b>10</b>	65	13	35	<b>0.9</b>	<b>0.9</b>
16	b6	30000	4894	420	781	<b>103</b>	87	360	<b>70</b>	309	<b>4</b>	<b>4</b>
	b7	30000	5805	708	TO	<b>239</b>	170	TO	<b>166</b>	487	TO	<b>5</b>
	b8	30000	5015	615	901	<b>159</b>	<b>106</b>	460	116	478	<b>3</b>	<b>3</b>
	b9	30000	7763	1336	1171	<b>178</b>	236	467	<b>128</b>	1055	6	<b>5</b>
	b10	30000	7144	1150	TO	<b>312</b>	<b>215</b>	TO	236	881	TO	<b>5</b>
20	b11	30000	2085	265	504	<b>115</b>	100	247	<b>67</b>	123	<b>5</b>	<b>5</b>
	b12	30000	3812	386	938	<b>142</b>	131	508	<b>86</b>	207	<b>6</b>	<b>6</b>
	b13	30000	1198	<b>143</b>	827	159	<b>56</b>	439	77	17	<b>3</b>	4
	b14	30000	2647	246	1430	<b>205</b>	<b>107</b>	709	108	56	<b>3</b>	<b>3</b>
	b15	30000	2707	713	TO	<b>406</b>	427	TO	<b>216</b>	108	TO	<b>5</b>

- M-B: the PARETO constraint associated with an MDD for representing the archive and using the Bidirectional marking algorithm as filtering algorithm.

The considered problems are the bin packing and the multi-criteria knapsack problems.

#### 4.1 Bin Packing Problem

The bin packing problem is a problem where  $n$  items have to be placed into bins. Each item has  $m$  types of weight, and each bin has a limit for each type of weight. For each type of weight the objective is to minimize the maximum weight among all the bins, so there are  $m$  objectives. These objectives encourage an equitable distribution of weights. The datasets used involve items with weights randomly chosen between 1 and 40, and a limit of 120 for each type of weight for each bin. These items have to be distributed between 8 bins. The problem is modeled using the matrix-based symmetry-breaking constraints proposed by Salem and Kieffer [10].

In order to compare the different methods, we focused on the time taken to find at most 30000 solutions. Moreover, we measured the total time taken by the filtering of the Pareto constraint throughout the search (Filtering time), and the total time taken to maintain the domination-free property of the archive throughout the search (Deletion and Insertion time, or D&I time). Table 1 shows the evolution of these times depending of the number of items  $n$  while Table 2 shows this evolution depending of the number of objectives  $m$ .

The first thing that comes out of the results of Table 1 is that M-B generally performs better than the list for this problem. These performances seem to depend on the size of the archive: the larger it is, the more the compression of MDDs shows its advantage. When we look at the time spent on the different operations, we can notice that this time saving is mainly done during the D&I part. For example with data b9, which has the largest archive with 7763 solutions, M-B is 7.5 times faster than the list and its D&I part is 200 times faster than the D&I part of the list. Concerning the filtering part M-B is sometimes slower than the list but when this is the case, it is not much slower. Moreover, when M-B is faster on the filtering it can be up to almost 2 times faster as with data b15.

The experiments in Table 2 show another interesting phenomenon about the filtering

■ **Table 2** Time (s) comparison between the use of lists and MDDs for the PARETO constraint with 16 items. The search ends when all solutions are found or if 30000 solutions are found, or if it exceeds 30 minutes (TO).

m	Data	# Solutions found	#Solutions in $\mathcal{A}$	Total time			Filtering time			Deletion and insertion time		
				List	M-U	M-B	List	M-U	M-B	List	M-U	M-B
5	b16	4619	121	<b>7</b>	17	9	<b>0.5</b>	5	2	<b>0.1</b>	0.2	0.2
	b17	5452	640	<b>14</b>	58	19	<b>4</b>	28	9	1	<b>0.3</b>	<b>0.3</b>
	b18	5679	382	<b>3</b>	12	5	<b>0.6</b>	6	2	<b>0.3</b>	<b>0.3</b>	<b>0.3</b>
	b19	8078	597	<b>16</b>	52	20	<b>4</b>	24	9	2	<b>0.6</b>	0.7
	b20	4658	234	<b>3</b>	6	4	<b>0.3</b>	2	1	<b>0.1</b>	0.2	0.2
10	b6	30000	4894	420	781	<b>103</b>	87	360	<b>70</b>	309	<b>4</b>	<b>4</b>
	b7	30000	5805	708	TO	<b>239</b>	170	TO	<b>166</b>	487	TO	<b>5</b>
	b8	30000	5015	615	901	<b>159</b>	<b>106</b>	460	116	478	<b>3</b>	<b>3</b>
	b9	30000	7763	1336	1171	<b>178</b>	236	467	<b>128</b>	1055	6	<b>5</b>
	b10	30000	7144	1150	TO	<b>312</b>	<b>215</b>	TO	236	881	TO	<b>5</b>
15	b21	30000	2242	78	290	<b>44</b>	<b>16</b>	179	30	53	<b>2</b>	<b>2</b>
	b22	30000	4651	583	1005	<b>116</b>	105	545	<b>91</b>	456	<b>4</b>	<b>4</b>
	b23	30000	3442	271	363	<b>68</b>	70	180	<b>39</b>	178	<b>3</b>	4
	b24	30000	3379	296	1180	<b>97</b>	<b>54</b>	614	63	223	5	<b>4</b>
	b25	30000	8421	1359	1213	<b>121</b>	305	532	<b>77</b>	1014	10	<b>9</b>
20	b26	30000	7145	911	1098	<b>77</b>	185	568	<b>47</b>	706	6	<b>5</b>
	b27	30000	6828	1059	TO	<b>149</b>	202	TO	<b>108</b>	828	TO	<b>7</b>
	b28	30000	5327	684	889	<b>87</b>	140	529	<b>61</b>	521	<b>5</b>	<b>5</b>
	b29	30000	5791	819	TO	<b>247</b>	243	TO	<b>176</b>	516	TO	<b>6</b>
	b30	30000	7142	864	855	<b>106</b>	178	342	<b>63</b>	650	<b>7</b>	<b>7</b>

part: compared to the list, the more objectives there are, the faster the filtering with M-B. For example with data b26 where there are 20 objectives, the filtering with M-B is almost 4 times faster than the filtering with the list. However, when there are few objectives the list is globally better as shown by the results for  $m = 5$ .

The results with M-U are not as good as those with M-B, and are even worse than those with the list. Indeed, even if M-U has an advantage on the D&I part compared to the list, the filtering takes too much time which negates totally the advantage.

## 4.2 Multi-Criteria Knapsack Problem

This problem is a variant of the knapsack problem with  $n$  items: the goal is not to maximize only one type of profit but  $m$  types of profit. Then, each item has  $m$  values and there are  $m$  objectives to maximize. The data sets used represent items with weights and values randomly chosen between 1 and 40. For each data set, the limit of the knapsack is equal to  $(\sum_{i=1}^n w_i)/2$  with  $w_i$  the weight of the  $i$ -th item.

We took the same types of measures as for the bin packing problem (i.e. the total time, the filtering time and the D&I time) but we only ran the methods with the list and M-B. Table 3 lists the results obtained.

The results between the methods are similar for this problem, the solving times are more or less equivalent for each instance. However, we can observe a behavior that we have already seen with the bin packing problem: M-B is generally better when the size of the archive is large. The results show also that for this problem the list is generally more efficient for the filtering while M-B is better for the D&I.



■ **Table 3** Time (s) comparison between the use of lists and MDDs for the PARETO constraint with 10 objectives. The search ends when all solutions are found or if 10000 solutions are found.

n	Data	# Solutions found	#Solutions in $\mathcal{A}$	Total time		Filtering time		Deletion and insertion time	
				List	M-B	List	M-B	List	M-B
20	k1	1809	640	<b>182</b>	186	<b>1</b>	4	0.3	<b>0.2</b>
	k2	3667	1750	<b>309</b>	317	<b>6</b>	13	4	<b>0.9</b>
	k3	1661	1661	<b>134</b>	135	<b>0.5</b>	2	<b>0.2</b>	<b>0.2</b>
21	k4	10000	6714	786	<b>672</b>	<b>52</b>	97	175	<b>6</b>
	k5	7806	3265	<b>865</b>	868	<b>27</b>	59	15	<b>2</b>
	k6	4537	815	472	<b>467</b>	<b>5</b>	11	<b>1</b>	<b>1</b>
22	k7	3705	2750	<b>1011</b>	1050	<b>20</b>	35	8	<b>1</b>
	k8	1895	601	450	<b>446</b>	<b>2</b>	6	<b>0.3</b>	<b>0.3</b>
	k9	10000	5192	1123	<b>1073</b>	<b>59</b>	92	73	<b>6</b>
23	k10	10000	4561	1116	<b>1101</b>	<b>54</b>	99	66	<b>6</b>
	k11	4634	1593	<b>1360</b>	1420	<b>21</b>	46	3	<b>1</b>
	k12	5390	2816	1225	<b>1193</b>	<b>22</b>	40	11	<b>2</b>
24	k13	1791	823	<b>848</b>	873	<b>4</b>	12	0.5	<b>0.3</b>
	k14	10000	5781	1402	<b>1335</b>	<b>77</b>	112	89	<b>5</b>
	k15	10000	3528	<b>1545</b>	1570	<b>54</b>	92	45	<b>5</b>

## 5 Conclusion

In this paper we presented methods to use an MDD as an archive for the PARETO Constraint. The insertion of a new solution into the MDD and the deletion from the MDD of the solutions dominated by this new solution are made by applying classical operators. We presented two methods for establishing the bound consistency of this constraint: the unidirectional marking method and the bidirectional marking method. The second method is linear in the size of the MDD.

We have shown that, depending on the problem, using an MDD as an archive with the bidirectional marking method can be very effective compared to the classical list representation of the archive. The use of MDDs is particularly well suited to maintaining the dominance-free relation of the archive. It is also interesting for the filtering algorithm. These performances are even more important as the number of objectives increases.

---

## References

- 1 David Bergman, André A. Ciré, Willem-Jan van Hoes, and John N. Hooker. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2016. doi:10.1007/978-3-319-42849-9.
- 2 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.
- 3 Marco Gavanelli. An algorithm for multi-criteria optimization in cps. In *ECAI'02: Proceedings of the 15th European Conference on Artificial Intelligence*, pages 136–140, January 2002.
- 4 Renaud Hartert and Pierre Schaus. A support-based algorithm for the bi-objective pareto constraint. *Proceedings of the National Conference on Artificial Intelligence*, 4:2674–2679, June 2014. doi:10.1609/aaai.v28i1.9119.
- 5 T.Y.K. Kam and Robert K. Brayton. Multi-valued decision diagrams. Technical Report UCB/ERL M90/125, EECS Department, University of California, Berkeley, 1990. URL: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/1990/1671.html>.
- 6 Sanaz Mostaghim and Jürgen Teich. *Quad-trees: A Data Structure for Storing Pareto Sets in Multiobjective Evolutionary Algorithms with Elitism*, pages 81–104. Springer London, London, 2005. doi:10.1007/1-84628-137-7\_5.

- 7 Guillaume Perez. *Decision diagrams: constraints and algorithms*. PhD thesis, Université Côte d'Azur, 2017.
- 8 Guillaume Perez and Jean-Charles Régim. Constructions and in-place operations for mdds based constraints. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 279–293, Cham, 2016. Springer International Publishing.
- 9 Charles Prud'homme and Jean-Guillaume Fages. Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708, 2022. doi:10.21105/joss.04708.
- 10 Khadija Salem and Yann Kieffer. An experimental study on symmetry breaking constraints impact for the one dimensional bin-packing problem. In *Conference: 2020 Federated Conference on Computer Science and Information Systems*, pages 317–326, September 2020. doi:10.15439/2020F19.
- 11 Pierre Schaus and Renaud Hartert. Multi-objective large neighborhood search. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, pages 611–627, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- 12 A. Srinivasan, T. Ham, S. Malik, and R. K. Brayton. Algorithms for discrete function manipulation. In *1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, pages 92–95, 1990. doi:10.1109/ICCAD.1990.129849.
- 13 Taha Vafaenezhad, Reza Tavakkoli-Moghaddam, and Naoufel Cheikhrouhou. Multi-objective mathematical modeling for sustainable supply chain management in the paper industry. *Computers & Industrial Engineering*, 135:1092–1102, 2019. doi:10.1016/j.cie.2019.05.027.
- 14 Yihan Wang, Zongguo Wen, Jianguo Yao, and Christian Doh Dinga. Multi-objective optimization of synergic energy conservation and co2 emission reduction in china's iron and steel industry under uncertainty. *Renewable and Sustainable Energy Reviews*, 134:110128, 2020. doi:10.1016/j.rser.2020.110128.



# Learning a Generic Value-Selection Heuristic Inside a Constraint Programming Solver

Tom Marty ✉ 🏠 

Polytechnique Montréal, Montreal, Canada  
Ecole Polytechnique, Palaiseau, France

Tristan François ✉

Ecole Polytechnique, Palaiseau, France

Pierre Tessier ✉

Ecole Polytechnique, Palaiseau, France

Louis Gautier ✉

Ecole Polytechnique, Palaiseau, France

Louis-Martin Rousseau ✉ 🏠 

Polytechnique Montréal, Montreal, Canada

Quentin Cappart ✉ 🏠 

Polytechnique Montréal, Montreal, Canada

---

## Abstract

Constraint programming is known for being an efficient approach to solving combinatorial problems. Important design choices in a solver are the *branching heuristics*, designed to lead the search to the best solutions in a minimum amount of time. However, developing these heuristics is a time-consuming process that requires problem-specific expertise. This observation has motivated many efforts to use machine learning to automatically learn efficient heuristics without expert intervention. Although several generic *variable-selection heuristics* are available in the literature, the options for *value-selection heuristics* are more scarce. We propose to tackle this issue by introducing a generic learning procedure that can be used to obtain a value-selection heuristic inside a constraint programming solver. This has been achieved thanks to the combination of a *deep Q-learning* algorithm, a tailored *reward signal*, and a *heterogeneous graph neural network*. Experiments on *graph coloring*, *maximum independent set*, and *maximum cut* problems show that this framework competes with the well-known impact-based and activity-based search heuristics and can find solutions close to optimality without requiring a large number of backtracks.

**2012 ACM Subject Classification** Computing methodologies → Artificial intelligence; Computing methodologies → Machine learning

**Keywords and phrases** Branching heuristic, Deep reinforcement learning

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.25

## Supplementary Material

*Software (Source Code)*: <https://github.com/corail-research/SeaPearl.jl>  
archived at `swh:1:dir:fd35f1159af064307cb36323831a0254e9d4060f`

**Funding** The research presented in this paper has been partially funded thanks to a NSERC Discovery Grant held by Quentin Cappart.

**Acknowledgements** We thank CIRRELT for providing essential computing resources. We extend our gratitude to all the people who contributed to the related open-source project: Léo Boisvert, Tom Sander, Ziad El Assal, Malik Attalah, and Marco Novaes.



© Tom Marty, Tristan François, Pierre Tessier, Louis Gautier, Louis-Martin Rousseau, and Quentin Cappart;

licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 25; pp. 25:1–25:19

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Combinatorial optimization has countless industrial applications, such as scheduling, routing, or finance. Unfortunately, most of these problems are NP-hard and, thereby, challenging to solve efficiently. It is why finding good solutions has motivated intense research efforts for many years. Traditional methods for tackling them are somehow based on a *search procedure*: A clever enumeration of the solution space is performed to find a feasible and possibly optimal solution. Among these methods, *constraint programming* (CP) is an exact procedure. It constitutes a popular approach as it offers the possibility to find the optimal solution or good feasible approximations by stopping the search early. An additional asset is its declarative paradigm in modeling, which makes the technology easier for the end-user to grasp. Introducing solver-agnostic modeling languages, such as MiniZinc [35] has greatly facilitated this aspect. Aligned with this goal, the propagation engine inside a CP solver is mostly hidden from the end-user. However, ensuring a generic search procedure is trickier as non-trivial heuristics must be designed to make the solving process efficient for an arbitrary problem. That being said, generic *variable-selection* and *value-selection* heuristics have been successfully designed. Notable examples are *impact-based search* [37] or *activity-based search* [31], but they require computationally intensive initialization and yield poor performance at the beginning of the search. This makes these methods not always appropriate for general use. As a concrete example, the current version of MiniZinc<sup>1</sup> does not propose generic value-selection heuristics, except *in(out)domain* or *impact-based search*. In practice, heuristics are often designed thanks to problem-specific expert knowledge, which is often out of reach for end-users that do not have a solid background in artificial intelligence.

In another context, *machine learning* (ML) has been recently considered for automating the design of branching heuristics, both in constraint programming [11], mixed-integer programming [16, 23], or SAT solving [41]. Specifically, *reinforcement learning* (RL) [45] or *imitation learning* [22] approaches, often combined with *deep learning* [27], have gained special attention. Although this idea seems appealing, this is not an easy task to achieve in practice as several technical considerations must be taken into account in order to ensure both the efficiency and the genericity of the approach. In constraint programming, we identified three questions to resolve when learning a generic branching heuristic inside a solver. They are as follows:

1. *How to train the machine learning model?* An intuitive way is to leverage an RL agent that would explore the tree search by making branching decisions and rewarding it based on the quality of the solution found on a terminal node. This would typically be done with a *depth-first search* traversal of the tree for getting a certificate of optimality. However, as pointed out by several authors [38, 42], the backtracking operations inside a solver raise difficulties when formalizing the task as a Markov decision process and may require redefining it. Besides, this training scheme intensifies the *credit assignment problem* [32], ubiquitous in reinforcement learning.
2. *How to evaluate the quality of a value selection?* A core component of an RL environment is the *reward function*, which gives a score to each decision performed. The end goal for the agent is to perform a sequence of decisions leading to the best-accumulated sum of rewards. In our case, an intuitive solution would be to reward the agent according to the quality of the solution found. However, this information is only available at terminal nodes, and only a zero reward is provided in branching nodes. This is related to the *sparse reward* problematic, which is known to complicate the training process.

---

<sup>1</sup> <https://www.minizinc.org/doc-2.7.0/en>

3. *How to learn from a CP model?* This question relates to the type of architecture that can obtain a value-selection heuristic from a search node (i.e., a partially solved CP model). A promising direction has been proposed by Gasse et al. [16] for binary mixed-integer programs. They introduced a bipartite graph linking variables and constraints (i.e., the two types of nodes) when a variable is involved in a given constraint. The subsequent architecture is a *heterogeneous graph neural network*. However, this encoding is not directly applicable in constraint programming, as a CP model generally involves non-binary variables and combinatorial constraints. This has been partially addressed by Chalumeau et al. [12], who introduced a tripartite graph where variables, values, and constraints are specific types of nodes. However, this approach lacks genericity as the method requires retraining when the number of variables changes.

To our knowledge, answering such questions is still an open challenge in the research community. This paper proposes to progress in this direction. It introduces a generic learning procedure that can be used to obtain a value-selection heuristic from a constraint programming model given as input. The approach has been designed to be generic in that it can be used for any CP model given as input. In practice, a specific way to extract *features* from a constraint should be designed for any available constraint, but this has to be done only once per constraint type. In this proof of concept, we limit our experiments to three combinatorial optimization problems, namely *graph coloring*, *maximum independent set*, and *maximum cut*. Specifically, we propose three main contributions, each dedicated to addressing one of the aforementioned difficulties. They are as follows: (1) a learning procedure, based on restarts, for training a reinforcement learning agent directly inside a CP solver, (2) a reward function able to assign non-zero intermediate rewards based on the propagation that has been carried out on the node, and (3) a neural architecture based on a tripartite graph representation and a heterogeneous graph neural network. Experimental results show that combining these three ideas enables the search to find good solutions without requiring many backtracks and competes with the well-known impact-based and activity-based search heuristics.

The paper is structured as follows. The next section presents other approaches related to our contribution. Then, Section 3 introduces succinctly technical background on reinforcement learning and graph neural networks. The core contributions are then presented in Section 4. Finally, Section 5 provides experimental results and closes with a discussion of the results.

## 2 Related Work

Bengio et al. [5] identified three ways to leverage machine learning for combinatorial optimization. First, *end-to-end* learning aims to solve the problem only with a trained ML model. This has been, for instance, considered for the traveling salesman problem [4, 25]. However, such an approach does not guarantee the validity nor optimality of the solution obtained. Second, *learning to configure* is dedicated to providing insights to a solver before its execution. This can be, for instance, the decision to linearize the problem in the context of quadratic programs [7] or to learn when a decomposition is appropriate [26]. This approach is also referred to as parameter tuning [21]. We refer to the initial survey for extended information about these two families of approaches. Third, *learning within a search procedure* uses machine learning within the solver. Our contribution belongs to this last category of methods. Although the idea of combining learning and searching for solving combinatorial optimization problems was already discussed in the nineties [36], it has re-emerged recently with the rise of deep learning. Most combinatorial optimization solvers are based on *branch-and-bound*

and *backtracking*. In this context, ML is often used with branching rules to follow. *Imitation learning* [22] has been for instance used to replicate the expensive *strong branching* strategy for mixed-integer programming solvers [16, 23]. One limitation of imitation learning is that the performances are bounded by the performance of the imitated strategy, which remains heuristic and perfectible [43]. This opens the door for RL approaches that have the guarantee to find the best branching strategy eventually [29]. A branching strategy can be split into two challenging decisions, *variable-selection* and *value-selection*. Reinforcement learning approaches have been considered for both of them.

Concerning the learning for selecting the next variable to branch on, Song et al. [42] proposed to combine a *double deep Q-network* algorithm [49] with a graph neural network for carrying out this task. The approach is trained to minimize the expected number of nodes to reach a leaf node using the *first-fail principle*. Although this is a good proxy for pruning a maximum of infeasible solutions for a constraint satisfaction problem, it does not extend naturally to optimization variants, for which one should consider a trade-off between the quality of the solution found and the number of nodes required to reach that solution. Similarly, van Driel et al. [48] leveraged a graph neural network to initialize a variable-selection heuristic for *Chuffed*, a hybrid CP-SAT solver. In an online setting, Doolard and Yorke-Smith [15] also proposed to learn variable ordering heuristics where training time is included in the total solving time. Bandit-based learning approaches were also considered by Xia and Yap to automatically select search heuristics [50].

For the value-selection heuristic, Chu and Stuckey [13] introduced a scoring function which gives a score indicating how good an assignation is, given the current domain. A training phase is carried out in a supervised manner to learn this scoring function. Cappart et al. [11] proposed to train a model with reinforcement learning outside the CP solver and to integrate the agent, once trained, subsequently in the solver. This has been achieved by reaping the benefits of a dynamic programming formulation of a combinatorial problem. An important limitation of this work is that no information related to the CP solver, such as the propagation achieved on a node, can be used to drive the decision. Chalumeau et al. [12] mitigated this issue by carrying out the learning inside the solver. The model is trained to find the optimal solution and to prove it with the least number of explored search nodes. However, this goal is disconnected from finding the best solution as quickly as possible and is practically hard to achieve, even with a good heuristic. A more realistic goal is to find a good solution quickly without closing the search. This is how the contribution of this paper is positioned.

We want to point out that learning *how to branch* is not the only way to leverage ML inside a combinatorial optimization solver. Related works have also been proposed on learning tight optimization bounds [9] or for accelerating column generation approaches [34]. A recurrent design choice is an architecture based on graph neural networks. We refer to the following survey for more information about combinatorial optimization with graph neural networks [10].

### 3 Technical Background

This section introduces the required background on reinforcement learning and graph neural network to grasp the technical aspects of the paper.

### 3.1 Reinforcement Learning

Let  $\langle S, A, T, R \rangle$  be a 4-tuple representing a *Markov decision process* where  $S$  is the set of states in the environment,  $A$  is the set of actions that the agent can do,  $T : S \times A \rightarrow S$  is a transition function leading the agent from one state to another, given the action taken, and  $R : S \times A \rightarrow \mathbb{R}$  is a reward function of taking an action from a specific state. The sequence  $[s_1, \dots, s_T]$  from the initial state ( $s_1$ ) of an agent towards a terminal state ( $s_T$ ) is referred to as an *episode*. The returned reward within a partial episode  $[s_t, \dots, s_T]$  can be formalized as follows:  $G_t = \sum_{i=t}^T R(s_i, a_i)$ . We intentionally omitted the discounting factor as we do not want to discount the late rewards in our application. The agent is governed by a policy  $\pi : S \rightarrow A$ , which indicates the action that must be taken on a given state. The agent's goal is to find the policy that will lead it to maximize the accumulated reward until a terminal state is reached. The core idea of reinforcement learning is to determine this policy by letting the agent interact with the environment and increasing the probability of taking action if it leads to high subsequent rewards. There are a plethora of reinforcement learning algorithms dedicated to this task, such as *trust region policy optimization* [40] or *soft actor-critic* [18]. We refer to *SpinningUp* website for explanations of the main algorithms [1].

This section presents the core principles of *deep Q-learning* [33], which is the algorithm used in this paper. The idea is to compute an *action-value* function  $Q^\pi(s_t, a_t) = G_t$ . Intuitively, this function gives the accumulated reward that the agent will obtain when performing the action  $a$  at state  $s$  while subsequently following a policy  $\pi$ . The output of this function for a specific action is referred to as a *Q-value*. Provided that the action-value function can be computed exactly, the optimal policy  $\pi^*$  turns to be simply the selection of the action having the highest Q-value on a specific state:  $\pi^* = \operatorname{argmax}_\pi Q^\pi(s, a), \forall (s, a) \in (S, A)$ . Although the exact computation of Q-values can theoretically be performed, a specific value must be computed for each pair of states and actions, which is not tractable for realistic situations. It is why a tremendous amount of work has been carried out to approximate accurately and efficiently Q-values. Among them, deep Q-learning aims to provide a neural estimator  $\hat{Q}(s, a, \theta) \approx Q(s, a)$ , where  $\theta$  is a tensor of parameters that must be learned during a training phase. This algorithm is commonly enriched with other mechanisms dedicated to speed-up or stabilizing the training process, such as the *double deep Q-network* variant [49] or *prioritized experience replay* [39]. Concerning the neural architecture, we opted for a graph neural network, which is explained in the next section.

### 3.2 Graph Neural Network

Intuitively, the goal of a *graph neural network* (GNN) is to embed information contained in a graph (e.g., the structure of the graph, spatial properties, features of the nodes, etc.) into a  $d$ -dimensional tensor for each node  $u \in V$  of the graph. To do so, information on a node is iteratively refined by aggregating information from neighboring nodes. Each iteration of aggregation is referred to as a *layer* of the GNN and involves parameters that must be learned. Let  $h_u^k \in \mathbb{R}^{d \times 1}$  be the tensor representation of node  $u$  at layer  $k$  of the GNN,  $h_u^{k+1} \in \mathbb{R}^{l \times 1}$  be the tensor representation of this node at the next layer ( $l$  being the dimension of a node at the layer  $k+1$ ), and  $\theta_1 \in \mathbb{R}^{l \times d}$  and  $\theta_2 \in \mathbb{R}^{l \times d}$  be two matrices of parameters, respectively. Each GNN layer carries out the following update:

$$h_u^{k+1} = g\left(\theta_1 h_u^k \star \left(\bigoplus_{v \in N(u)} \theta_2 h_v^k\right)\right) \quad \forall u \in V \quad (1)$$

Three operations are involved in this update: (1)  $\bigoplus$  is an *aggregation* operator that is dedicated to aggregating the information of neighbors (e.g., *mean-pooling* or *sum-pooling*), (2)  $\star$  is a *merging* which enables to combine of the information of a node with the ones from the



neighbors (e.g., a concatenation), and (3)  $g$  is an element-wise non-linear activation function, such as the ones commonly used in fully-connected neural networks (e.g., ReLU [17]). Without loss of generality, the *bias* term is not included in the equation. A concrete implementation of a GNN defines these three functions adequately. The training is conducted in a fully-connected neural network through back-propagation and an optimizer based on stochastic gradient descent such as Adam [24].

## 4 Learning a Value-Selection Heuristic Inside a Solver

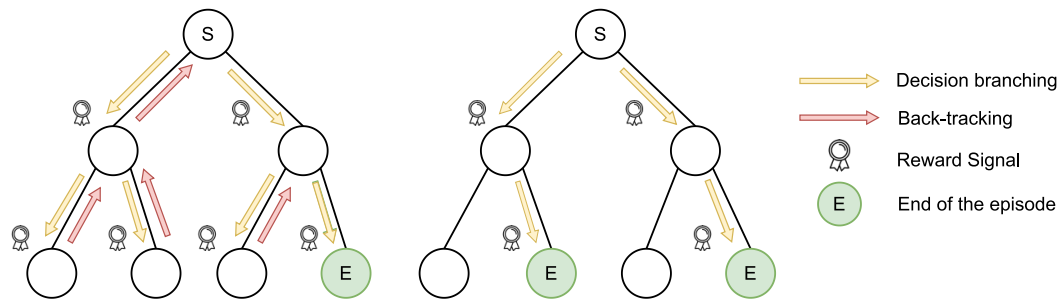
This section presents how a value-selection heuristic can be learned with reinforcement learning in a CP solver from a model given as input. This is the core contribution of the paper. Three mechanisms are introduced: (1) *a training procedure based on restarts*, (2) *a reward function leveraging propagation of domains*, and (3) *a heterogeneous graph neural network architecture*. They are described individually in the next subsections. They have been implemented in the recently introduced *SeaPearl.jl* solver [12]. Inspired by the architecture of MiniCP [30], the main specificity of SeaPearl is to natively integrate support for learning inside the search procedure. This greatly facilitates the prototyping of new search algorithms based on learning.

### 4.1 Restart-Based Training

Generally speaking, the performance of a reinforcement learning agent is tightly correlated with the definition of an *episode*. This corresponds to the agent’s interactions with the CP solver’s search procedure and is related to the goal desired for the agent. Two options are discussed in this section, (1) an *episode based on depth-first search*, introduced by Chalumeau et al. [12], and (2) an *episode based on restarts*, which is our first contribution.

Building branching heuristics for solving exact combinatorial optimization problems often concurrently targets two objectives: *finding quickly good solutions* and *proving the optimality of a solution*. The approach of Chalumeau et al. [12] relies heavily on the second objective and aims to minimize the number of visited search nodes before proving optimality (e.g., closing the search). To do so, they defined a training episode as a complete solving process carried out by the depth-first search of a solver and penalized through the reward function the generation of each node. This is illustrated in the left picture of Figure 1. However, this approach suffers from an important difficulty. An episode only terminates when the search is completed, which is often intractable for realistic problems as it requires exploring an exponentially large search tree. This is especially problematic during training, where the heuristic is still mediocre. In addition, using a depth-first search algorithm in a Markov Decision Process (MDP) framework required additional considerations not considered by Chalumeau et al. [12]. For example, using a backtracking algorithm in a regular temporal MDP renders their method prone to the *credit assignment problem* [32]. These considerations have been pointed out by Scavuzzo et al. [38] for mixed-integer programming.

Unlike this approach, we propose to train the model to find high-quality solutions quickly. To do so, we followed the approach proposed by Cappart et al. [11]: an episode is defined as a *single dive* in the search tree. No backtrack is allowed; the episode stops when a complete solution is found or a failure is generated. Once the episode is terminated, a restart from the root node is performed, and a new episode is generated, hence the name of *restart-based episode*. This is illustrated in the right picture of Figure 1. One limitation of Cappart et al. [11] is that episodes are executed outside the CP solver during the training and cannot use the information updated during propagation for the branching. Inspired by Song et



■ **Figure 1** The two training procedures (left: *depth-first search* [12], right: *restart-based* - ours).

al. [42] for variable-selection heuristics, we addressed this limitation by executing each episode inside the solver during the training. Formally, this requires defining the dynamics of the environment as a Markov Decision Process (i.e., a tuple  $\langle S, A, T, R \rangle$ , see Section 3.1). It is defined as follows.

**Set of states** Let  $\mathcal{P} = \langle X, D(X), C, O \rangle$  be the expression of a combinatorial optimization problem (COP), defined by its variables ( $X$ ), the related domains ( $D$ ), its constraints ( $C$ ), and an objective function ( $O$ ). Each state  $s_t \in S$  is defined as the pair  $s_t = (\mathcal{P}_t, x_t)$ , where  $\mathcal{P}_t$  is a partially solved COP (i.e., some variables may have been assigned), and  $x_t \in X$  is a variable selected for branching, at step  $t$  of the episode. The initial state  $s_1 \in S$  corresponds to the situation after the execution of the fix-point at the root node. A terminal node is reached either if all the variables are assigned ( $\forall x \in X : |D_t(x)| = 1$ ), or if a failure is detected ( $\exists x \in X : |D_t(x)| = 0$ ). The variable selected for branching is obtained through a standard heuristic such as *first-fail*.

**Set of actions** Given a state  $s_t = (\mathcal{P}_t, x_t)$ , an action  $a_t$  corresponds to the selection of a value  $v \in D(x_t)$  for branching at step  $t$ . Finding the most promising value to branch on is the problem addressed in this paper.

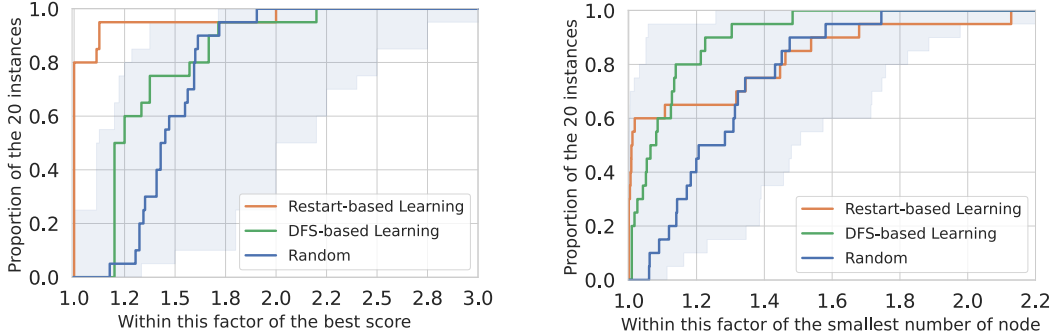
**Transition function** Given a state  $s_t = (\mathcal{P}_t, x_t)$  and an action  $a_t = v$ , the transition function executes three successive operations. First, it assigns the value  $v$  to the variable  $x$  (i.e.,  $D(x_{t+1}) = v$ ). Second, it executes the fix-point on  $\mathcal{P}_t$  in order to prune the domains (i.e.,  $\mathcal{P}_{t+1} = \text{fixPoint}(\mathcal{P}_t)$ ). Third, it selects the next variable to branch on (i.e.,  $x_{t+1} = \text{nextVariable}(\mathcal{P}_{t+1})$ ). This results in a new state  $s_{t+1} = (\mathcal{P}_{t+1}, x_{t+1})$ . Integrating the propagation inside the transition is one important difference with Cappart et al. [11].

**Reward function** The function is defined separately in Section 4.2.

Concerning the training, we opted for a *double deep Q-learning* algorithm [49], known to perform well for discrete action spaces. However, other RL algorithms could also be used. We compared our restart-based training procedure using a simple terminal reward based on the solution's score with the backtracking-based approach of Chalumeau et al. [12] using their reward at each step (penalty of 1 for each explored node). We selected the *maximum independent set problem* for this comparison with instances with 50 nodes. Results are presented using performance profiles [14] in Figure 2. A detailed explanation of the experimental protocol is proposed in Section 5.

We evaluated both methods on two metrics matching the objective for which they were specifically trained. We look at the value of the solution obtained after a single dive (Figure 2a) in the tree search and the number of nodes visited to prove optimality using a depth-first search (Figure 2b). As expected, we observe that the agent trained with the *restart-based learning* strategy allows good results regarding the optimality gap for the first solution found

after a single dive. Remarkably, our method yields a comparable ability to prove optimality compared to Chalumeau et al. [12], whose primary aim was specifically to solve the problem in the minimum number of nodes. This last result has to be mitigated as both RL-based methods lie in the range of the random strategy (shaded blue area).



(a) Score of the first solution obtained.

(b) Number of nodes visited until optimality.

■ **Figure 2** Comparison of both training methods on maximum independent set (50 nodes). As a non-learned baseline, we added the performances of an agent performing only random decisions. Training is carried out on randomly generated Barabási-Albert graphs [2]; we selected this type of distribution as the generated graphs are known to mimic human-made and natural organizations. The evaluation is performed on 20 other graphs following the same distribution.

Finally, as shown in Figure 2a, it is important to notice that the optimality gap returned by our method is still non-negligible at the first solution obtained. The complexity of a combinatorial problem lies mainly in closing this gap, which is why backtracking is required. Experiments with backtracking are proposed in Section 5.

## 4.2 Propagation-Based Reward

The definition of our reward must be aligned with our objective of *finding quickly good solutions* for the combinatorial problem. Based on our training procedure, an intuitive function is to reward the agent proportionally to the solution quality found at the end of an episode. In case of an infeasible solution found, a penalty can be given. The main drawback of this rewarding scheme is that this information is only available at terminal nodes, and no reward is provided in branching nodes. This is related to the *sparse reward* problem, which complicates the training process [47]. To address this challenge, one should find a way to give informative intermediate rewards along the solving process. To this end, we propose a new rewarding scheme based on the domain reduction of the objective variable (i.e., the variable that must be minimized or maximized). This reduction happens either thanks to the branching assignment or the application of the fix-point. There are two main components: (1) an *intermediate reward* ( $r^{\text{mid}}$ ) collected at branching nodes, and (2) *terminal reward* ( $r^{\text{end}}$ ) collected only at the end of an episode.

Assuming a minimization problem, the intermediate reward follows two principles: each domain reduction of the largest values of the domain is rewarded, and each domain reduction of the lowest values of the domain is penalized. It is important to note that following these principles does not guarantee the discovery of a good solution at the end of the branch. The rationale is to lead the agent to a situation where the minimum cost can be *eventually* obtained while removing costly solutions. It is formalized in Equations (2) to (4), where  $r_t^{\text{mid}}$  is the reward obtained at step  $t$ , and is illustrated in Figure 3. As shown in Equation (5),

the terminal reward is set to -1 if the leaf node corresponds to an infeasible solution and 0 if it is feasible. Finally, the total reward ( $r^{\text{acc}}$ ) accumulated during an episode of  $T$  steps is the sum of all intermediate rewards with the final term, as proposed in Equation (6).

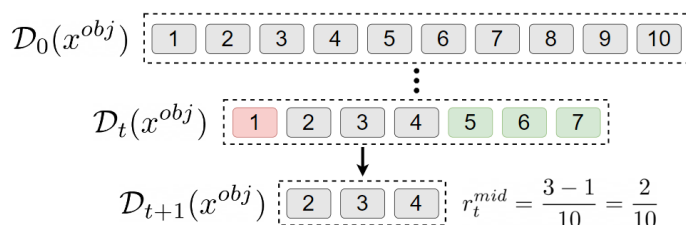
$$r_t^{\text{ub}} = \#\left\{v \in D_t(x^{\text{obj}}) \mid v \notin D_{t+1}(x^{\text{obj}}) \wedge v > \max(D_t(x^{\text{obj}}))\right\} \quad (2)$$

$$r_t^{\text{lb}} = \#\left\{v \in D_t(x^{\text{obj}}) \mid v \notin D_{t+1}(x^{\text{obj}}) \wedge v < \min(D_t(x^{\text{obj}}))\right\} \quad (3)$$

$$r_t^{\text{mid}} = \frac{r_t^{\text{ub}} - r_t^{\text{lb}}}{|D_1(x^{\text{obj}})|} \quad (4)$$

$$r_t^{\text{end}} = -1 \text{ if infeasible solution found (0 otherwise)} \quad (5)$$

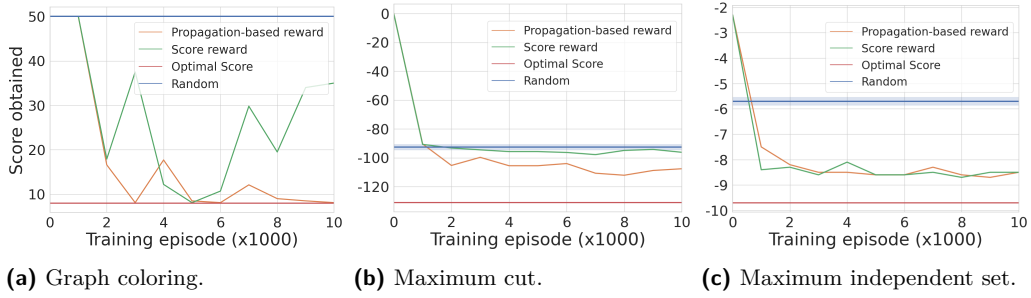
$$r^{\text{acc}} = \left(\sum_{t=1}^{T-1} r_t^{\text{mid}}\right) + r_T^{\text{end}} \quad (6)$$



■ **Figure 3** Intermediate reward when four values are pruned from the domain.

An experimental analysis of this new reward scheme (*propagation-based reward*) is carried out for the *graph coloring*, *maximum cut*, and *maximum independent set* problems; we look at the quality of the solution found after a single dive in the search tree. As a baseline, we consider a reward (*score reward*) that only gives a value at terminal nodes ( $r_T^{\text{end}}$ ) without an intermediate reward. Besides, we also consider the solutions returned by a random value-selection heuristic as a baseline. Figure 4 shows the evolution of the quality of the first solution returned ( $y$ -axis, averaged on 20 instances of the validation step) with the training time (number of episodes in the  $x$ -axis) using for training our restart-based search strategy defined in Section 4.1. Instances are Barabási-Albert randomly generated graphs with 50 nodes. Except for the rewarding scheme, the other parts of the architecture are unchanged. We observe that the *propagation-based reward* provides a more stable training (Figure 4a) and can converge to a better model or, at least, to an equally good model as the terminal *score reward* (Figures 4b and 4c).

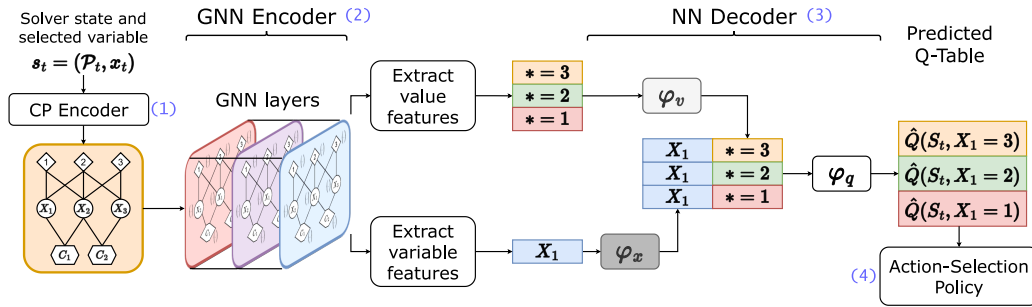
It should be noted that depending on the problem, the reward signal may remain sparse inside episodes even with our definition; this explains the discrepancy across the three class problems. Indeed, constraint propagation might take several steps to reach the objective variable, meaning that for related intermediate decisions, no value will be pruned from the domain of the objective variable. The *graph coloring* problem is thus the problem for which taking these intermediate rewards is the most beneficial. Indeed, any previously unused color added will negatively impact the domain of the objective function, yielding an insightful negative reward. Conversely, branching on the *maximum independent set* problem does not consistently impact the objective function domain through the mechanism of constraint propagation, particularly at the beginning of the search. Our method yields no worse result than the usual reward signal in this setting. This worst-case scenario empirically validates the robustness of this reward.



■ **Figure 4** Training curve for the two rewarding schemes, each validation step corresponds to performing a single dive in the search tree, the *score* obtained refers to the quality of the solution found on the leaf node.

### 4.3 Heterogeneous Graph Neural Network Architecture

An important part of the framework is the neural network architecture that we designed to perform a prediction of the next value to branch on. A high-level representation is proposed in Figure 5. Four steps are carried out: (1) a *CP model encoder*, (2) a *graph neural network encoder*, (3) a *neural network decoder*, and (4) an *action-selection policy*. They are detailed in the next subsections.



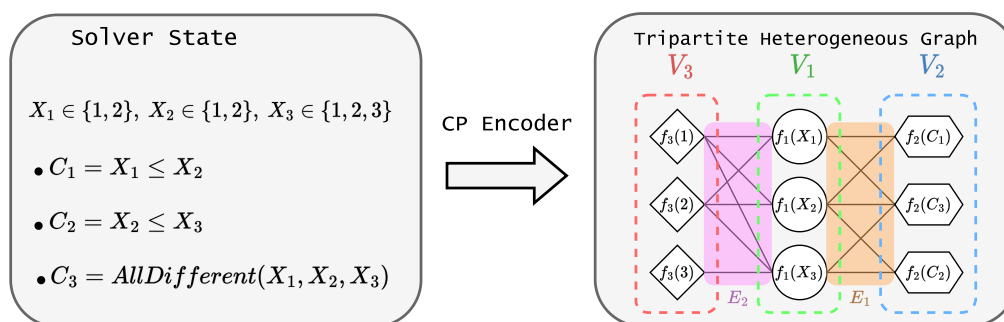
■ **Figure 5** High-level overview of the neural architecture designed.

#### Step 1: CP Model Encoder

The core idea is to learn for any CP model given as input, unlike Cappart et al. [11], who require a specific encoding for each combinatorial problem. This has been achieved for mixed-integer programs thanks to a bipartite graph representation [16] and by Chalumeau et al. [12] for CP models thanks to a tripartite graph. This last work does not leverage any feature related to the variables, values, or constraints. We built upon this last approach by adding such features. Specifically, let  $\mathcal{P} = \langle X, D(X), C, O \rangle$  be the combinatorial problem we want to encode. The idea consists in building a simple undirected graph  $\mathcal{G}(V_1, V_2, V_3, f_1, f_2, f_3, E_1, E_2)$  encoding all the information of  $\mathcal{P}_t$  from a state  $s_t = (\mathcal{P}_t, x_t)$ . In this representation,  $V_1, V_2$ , and  $V_3$  are three sets of vertices,  $f_1, f_2$ , and  $f_3$  are three sets of feature vectors, and  $E_1$  with  $E_2$  are two distinct sets of edges. This yields a graph with three types of nodes decorated with features. The first part of the encoding we propose is as follows: (1) each variable, constraint, and value corresponds to a specific type of node ( $V_1 = X, V_2 = C$ , and  $V_3 = D$ ), (2) each time a variable  $x \in V_1$  is involved in a constraint  $c \in V_2$ , an edge  $(x, c) \in E_1$  is added

between both nodes, (3) each time a value  $v \in V_3$  is in the domain of a variable  $x \in V_1$ , an edge  $(v, x) \in E_2$  is added between both nodes. This gives a tripartite graph representation of a CP model generically. This is illustrated in Figure 6. The second part of the encoding is to add features to each node. Intuitively, the features will provide meaningful information and thus improve the quality of the model. The features we considered are proposed below. We note that we can easily extend this encoding by integrating new features.

1. *Features attached to variables* ( $f_1$ ): the current domain size, the initial domain size, a binary indication if the variable is already assigned, and a binary indication if the variable corresponds to the objective.
2. *Features attached to constraints* ( $f_2$ ): the constraint type (one-hot encoding), and a binary indication if the constraint propagation has reduced domains.
3. *Features attached to values* ( $f_3$ ): its numerical value.



■ **Figure 6** Representation computed by the CP encoder on a simple example.

## Step 2: Graph Neural Network Encoder

Once the CP model has been encoded as a graph, the next step is to embed this representation as a latent vector of features for each node of the graph (see Section 3.2). We propose to carry out this operation with a graph neural network. Unlike the standard prediction scheme presented in Equation (1), our graph has three types of nodes. For this reason, we opted for a *heterogeneous* architecture. Concretely, a specific convolution is carried out for each node type. The architecture is detailed in Equations (7) to (9), where  $\oplus$  is the *sum-pooling* or *mean-pooling* aggregation, operator  $(\cdot\|\cdot)$  is a concatenation of vectors,  $N_x(n)$  is the set of neighbouring nodes of  $n$  from  $V_1$  (variable),  $N_c(n)$  is the set of neighbouring nodes of  $n$  from  $V_2$  (constraint),  $N_v(n)$  is the set of neighbouring nodes of  $n$  from  $V_3$  (value),  $\theta_{1,\dots,10}^k$  are weight matrices at layer  $k$ , and  $g$  is the *leakyReLU* activation function [28]. Another difference with the canonical GNN equation is the integration of *skip connections* ( $h_x^0$ ,  $h_c^0$ , and  $h_v^0$ ) allowing to keep at each layer information from the input features. This technique is ubiquitous in deep convolutional networks such as in *ResNet* [20]. Finally, the initial embedding are initialized as follows:  $h_x^0 = \theta_{11}f_1$ ,  $h_c^0 = \theta_{12}f_2$ , and  $h_v^0 = \theta_{13}f_3$ , where  $\theta_{11,\dots,13}$

are new weight matrices.

$$h_x^{k+1} = g\left(\theta_1^k h_x^0 \parallel \theta_2^k h_x^k \parallel \left(\bigoplus_{c \in N_c(x)} \theta_3^k h_c^k\right) \parallel \left(\bigoplus_{v \in N_v(x)} \theta_4^k h_v^k\right)\right) \quad \forall x \in V_1 \quad (7)$$

$$h_c^{k+1} = g\left(\theta_5^k h_c^0 \parallel \theta_6^k h_c^k \parallel \left(\bigoplus_{x \in N_x(c)} \theta_7^k h_x^k\right)\right) \quad \forall c \in V_2 \quad (8)$$

$$h_v^{k+1} = g\left(\theta_8^k h_v^0 \parallel \theta_9^k h_v^k \parallel \left(\bigoplus_{x \in N_x(v)} \theta_{10}^k h_x^k\right)\right) \quad \forall v \in V_3 \quad (9)$$

### Step 3: Neural Network Decoder

At this step, a  $d$ -dimensional tensor is obtained for each graph node. Let  $x \in V_1$  be the node representing the current variable selected for branching, and  $V_x \subseteq V_3$  the subset of nodes representing the values available for  $x$  (i.e., the values that are in the domain of the variable). The goal of the *decoder* is to predict a  $Q$ -value (see Section 3.1) for each  $v \in V_x$ . The computation is formalized in Equation (10), where  $h_x^K$  and  $h_v^K$  are the node embedding of variable  $x$  and value  $v$ , respectively, after  $K$  iterations of the GNN architecture. The functions  $\varphi_x : \mathbb{R}^d \rightarrow \mathbb{R}^l$ ,  $\varphi_v : \mathbb{R}^d \rightarrow \mathbb{R}^l$ ,  $\varphi_q : \mathbb{R}^{2l} \rightarrow \mathbb{R}$  are fully-connected neural networks. Such a  $Q$ -value must be computed for each value  $v \in V_x$ . It is internally done thanks to matrix operations, allowing a more efficient computation.

$$\hat{Q}(h_x^K, h_v^K) = \varphi_q\left(\varphi_x(h_x^K) \parallel \varphi_v(h_v^K)\right) \quad \forall v \in V_x \quad (10)$$

### Step 4: Action-Selection Policy

Once all the  $Q$ -values have been computed for the current variable, the policy is defined by an *explorer* that can decide to exploit the approximated  $Q$ -values by greedily choosing the best action as shown in Equation (11) or decide to select unpromising action associated with a lower  $Q$ -value (for example, by selecting a random action with probability  $\epsilon$ ). This behavior derives from the trade-off between exploitation and exploration, which is necessary for early learning when the estimates of  $Q$ -values are poor, and when only a few states have been visited. Once trained, the  $Q$ -values should represent the branching choice leading to the best decision according to the reward of Equation (6).

$$\pi(v|x) = \operatorname{argmax}_{v \in V_x} \hat{Q}(h_x^K, h_v^K) \quad (11)$$

Assembling all the pieces, this architecture gives a generic approach to obtaining a data-driven value-selection heuristic inside a CP solver. Concerning the search strategy used for evaluation (which is different from the *restart-based* one used for training), we propose to embed our predictions inside an *iterative limited discrepancy search* (ILDS) [19]. This strategy is commonly used when we are confident in the quality of the heuristic. The core idea is to restrict the number of branching choices deviating from the heuristic (i.e., a *discrepancy*). By doing so, the search will explore a subset of solutions expected to be good while giving a chance to reconsider the value-heuristic selection which is nevertheless prone to errors. This mechanism is enriched with a procedure that iteratively increases the number of discrepancies allowed once a level has been explored.

## 5 Experiments

The goal of this section is to evaluate the quality of the learned value-selection heuristic and the efficiency of the approach. Three combinatorial optimization problems are considered: *graph coloring* (COL), *maximum independent set* (MIS), and *maximum cut* (MAXCUT).

## 5.1 Experimental Protocol

Three configurations for the distribution of the problems generated are proposed for each problem: *small* (20 to 30 nodes), *medium* (40 to 50 nodes), and *large* (80 to 100 nodes) instances, except for MAXCUT which was already challenging for the *medium* size. Training is carried out on randomly generated Barabási-Albert graph [2] with a density factor varying between 4 and 15 according to the size of the instances. A specific model is trained for each configuration of each combinatorial problem. The training is done using randomly generated instances. Evaluation is then performed on 20 new graphs following the same distributions. The models are trained on an Nvidia Tesla V100 32Go GPU until convergence. It took up to 72 hours of training time for the most difficult cases (*graph coloring* with 80 nodes) and less than 1 hour for the simplest cases (*graph coloring* with 20 nodes). Each operation of the CP solver during training and evaluation is carried out on a CPU Intel Xeon Silver 4116 at 2.10GHz. The approach has been implemented in *Julia* and is integrated into the solver *SeaPearl*. The implementation is available on GitHub with BSD 3-Clause licence<sup>2</sup>.

We compared our approach (**Learned**, ILDS) with two other generic value selection heuristics: *impact-based search* (**Impact**) [37] and *activity-based search* (**Activity**) [31]. The standard *minDomain* heuristic is used for the variable selection. Comparisons with Chalumeau et al. [12] have been provided in Section 4.1. As it has been highlighted that this approach is not suited to find good solutions quickly, it is not included again in the next experiments. Each approach is evaluated with a fixed node budget depending on the parameters of the distribution used to generate the problems. For our approach, the performance obtained after the first dive in the tree search is also monitored (**Learned**, 1<sup>st</sup> dive). As **Impact** and **Activity** are online learning methods, they perform similarly to a random selection at the beginning of the search. For this reason, the performance obtained after the first dive in the tree search with such methods is omitted. Finally, we also included a comparison with a random selection using DFS with the same node budget (**Random**). Finally, the optimal cost (**OPT**) has been obtained with an exact approach without any restriction on the budget.

## 5.2 Quantitative Results

Table 1 summarizes the main results of our approach. As a general comment, our approach can find solutions of superior quality given a node budget or find the optimal solution by exploring fewer nodes than the baselines. Interestingly, our approach (**Learned**, ILDS) can learn a branching strategy giving high-quality solutions, even without backtracking (1<sup>st</sup> dive). For instance, a single dive for *maximum cut* with 50 nodes yields almost instantly a solution with an optimality gap of 0.16, whereas a depth-first search with a random selection (**Random**, DFS) required 19 seconds and roughly 53,000 nodes explored to find a solution with the same gap. Within this same budget, (**Learned**, ILDS) significantly improves the solution and achieves an optimality gap of 0.09. It is worth highlighting that (**Learned**, ILDS) took 130 seconds to explore 38,744 nodes and has, thereby, an exploration rate slower than the other methods. This significantly increased execution time is mainly because calling the graph neural network architecture (Section 4.3) at each tree search node is much more computationally expensive than calling a simple heuristic. This difficulty is further discussed in Section 5.3.

Concerning **Activity** and **Impact** heuristics, they yield no improvement on *graph coloring* compared to a random strategy. This can be explained by the fact that this class of problem has many possible combinations of variables and values for branching. This requires a

---

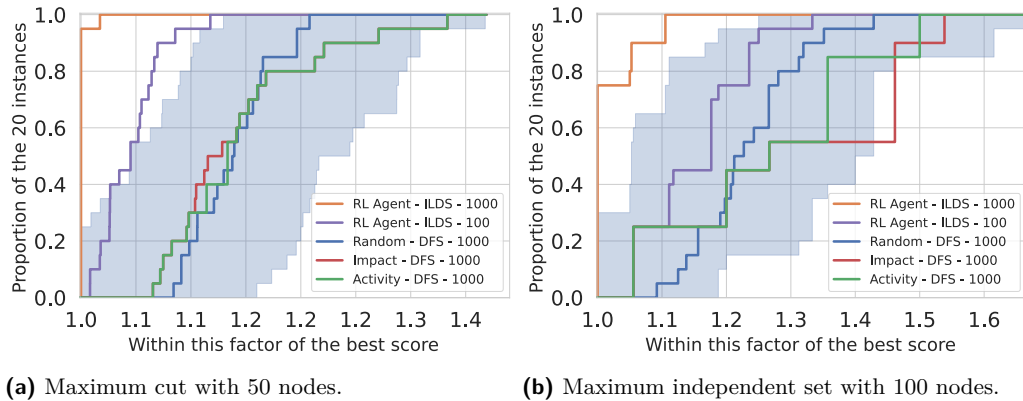
<sup>2</sup> <https://github.com/corail-research/SeaPearl.jl>



■ **Table 1** Results for the three problems given a fixed node budget. The average result (rounded) on the 20 test instances is reported for each configuration. *Gap* indicates the optimality gap, *Node* gives the number of nodes explored before finding the best solution within the budget, and *Time* gives the time (seconds) before finding this solution.

	Size	OPT	1 <sup>st</sup> dive			Learned ILDS			Activity-Based DFS			Impact-Based DFS			Random DFS			Budget
			Gap	Gap	Node	Time	Gap	Node	Time	Gap	Node	Time	Gap	Node	Time			
COL	20	5.05	0.06	0	27	< 1	0	378	< 1	0	374	< 1	0	378	< 1	10 <sup>3</sup>		
	40	7.90	0.08	0	104	< 1	0	1,664	< 1	0	1732	< 1	0	1735	< 1	10 <sup>4</sup>		
	80	8.75	0.06	0	120	1	0	7,051	2	0	7,057	2	0	7,211	2	10 <sup>5</sup>		
MIS	30	9.90	0.08	0	88	< 1	0	215	< 1	0	297	< 1	0	293	< 1	10 <sup>3</sup>		
	50	15.00	0.09	0	539	1	0	5,807	1	0	7,474	1	0	8,942	1	10 <sup>4</sup>		
	100	21.70	0.20	0.02	28,392	253	0.09	35,536	7	0.10	38,154	8	0.10	41,774	9	10 <sup>5</sup>		
MAXCUT	20	46.70	0.15	0.03	3,714	5	0.04	4,635	1	0.03	5,959	2	0.04	4877	1	10 <sup>4</sup>		
	50	222.00	0.16	0.09	38,744	130	0.17	44,664	14	0.17	47,970	17	0.17	53,110	19	10 <sup>5</sup>		

significantly larger number of explored nodes to initialize these two heuristics efficiently. For the two other problems, characterized by a binary domain for the values to branch on, **Activity** and **Impact** provide significantly better results than the random strategy, which is the expected behavior. In all the tested situations, (**Learned, ILDS**) provides the best optimality gap within the node budget. Additional results are proposed in Figure 7 using performance profiles [14] for the two hardest situations (100 for *maximum independent set*, and 50 for *maximum cut*) given a node budget of 100 or 1000 nodes.



■ **Figure 7** Best solutions found within a restricted node budget on largest instances for the three problems considered. We set a small budget to evaluate the ability of each approach to *find quickly a good solution*, which is the objective aimed by this work. The performance profile ratio is computed using the optimal solution as a reference. Within the same maximal number of nodes visited (1000), we observe that (**Learned, ILDS**) dominate all the other methods. Besides, we still perform better than the baselines when restricting ten times the budget for ILDS-Learned.

### 5.3 Discussions and Opportunities of Further Research

The previous experiments showcased the promise of this framework to quickly find good solutions towards a generic value-selection heuristic inside a CP solver. There are nonetheless open challenges that must be considered for practical use. Four of them are discussed.

### Challenge 1: Scalability of the Representation

Our approach faces a double penalty regarding its *scaling* capability: as the problem grows larger, the tripartite representation increases significantly in size, which results in a longer computation time required to make one branching decision. This impacts both training and evaluation. Additionally, the number of nodes (and, therefore, decisions to be made) in the search tree grows exponentially with the problem size, exacerbating the aforementioned phenomenon. Consequently, our approach is penalized twice due to the exponential behavior of combinatorial problems. As a concrete example, *graph coloring* instances with 80 nodes require 72 hours of training on a GPU, while only 1 hour is required for the smallest instances. An interesting research direction to mitigate this difficulty is to build a mechanism to compact the representation, for instance, thanks to network pruning tools [51] or with *transfer learning*. Another idea is to call the model only in a few nodes, in a similar fashion as Cappart et al. [9] did for *decision-diagram-based branch-and-bound* [6]. On a lower level of computation, standard constraint programming solvers perform sequential decisions and are therefore optimized for CPU architecture. Concerning the training, it is carried out on a GPU. In the current implementation, each branching decision requires loading the entire tripartite graph on the Video RAM, which is inefficient. We believe much work could be done to optimize this CPU/GPU architecture, for instance by delegating other operations on the GPUs, such as the propagation of few constraints [8, 46].

### Challenge 2: Tackling Highly Constrained Problems

The experiments proposed in the paper considered combinatorial problems where the difficulty lay in finding the *best* solution. Still, it was easy to find a *feasible* solution, even of poor quality. We empirically observed that the learning performance largely depends on the abundance of feasible solutions in the search space. This is explained by the definition of the reward, which is based on the propagation occurring on the objective variable (see  $r_t^{\text{mid}}$  in Section 4.2). However, when feasible solutions are not easily obtained, such as in highly constrained problems, the reward signal becomes less informative. Addressing such combinatorial problems remains an open challenge. We believe an extension of the reward signal can address this in order to handle other situations.

### Challenge 3: Learning a Combined Variable/Value Heuristic

Although this work proposes to learn a value-selection heuristic, learning how to branch on variables has already been considered in the literature [42]. An interesting research direction is to adapt this architecture to learn a variable-selection and a value-selection heuristic in a unified way. A possible direction is to consider a model with a double-head decoder, the first for selecting the variable and the second for selecting the value. On the training aspect, two reinforcement learning agents could be trained, with an the incentive to cooperate with the information sharing [44].

### Challenge 4: Proving the Optimality of a Solution

The goal pursued in this paper is to find the best solution as quickly as possible. Another direction is to guide the search to speed-up the optimality proof. It is what has been proposed by Chalumeau et al. [12]. In practice, finding good solutions and proving optimality are complementary aspects inside a constraint programming solver and should be both considered. Possible directions to do so could be to redefine the reward function appropriately or to revise the definition of an episode, as proposed by Scavuzzo et al. with TreeMDPs [38].

## 6 Conclusion

The efficiency of constraint programming solvers is partially due to the branching heuristics used to guide the search. In practice, value-selection heuristics are often designed thanks to problem-specific expert knowledge, often out of reach for non-practitioners. In this paper, we proposed a method based on reinforcement learning for obtaining such a heuristic, thanks to historical data, characterized by problem instances following the same distribution of the one that must be solved. This has been achieved thanks to a restart-based training procedure, a non-sparse reward signal, and a heterogeneous graph neural network architecture. Experiments on three combinatorial optimization problems show that the framework can find better solutions close to optimality in fewer nodes visited than other generic baselines. Several limitations and challenges (e.g., tractability for larger or real-world instances, transfer learning, sparsity of the reward signal) have been identified, and addressing them is part of future work. We also plan to consider other combinatorial problems, such as the ones proposed in XCSP3 competitions [3].

---

### References

- 1 Joshua Achiam. Spinning up as a deep RL researcher, October 2018. URL: [spinningup.openai.com/en/latest/spinningup/spinningup.html](https://openai.com/en/latest/spinningup/spinningup.html).
- 2 Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.
- 3 Gilles Audemard, Christophe Lecoutre, and Emmanuel Lonca. Proceedings of the 2022 XCSP3 competition. *arXiv preprint arXiv:2209.00917*, 2022.
- 4 Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.
- 5 Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d’horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.
- 6 David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.
- 7 Pierre Bonami, Andrea Lodi, and Giulia Zarpellon. Learning a classification of mixed-integer quadratic programming problems. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 595–604. Springer, 2018.
- 8 Federico Campeotto, Alessandro Dal Palu, Agostino Dovier, Ferdinando Fioretto, and Enrico Pontelli. Exploring the use of GPUs in constraint solving. In *Practical Aspects of Declarative Languages: 16th International Symposium, PADL 2014, San Diego, CA, USA, January 20-21, 2014. Proceedings 16*, pages 152–167. Springer, 2014.
- 9 Quentin Cappart, David Bergman, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Augustin Parjadis. Improving variable orderings of approximate decision diagrams using reinforcement learning. *INFORMS Journal on Computing*, 2022.
- 10 Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023. URL: <http://jmlr.org/papers/v24/21-0449.html>.
- 11 Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A. Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3677–3687, 2021.
- 12 Félix Chalumeau, Ilan Coulon, Quentin Cappart, and Louis-Martin Rousseau. Seapearl: A constraint programming solver guided by reinforcement learning. In *International Conference*


- on *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 392–409. Springer, 2021.
- 13 Geoffrey Chu and Peter J. Stuckey. Learning value heuristics for constraint programming. In *International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems 2015*, pages 108–123. Springer, 2015.
  - 14 Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.
  - 15 Floris Doolaard and Neil Yorke-Smith. Online learning of variable ordering heuristics for constraint optimisation problems. *Annals of Mathematics and Artificial Intelligence*, pages 1–30, 2022.
  - 16 Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.
  - 17 Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.
  - 18 Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.
  - 19 William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th international joint conference on Artificial intelligence*, volume 1, pages 607–613, 1995.
  - 20 Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
  - 21 Holger H. Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous search*, pages 37–71. Springer, 2011.
  - 22 Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.
  - 23 Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.
  - 24 Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
  - 25 Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL: <https://openreview.net/forum?id=ByxBFsRqYm>.
  - 26 Markus Kruber, Marco E Lübbecke, and Axel Parmentier. Learning when to use a decomposition. In *International conference on AI and OR techniques in constraint programming for combinatorial optimization problems*, pages 202–210. Springer, 2017.
  - 27 Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
  - 28 Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.
  - 29 Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.
  - 30 Laurent Michel, Pierre Schaus, and Pascal van Hentenryck. MiniCP: a lightweight solver for constraint programming. *Mathematical Programming Computation*, 13:133–184, 2021.

- 31 Laurent Michel and Pascal van Hentenryck. Activity-based search for black-box constraint programming solvers. In *International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming*, pages 228–243. Springer, 2012.
- 32 Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961. doi:10.1109/JRPROC.1961.287775.
- 33 Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- 34 Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Machine-learning-based column selection for column generation. *Transportation Science*, 55(4):815–831, 2021.
- 35 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.
- 36 Jean-Yves Potvin, Danny Dubé, and Christian Robillard. A hybrid approach to vehicle routing using neural networks and genetic algorithms. *Applied Intelligence*, 6(3):241–252, 1996.
- 37 Philippe Refalo. Impact-based search strategies for constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 557–571. Springer, 2004.
- 38 Lara Scavuzzo, Feng Yang Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree MDPs. *arXiv preprint arXiv:2205.11107*, 2022.
- 39 Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.
- 40 John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.
- 41 Daniel Selsam and Nikolaj Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 336–353. Springer, 2019.
- 42 Wen Song, Zhiguang Cao, Jie Zhang, Chi Xu, and Andrew Lim. Learning variable ordering heuristics for solving constraint satisfaction problems. *Engineering Applications of Artificial Intelligence*, 109:104603, 2022.
- 43 Haoran Sun, Wenbo Chen, Hui Li, and Le Song. Improving learning to branch via reinforcement learning. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020.
- 44 Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017. arXiv:1706.05296.
- 45 Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- 46 Fabio Tardivo and Agostino Dovier. Constraints propagation on GPU: A case study for alldifferent. In *Proceedings of the 37th Italian Conference on Computational Logic*, 2022.
- 47 Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. *Advances in Neural Information Processing Systems*, 32, 2019.
- 48 Ronald van Driel, Emir Demirović, and Neil Yorke-Smith. Learning variable activity initialisation for lazy clause generation solvers. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings 18*, pages 62–71. Springer, 2021.
- 49 Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

- 50 Wei Xia and Roland Yap. Learning robust search strategies using a bandit-based approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- 51 Xin Yu, Thiago Serra, Srikumar Ramalingam, and Shandian Zhe. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *International Conference on Machine Learning*, pages 25668–25683. PMLR, 2022.



# Proof Logging for Smart Extensional Constraints

Matthew J. McIlree ✉ 

University of Glasgow, UK

Ciaran McCreesh ✉ 

University of Glasgow, UK

---

## Abstract

Proof logging provides an auditable way of guaranteeing that a solver has produced a correct answer using sound reasoning. This is standard practice for Boolean satisfiability solving, but for constraint programming, a challenge is that every propagator must be able to justify all inferences it performs. Here we demonstrate how to support proof logging for a wide range of previously uncertified global constraints. We do this by showing how to justify every inference that could be performed by the propagation algorithms for two families of generalised extensional constraint: “Smart Table” and “Regular Language Membership”.

**2012 ACM Subject Classification** Theory of computation → Logic and verification; Theory of computation → Discrete optimization

**Keywords and phrases** Constraint programming, proof logging, extensional constraints

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.26

**Supplementary Material** *Software*: <https://doi.org/10.5281/zenodo.8132457>

**Funding** Part of this work was done while the authors were participating in a programme at the Simons Institute for the Theory of Computing.

*Ciaran McCreesh*: Supported by a Royal Academy of Engineering research fellowship.

**Acknowledgements** The authors would like to thank Jakob Nordström for several helpful discussions regarding pseudo-Boolean encodings and unit propagation.

## 1 Introduction

A proof log for a problem-solving algorithm provides a verifiable certificate that the result is correct, and also an auditable record of the steps taken to obtain that result. In the field of Boolean satisfiability (SAT), proof-logging has become an expected capability of modern solvers, with a series of proof formats including DRAT [12, 13], LRAT [7], and FRAT [2] widely accepted for independent verification. A similar standard practice has not yet been adopted for Constraint Programming (CP) due to the difficulties of creating easily verifiable proofs for the more expressive formulations and reasoning present in this paradigm. However, recent work by Gocht et al. [10] has shown how the VeriPB proof system [3, 9] can be used to certify the reasoning carried out for several important expressive global constraints, offering a strong candidate for a complete, general CP proof logging method. In this setting, every propagator for a global constraint must be able to do two things: describe its semantics in a lower-level *pseudo-Boolean* format, and justify any reasoning it carries out using either *cutting planes* [6] or *reverse unit propagation* (RUP) [11] reasoning. Describing a constraint’s semantics is a well-understood problem, but justifying propagation is not. Gocht et al. demonstrated proof logging for a range of global constraints and propagation strengths, including bounds-consistent integer linear inequalities and domain-consistent table constraints, but it remains an open question whether *every* global constraint propagation algorithm can be justified in this manner.



© Matthew J. McIlree and Ciaran McCreesh;

licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 26; pp. 26:1–26:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



This paper shows that the reasoning carried out by domain-consistent propagators for the `SmartTable` [16] and `Regular` [19] constraints can similarly be proof logged efficiently inside the `VeriPB` proof system. As well as being useful in their own right, these two constraints provide the necessary building blocks for implementing many others, since they allow for efficient strong propagation for extensional constraints that cannot be expressed efficiently as a conventional table. For example, `SmartTable` can be used to implement the `Lex`, `AtMostOne` and `NotAllEqual` constraints [16], whilst `Regular` can be used to implement `Stretch`, `Geost` and `DiffN` [15, 19]. Together, these two new constraints bring us a lot closer to fully auditable combinatorial solving, particularly in areas such as workforce scheduling where legal restrictions apply, and where algorithmic decisions can have a large effect upon people’s livelihoods.

## 2 An Overview of How Proof Logging Works

The `VeriPB` proof system is built upon pseudo-Boolean (PB) problems. A PB model is a restricted constraint satisfaction problem, where all variables  $x_i$  have domain  $\{0, 1\}$  and all constraints are integer linear inequalities of the format  $\sum_i c_i l_i \geq A$  where  $c_i, A \in \mathbb{Z}$ , and each  $l_i$  is a pseudo-Boolean *literal*: either a variable  $x$  or its negation  $\bar{x} = 1 - x$ . For clarity and by convention we will always refer to CP variables with upper-case symbols and PB variables with lower-case symbols. Additionally, to save space and make the meaning clearer we will make use of *reified* PB constraint shorthands:

- $(\ell_1 \wedge \dots \wedge \ell_m) \implies \sum_i c_i x_i \geq A$  means  $\sum_{i=1}^n c_i x_i + J\bar{\ell}_1 + \dots + J\bar{\ell}_m \geq A$  where  $J$  is chosen to be suitably large, e.g.  $J = A - \sum_{i=1}^n \min(c_i, 0)$ .
- $\sum_i c_i x_i \geq A \implies (\ell_1 \wedge \dots \wedge \ell_m)$  means  $\{\sum_{i=1}^n -c_i x_i + K\bar{\ell}_t \geq -A + 1 : 1 \leq t \leq m\}$  where  $K$  is chosen to be suitably large, e.g.  $K = -A + 1 - \sum_{i=1}^n \min(c_i, 0)$ .
- $(\ell_1 \wedge \dots \wedge \ell_m) \iff \sum_i c_i x_i \geq A$  means both of the above together.

PB constraints can also be viewed as a superset of conjunctive-normal form (CNF) for SAT problems, since a clause such as  $x_1 \vee \bar{x}_2 \vee x_3$  is always equivalent to a PB constraint such as  $x_1 + \bar{x}_2 + x_3 \geq 1$ . This means that it is straightforward to generalise known SAT encodings of CP constraints to PB. The concept of *unit propagation* from SAT can also be generalised to a PB setting. We say a PB constraint *propagates* a literal  $\ell$  if it cannot possibly be satisfied unless  $\ell = 1$ . For the “at least 1” constraints that can be viewed as clauses, this is the same as SAT unit propagation, but in general it is enforcing integer bounds consistency [5] on the PB inequalities. The (generalised) notion of unit propagation for PB constraints is therefore repeatedly setting any literals that propagate until either a contradiction or a fixed point is reached.

An auditable constraint solver following the proof-logging methodology of Gocht et al. [10] must be able to produce a pseudo-Boolean representation of the problem being solved, and then justify its backtracking and inference steps in a proof log by outputting further PB constraints derivable by one of the proof rules allowed by the `VeriPB` proof checker. The full proof system associated with `VeriPB` is described in detail by Bogaerts et al. [3], but for the purposes of this paper the only rule that is needed is the *reverse unit propagation* (RUP) rule, which says that a PB constraint  $D$  is derivable from a set of PB constraints  $F$  if applying the pseudo-Boolean generalisation of unit propagation (repeated integer bounds consistency) to  $F \cup \bar{D}$  results in a contradiction. Intuitively, it is a constraint that “obviously” follows once it is pointed out to the verifier.

This paper will omit discussion of how the variables and constraints of CP problem can be represented as PB constraints and variables in general, focussing specifically on how `Regular` and `SmartTable` can be represented. We will assume that we have access to a

collection of properly defined PB variables  $x_{bneg}, x_{b0}, x_{b1}, x_{b2} \dots$  representing bits in the two's-complement encoding of a CP variable  $X$  (with suitable constraints to enforce the domain), but also that we can freely use flags of the form  $x_{=v}$ , which are defined to equal 1 if and only if  $X = v$ . Explanation of how these encodings can be achieved is given by Gocht et al. [10].

Similarly, we will omit detailed discussion of how complete proofs of solutions, unsatisfiability, optimality, or enumeration are achieved. We simply note that each proof is essentially a description of the solver's backtracking search tree. Any time the solver backtracks, a RUP constraint that encodes the negated conjunction of the currently guessed assignments is emitted. A new propagation algorithm can fit into this existing framework by ensuring that the encoding of any specific filtering and failure inferences it makes are visible to the verifier on reverse unit propagation of the backtrack constraint. So for example, if a solver has guessed  $A = 2, B = 1$ ; and a propagator is able to infer  $C \neq 1$  then the proof logging methodology requires that the PB constraint

$$\bar{a}_{=2} + \bar{b}_{=1} + \bar{c}_{=1} \geq 1 \quad (1)$$

is somehow derived, which can be interpreted as

$$\bar{a}_{=2} \vee \bar{b}_{=1} \vee \bar{c}_{=1} \quad \text{i.e.} \quad A = 2 \wedge B = 1 \implies C \neq 1. \quad (2)$$

This is in many ways similar to how lazy clause generating solvers work [18], except that the new constraints introduced must be justified, rather than asserted.

### 3 Proof Logging for Smart Table Constraints

A smart table constraint generalises the idea of a table constraint to allow wildcards and comparisons, allowing for compact representations for a much larger set of relations, whilst still retaining an efficient domain-consistent propagator [16, 20, 21]. There are several ways to define `SmartTable` (also called `HybridTable`), depending on which restriction types are allowed within tuples [4]. In this work we restrict our focus to unary comparisons, binary comparisons, and set membership, and define `SmartTable` as follows.

► **Definition 1.** *Smart Table Constraint*

Let  $\mathbf{X}$  be a sequence of finite-domain variables  $\langle X_1, \dots, X_n \rangle$ . A smart entry constraint is a unary or binary constraint in one of three possible forms:

1.  $\langle var \rangle \langle op \rangle a$
2.  $\langle var \rangle \in S$  or  $\langle var \rangle \notin S$
3.  $\langle var \rangle \langle op \rangle \langle var \rangle$

where  $a$  is an integer constant,  $S$  is a set of integer constants,  $\langle op \rangle$  denotes an operator in the set  $\{<, \leq, =, \neq, \geq, >\}$ , and  $\langle var \rangle$  denotes a variable. A smart tuple is a set of these smart entry constraints, and a smart table is a set of smart tuples. When a variable does not appear in a given tuple, it is implicitly unrestricted (equivalent to a wildcard entry).

For a given smart table  $T$ , where the scope of every smart entry constraint is a subset of  $\mathbf{X}$ , a smart table constraint `SmartTable`( $\mathbf{X}, T$ ) requires that there is at least one smart tuple in  $T$  where every smart entry constraint holds. It can hence be thought of as a special case of a disjunction of conjunctions of simple constraints.

### 3.1 PB Encodings for Smart Table Constraints

Recall that in order to support proof logging for a constraint, we must be able to describe its semantics in a lower-level PB form. Let  $T$  be a smart table with smart tuples  $\sigma_1, \dots, \sigma_k$ . We can encode the `SmartTable` constraint by making use of some auxiliary PB variables. For each smart tuple  $\sigma_i$  in the table we will have a selector PB variable  $s_i$  that controls whether the tuple is active (i.e. not yet infeasible). Then for each smart entry  $C_j \in \sigma$  we will have a PB variable  $e_{ij}$  that is set to 1 if and only the constraint  $C_j$  is satisfied, which we denote by `satisfied`( $C_j$ ).

Since smart entries are themselves simple binary and unary constraints, with all of them having PB encodings that already well understood, the process of encoding this `satisfied` property is straightforward. Inequalities on one or two variables can be encoded by imposing the inequality on the evaluation of the bit encodings. For example, for two CP variable  $A$  and  $B$  both with domain  $\{1, \dots, 7\}$ , and recalling that PB variables  $a_{bi}$  and  $b_{bi}$  denote the  $i$ th bits in the encoding of the variables  $A$  and  $B$ , we would have

$$\text{satisfied}(A < B) := -a_{b0} - 2a_{b1} - 4a_{b2} + b_{b0} + 2b_{b1} + 4b_{b2} \geq 1. \quad (3)$$

Other inequality constraints can be handled similarly, adding or subtracting constants from the right-hand side as necessary. For  $\in$  and  $\notin$  it is simply a case of imposing “at least  $n$ ” constraints on the flags representing the disallowed values, e.g.

$$\text{satisfied}(A \in \{1, 3, 5\}) := \bar{a}_{=2} + \bar{a}_{=4} + \bar{a}_{=6} + \bar{a}_{=7} \geq 4, \quad (4)$$

$$\text{satisfied}(B \notin \{1, 3, 5\}) := \bar{b}_{=1} + \bar{b}_{=3} + \bar{b}_{=5} \geq 3. \quad (5)$$

Finally, for  $=$  and  $\neq$  we can make direct use of the  $x_{=v}$  flag if the right-hand side is a value. If the right-hand side is instead another variable we make use of further intermediate flags to express the relation in terms of strict or non-strict inequalities, so:

$$\text{satisfied}(A = B) := f_{A>B} + f_{A<B} \geq 2, \quad (6)$$

$$\text{satisfied}(A \neq B) := f_{A>B} + f_{A<B} \geq 1, \quad (7)$$

where the flags are enforced with two-way implication, e.g.

$$f_{A<B} \iff \text{satisfied}(A < B). \quad (8)$$

Obviously, the negation of these encodings can be expressed by encoding the negation of the constraint, e.g.

$$\neg \text{satisfied}(A < B) := \text{satisfied}(A \geq B), \quad (9)$$

$$\neg \text{satisfied}(A \neq B) := \text{satisfied}(A = B). \quad (10)$$

With the procedures for each of these smart entry constraint encodings in place, and recalling that we know how to reify arbitrary PB constraints on a literal, the PB model for smart table can then be created according to the following specification.

► **Encoding 1.** *Smart Table PB Encoding*

```

1: for all  $\sigma_i \in T$ 
2:   for all  $C \in \sigma_i$ 
3:      $e_C \iff \text{satisfied}(C)$ 
4:    $s_i \iff \sum_{C \in \sigma_i} e_C \geq |\sigma_i|$ 
5:  $s_1 + s_2 + \dots + s_{|T|} \geq 1$ 

```

### 3.2 Justifying Smart Table Propagation

The goal of the propagator for the smart table constraint, as with many global constraint propagators, is to achieve *domain consistency*. As a running example, suppose we had a smart table constraint on three variables,  $A, B, C$ , all with domains  $\{1, 2, 3\}$ , defined by the following table  $T$  with two tuples:

$$T := \{\sigma_1, \sigma_2\}; \quad \sigma_1 = \{(A < B), (A \in \{1, 2\}), (C = 3)\}, \quad (11)$$

$$\sigma_2 = \{(A = B), (A \neq 1), (B \geq C)\}. \quad (12)$$

By inspection, we can observe that the literal  $B = 1$  does not have any support on this constraint, as it would contradict  $A < B$  on the first tuple and  $A = B, A \neq 1$  together on the second. Therefore, a propagation algorithm that achieves domain consistency should immediately remove the value 1 from the domain of  $B$ .

The propagation algorithm described by Mairy et al. [16] uses a ‘‘Simple Tabular Reduction’’ strategy to eliminate such unsupported values. Essentially it initialises a set  $sl$  (for ‘‘supportless’’) with every variable value/pair that is possible given the current domains of variables, and then iterates through the tuples, removing any values that they in turn support. The literals that are left in  $sl$  are not supported by any tuple and hence the ones that should be eliminated. A key observation is that each smart tuple  $\sigma$ , as a conjunction of simple constraints, can be thought of as a small CSP  $P_\sigma$  in its own right, and so the supported values are just all the solutions for this problem. It is shown that as long as the constraint graph of  $P_\sigma$  is acyclic it can be effectively filtered as a collection of tree CSPs via a two pass filtering process. Due to the restricted structure of the smart tuples, the result of this is a collection of copies of the domains where the only values present are those that appear in some solution (global consistency for  $P_\sigma$ ), and so this can be used to remove values from  $sl$  in polynomial time.

For proof logging, the main concern is then how to justify the elimination of these unsupported values. In particular, for a sequence of guesses  $\mathcal{G}$  and newly derived unsupported assignment  $X \neq v$ , we would like to log the PB constraint

$$\wedge \mathcal{G} \implies \bar{x}_{=v} \geq 1. \quad (13)$$

In some situations this might follow directly by reverse unit propagation, as is always the case for proof-logging the classical table constraint as shown by Gocht et al. [10]. For smart tables, however, we can show that more work is required in general. Going back to the previous example and following Encoding 1, we note that if we try to derive  $\bar{b}_{=1} \geq 1$  by reverse unit propagation, we do not reach a contradiction after propagating its negation  $b_{=1}$ . This is despite  $B = 1$  lacking support in the table. To see why this is the case, note that the only way unit propagation of a non-auxiliary literal such as  $b_{=1}$  could falsify the whole model would be for it to eventually result in all the selectors  $s_i$  being falsified. In turn, the only way that could happen is for there to be at least one  $e_{iC}$  falsified for every  $i \in \{1, \dots, |T|\}$ . We can observe that this does not happen on propagation of  $b_{=1}$ . The smart entry constraints that are responsible for eliminating  $(B = 1)$ ’s support are  $A < B$  in the first tuple and both  $A = B, A \neq 1$  in the second. From Encoding 1, the first of these is present in the PB model in the form of two constraints:

$$e_{A < B} \implies b_{b0} + 2b_{b1} - a_{b0} - 2a_{b1} \geq 1; \quad (14)$$

$$\bar{e}_{A < B} \implies -b_{b0} - 2b_{b1} + a_{b0} + 2a_{b1} \geq 1. \quad (15)$$

## 26:6 Proof Logging for Smart Extensional Constraints

Assuming the bit encodings are correctly defined, propagating  $b_{=1}$  should propagate  $b_{b0}$  and  $\bar{b}_{b1}$ , meaning (14) is reduced to

$$e_{A<B} \implies 1 - a_{b0} - 2a_{b1} \geq 1. \quad (16)$$

Now it might seem that  $\bar{e}_{A<B}$  would then propagate, as we know  $a_{b0} + 2a_{b1}$  is at least 1 due to the domain constraints of the variable  $A$  and so the right-hand side of the implication in (16) must be false. However, because the falsity does not follow from a contradiction intrinsic to the PB itself – in isolation it can be satisfied by having both  $b_{b0}$  and  $b_{b1}$  equal to 0 – unit propagation alone is not strong enough to determine that  $\bar{e}_{A<B}$  must follow. A similar problem holds for the other tuple, where the contradiction arises due to two constraints being incompatible rather than just one being falsified.

Fortunately, we can ensure that the desired constraints do follow by RUP by first explicitly deriving some intermediate proof steps. For a sequence of guesses  $\mathcal{G}$  and newly derived unsupported assignment  $X \neq v$ , instead of simply logging the PB constraint (13) as above, we first log

$$\wedge \mathcal{G} \implies \bar{x}_{=v} + \bar{s}_k \geq 1 \quad (17)$$

for each  $k \in \{1, \dots, |T|\}$ , i.e. first show by RUP that no tuple selector can be set to true without contradiction, and then derive from these the desired constraint (13). So in our running example, we would log

$$\bar{b}_{=1} + \bar{s}_1 \geq 1; \quad \bar{b}_{=1} + \bar{s}_2 \geq 1; \quad \bar{b}_{=1} \geq 1. \quad (18)$$

This is now sufficient for most cases, however, it is still possible to construct examples where even these constraints in the form of (17) do not follow by RUP. Suppose we have four variables  $W, X, Y, Z$ , all with domain  $\{-2, -1, 0\}$ , and then a single smart tuple of the form

$$(W < X), (X < Y), (Y < Z). \quad (19)$$

Clearly this is unsatisfiable, and so in particular  $W = -2$  should be unsupported by the tuple. But the constraint

$$\bar{w}_{=-2} + \bar{s}_1 \geq 1 \quad (20)$$

does not unit propagate to contradiction. This can be seen by considering the PB encoding of the first smart entry constraint,

$$e_{W<X} \iff -2x_{bneg} + x_{b0} + 2w_{bneg} - w_{b0} \geq 1. \quad (21)$$

The negation of (20) will lead to  $e_{W<X}$ ,  $w_{bneg}$ , and  $\bar{w}_{b0}$  being propagated, which would mean the two PB constraints encoding both implication directions in (21) are reduced to

$$-2x_{bneg} + x_{b0} \geq -1; \quad \text{and} \quad 2x_{bneg} - x_{b0} \geq -1. \quad (22)$$

But then no further propagation results from these constraints, which can be seen either by calculating the *slack* value (see Elffers et al. [8]), or simply by observing that the value sets  $\{0, 0\}$ ,  $\{1, 1\}$ ,  $\{0, 1\}$  would each satisfy both constraints if assigned to  $\{x_{bneg}, x_{b0}\}$  and so both 0 and 1 are both supported possibilities for each variable. None of the other constraints contain bit variables for  $W$  and so it is clear that no propagation will result from these either, and hence no contradiction will be reached.

This weak propagation is not surprising given the context given by Gocht et al. [10], where integer linear inequality constraints need to log all the inferences made when enforcing domain consistency and cannot rely on the disallowed values being implicit. Fortunately for us, these same inferences are being made in the solver as part of the smart table propagator anyway, at the tree filtering stage, and so the desired constraints in the form of (17) and then (13) can still be logged providing we have first logged the inferences for binary constraints explicitly. The only difference in proof logging these and the proof logging of equality and inequality constraints described by Gocht et al. [10] is that each inference will be conditional on a tuple selector. For example, if a variable  $B$  has domain with maximum value  $l$  and the filtering process for  $A < B$  updates the domain copy for  $A$  so that  $A < l$ , then we would log

$$\wedge \mathcal{G} \implies \bar{s}_1 + a_{<l} \geq 1; \quad (23)$$

where the flag  $a_{<l}$  can be introduced via reification with corresponding constraints on the bit variables.

We can now be confident we have a complete proof logging procedure. We can show by an inductive argument on the binary representation that the PB constraints that encode the filtering inferences made by simple binary constraints such as (23) will always be RUP, providing they are logged in the same order as they are made by the solver. Then, any unsupported value must be unsupported in every tuple, and hence removed in the filtering process for some smart entry constraint. Given the negation of this removal, i.e. a variable being equal to some unsupported value, if it was removed due to a binary constraint then we have already logged a corresponding contradicting restriction and hence will unit propagate to contradiction. Otherwise, if it was removed due to a unary constraint it will propagate to contradiction immediately as the assignment will contradict a single PB constraint.

### 3.3 A Complete Worked Example for Smart Table Propagation

To demonstrate how this procedure works on a single round of the domain-consistent smart table propagator we will now show all the steps required for the running example. Firstly, we will have the PB constraints encoding that the of the domains of  $A, B, C$  are all  $\{1, 2, 3\}$ , i.e.

$$1 \leq a_{b_0} + 2a_{b_1} + 4a_{b_2} \leq 3; \quad 1 \leq b_{b_0} + 2b_{b_1} + 4b_{b_2} \leq 3; \quad 1 \leq c_{b_0} + 2c_{b_1} + 4c_{b_2} \leq 3. \quad (24)$$

Next we can follow Encoding 1 to encode the smart table constraint. First we have the encodings of the four necessary auxiliary smart entry flags

$$e_{A < B} \iff b_{b_0} + 2b_{b_1} + 4b_{b_2} - a_{b_0} - 2a_{b_1} - 4a_{b_2} \geq 1; \quad (25a)$$

$$e_{A \in \{1,2\}} \implies \bar{a}_{=3} \geq 1; \quad (25b)$$

$$\bar{e}_{A \in \{1,2\}} \implies \bar{a}_{=1} + \bar{a}_{=2} \geq 2; \quad (25c)$$

$$f_{A > B} \iff a_{b_0} + 2a_{b_1} + 4a_{b_2} - b_{b_0} - 2b_{b_1} - 4b_{b_2} \geq 1; \quad (25d)$$

$$f_{A < B} \iff b_{b_0} + 2b_{b_1} + 4b_{b_2} - a_{b_0} - 2a_{b_1} - 4a_{b_2} \geq 1; \quad (25e)$$

$$e_{A=B} \iff \bar{f}_{A > B} + \bar{f}_{A < B} \geq 2; \quad (25f)$$

$$e_{B \geq C} \iff b_{b_0} + 2b_{b_1} + 4b_{b_2} - c_{b_0} - 2c_{b_1} - 4c_{b_2} \geq 0; \quad (25g)$$

Then we can use these to encode the two tuples

$$s_0 \iff e_{A < B} + e_{A \in \{1,2\}} + c_{=3} \geq 3; \quad (25h)$$

$$s_1 \iff e_{A=B} + \bar{a}_{=1} + e_{B \geq C} \geq 3; \quad (25i)$$



(a) The graph for  $P_{\sigma_1}$ , which consists of two trees. (b) The graph for  $P_{\sigma_2}$ , which consists of a single tree.

■ **Figure 1** The constraint graphs for the two sub-CSPs  $P_{\sigma_1}$  and  $P_{\sigma_2}$  represented by the smart tuples  $\sigma_1$  and  $\sigma_2$ . Both graphs are acyclic and thus composed of trees.

And then finally we require

$$s_0 + s_1 \geq 1. \quad (25j)$$

Constraints that define the  $x_{=v}$  flags are also included but are omitted here for brevity.

Now, for the domain-consistent propagator itself, we note that constraint networks of the smart tuples do indeed form acyclic graphs and so the procedure is applicable. This can be seen in Figure 1, where unary constraints are represented as edges to a dummy node. The first tuple is clearly made up of two small trees, and the second consists of a single tree.

Initially all variables have  $\{1, 2, 3\}$  in their domains and the set of unsupported values  $sl$  is set to:

$$sl = \{A : 1, 2, 3; \quad B : 1, 2, 3; \quad C : 1, 2, 3\} \quad (26)$$

To find the set of values supported by  $P_{\sigma_1}$  we find the values supported by each tree making it up. This involves two filtering passes over copies of the domain, working first from the root outwards, and then back again.

Taking the left tree first, on the first pass 3 is removed from the domain copy for  $A$  and 1 from the domain of  $B$  due to  $A < B$ . So we would log

$$\bar{s}_0 + \bar{a}_{<3} \geq 1; \quad \bar{s}_0 + \bar{b}_{>1} \geq 1; \quad (27)$$

as these inferences were made due to a binary constraint. No further inferences are made on the second pass. This leaves the values

$$\{A : 1, 2; \quad B : 2, 3\} \quad (28)$$

supported by the first tree, and so these are removed from  $sl$ . Filtering the other tree simply reduces the domain copy for  $C$  to  $\{3\}$ , and so  $C = 3$  is removed from  $sl$ . No inferences need to be logged here as this is a unary constraint. We now have

$$sl = \{A : 3; \quad B : 1; \quad C : 1, 2\} \quad (29)$$

Moving on to the second tuple, no inference occurs on the first pass due to  $A = B$  nor  $B \geq C$ , but 1 is removed from the (fresh) domain copy for  $A$  due to the unary  $A \neq 1$  entry. Then on the second pass, 1 is now removed from the domain copy for  $B$  due to  $A = B$ , and so we log

$$\bar{s}_1 + \bar{b}_{=1} \geq 1; \quad (30)$$

and then no further inference occurs. So the values occurring in some solution for this tree are

$$\{A : 2, 3; \quad B : 2, 3; \quad C : 1, 2, 3\} \quad (31)$$

and so these can be removed from  $sl$ , leaving

$$sl = \{B : 1\}. \quad (32)$$

Ultimately,  $B = 1$  is the only assignment lacking support in the table after this initial execution of the domain-consistent propagator and the only value to be removed from an actual variable domain. We are then able to log the three constraints in (18) as discussed above, and thus we have successfully maintained the required invariant for pseudo-Boolean proof logging of this propagator.

## 4 Proof Logging for Regular Language Membership Constraints

A regular language provides another compact way of representing a constraint extensionally [19]. It is defined as follows.

► **Definition 2.** *Regular Language Membership Constraint*

Let  $\mathbf{X}$  again be a sequence of finite-domain variables  $\langle X_1, \dots, X_n \rangle$  and let  $M = (Q, \Sigma, \delta, q_0, F)$  denote a deterministic finite automaton (DFA), where

- $Q$  is a set of states;
- $q_0 \in Q$  is the initial state and  $F \subseteq Q$  is the set of accepting states;
- $\Sigma$  is a set of symbols having the domain of every variable in  $\mathbf{X}$  as a subset; and
- $\delta : (Q \times \Sigma) \rightarrow Q$  is the transition function.

A regular language membership constraint  $\text{Regular}(\mathbf{X}, M)$  requires that any sequence of values taken by the variables of  $\mathbf{X}$  must belong to the regular language recognised by  $M$ .

Throughout this section we will assume that the DFA is specified explicitly by a trusted source. Verified compilation to create the constraint from a regular expression is left as future work.

### 4.1 PB Encodings for Regular Language Membership Constraints

The **Regular** constraint is similarly easy to encode using auxiliary PB variables. If we let  $M = (Q, \Sigma, \delta, q_0, F)$  denote a deterministic finite automaton (DFA) and we impose a regular constraint using this on  $n$  variables, we simply need to define flags  $r_{ij}$  that are set to true if and only if the automaton is in the  $j$ th state at position before the  $(i + 1)$ th symbol of the sequence is processed (including an additional set of flags  $r_{nj}$  to denote the final state).

► **Encoding 2.** *Regular Language Membership PB Encoding*

<pre> 1: for all <math>i \in \{0, \dots, n\}</math> 2:   <math>r_{i0} + \dots + r_{i Q -1} \geq 1</math> 3:   <math>-r_{i0} - \dots - r_{i Q -1} \geq -1</math> 4: for all <math>i \in \{0, \dots, n-1\}, q \in Q, v \in \Sigma</math> 5:   if <math>\delta(q, v)</math> is an allowed transition 6:     <math>r_{iq} \wedge x_{i=v} \implies r_{i+1\delta(q,v)} \geq 1</math> 7:   else 8:     <math>\bar{r}_{iq} + \bar{x}_{i=v} \geq 1</math> 9: <math>r_{00} \geq 1</math> 10: <math>\sum_{f \in F} r_{(n)f} \geq 1</math> </pre>
---

This is a simpler version of the stronger SAT encoding for **Regular** given by Bacchus [1].



## 4.2 Justifying Regular Propagation

As with `SmartTable`, the `Regular` propagator enforces domain consistency, and so our goal once again is to log

$$\wedge \mathcal{G} \implies \bar{x}_{i=v} \geq 1 \quad (33)$$

where  $\mathcal{G}$  is the current sequences of guesses made by the solver, and  $X_i = v$  is an assignment that is not supported by the regular language membership constraint. In particular,  $X_i = v$  is not supported when there exists no sequence of symbols belonging to the language recognised by  $M$  that has  $v$  as the  $i_{th}$  symbol.

It is trivial to establish that this inference will not immediately follow by unit propagation, which is unsurprising given the relative simplicity of the encoding. Just as we did for `SmartTable`, we will have to log some additional facts during the execution of the propagation algorithm.

We will consider here the original domain-consistent `Regular` propagator by Pesant [19], on a constraint over a sequence of  $n$  variables  $\mathbf{X} = \langle X_1 \dots, X_n \rangle$  and associated with a DFA  $M = (Q, \Sigma, \delta, q_0, F)$ . This works by maintaining a layered, directed multigraph that has  $n + 1$  layers, with the  $i_{th}$  layer having  $|Q|$  nodes  $q_0^i, \dots, q_{|Q|-1}^i$ . The edges in this graph are labelled with values in  $\Sigma$ , and are required to respect several properties that are satisfied from the outset and then maintained through propagation. These are:

1. Edges can only appear between nodes in consecutive layers.
2. An edge labelled with the value  $v$  can only be included between nodes  $q_k^i$  and nodes  $q_l^{i+1}$  if there exists a transition between the states numbered  $k$  and  $l$  in the DFA for a  $v$ , with  $v$  currently in the domain of  $X_i$ .
3. Furthermore, every edge included must appear in a path from the node  $q_0^0$  (representing the initial state) to a node representing a final state in the last layer.

The graph is constructed at the start of the solving process, and then updated incrementally during propagation to reflect changes in the variable domains and preserve the required properties. It is clear then that once this is achieved, an assignment  $X_i = v$  loses support on the regular constraint exactly when there are no edges labelled with  $v$  between nodes in the  $i_{th}$  and  $(i + 1)_{th}$  layers.

The strategy for proof logging is then as follows. Every time an edge  $(q_k^i, q_l^{i+1})$  labelled with the symbol  $v$  is removed by the propagation algorithm (or excluded when the graph is first built), we log the PB constraint

$$\wedge \mathcal{G} \implies \bar{r}_{ik} + \bar{x}_{i=v} \geq 1, \quad (34)$$

which can be interpreted as saying: “given these guesses, we can’t both be in state  $k$  after we’ve processed the first  $i - 1$  symbols, and have the  $i_{th}$  symbol be  $v$ ”. If we do this, then when an assignment  $X_i = v$  loses support we will have logged a constraint in the form of (34) for all  $r_{iq}$  where  $\delta(q, v)$  is an allowed transition. Thus, the negation of (33) will result in  $\bar{r}_{iq}$  being propagated for all  $q \in Q$ , and then this will contradict one of the constraints specified by the second line of Encoding 2, which says that at least one  $r_{iq}$  must be set to 1. So with these constraints in place we would be able to log our desired constraint (33) by RUP.

The problem is therefore reduced to that of deriving the constraints in the form of (34). It turns out that these will either follow immediately by RUP, or require additional justification depending on how the corresponding edge elimination is inferred by the propagation.

Edges are removed by the `Regular` propagator in one of three ways. Firstly, when an assignment  $X_i = v$  has already been removed, either because another value has been guessed or because it has been eliminated by a propagator for another CP constraint, all the

edges extending from the  $i_{th}$  layer labelled with value  $v$  are also removed. In this case the corresponding constraints in (34) can be logged by RUP, as  $x_{i=v}$  will immediately contradict the guesses or previous inferences.

Secondly, when a particular node loses all of its outgoing edges, none of its incoming edges can be part of a valid path and so should be removed. These removals occur recursively in a backward pass (Algorithm 3 from Pesant [19]), so if a removed incoming edge was the last outgoing edge of a node in a previous layer then the incoming edges of that previous node are also removed, and so on. In this case too, the constraints in the form of (34) follow by RUP. This can be shown inductively. Suppose all of the outgoing edges for a node  $q_l^i$  have been removed and the corresponding constraints in the form of (34) have already been logged. Then for a given incoming edge  $(q_k^{i-1}, q_l^i)$  labelled with the value  $v$ , the negation of the corresponding constraint in the form of (34) will entail  $r_{ik}$ , and since  $\delta(q_k, v)$  must be  $q_l$ ,  $r_{il}$  will then unit propagate by one of the constraint specified by line 6 of Encoding 2. Then, since we have already eliminated all possible outgoing edges, we will propagate  $\bar{x}_{i=v'}$  for all  $v' \in \Sigma$ , which will contradict the encoding of the variable  $X_i$ , as it must take at least one value.

Finally, in the other direction, when a particular node loses all of its incoming edges, similarly none of its outgoing edges can be part of a valid path and so should be removed. These removals also occur recursively, this time in a forward pass (Algorithm 4 from Pesant[19]). In contrast to the previous two cases, this inference requires more justification as the desired constraint will not always follow by RUP, as is the case for (45) in the worked example below. So when a state  $q_l^i$  loses all of its incoming edges we first need to log the constraints

$$\wedge \mathcal{G} \implies \bar{r}_{i-1k} + \bar{r}_{il} \geq 1 \quad (35)$$

for each  $k \in \{1, \dots, k\}$ , which intuitively say that no previous state could lead to this state, before logging constraints in the form of (34) for each outgoing edge. These constraints in the form of (35) will all follow by RUP, since for every  $v \in \Sigma$  we either have a constraint

$$\wedge \mathcal{G} \implies \bar{r}_{i-1k} + \bar{x}_{i-1=v} \geq 1; \quad (36)$$

from those in the form of (34) logged at the previous stage of the recursion; or there is a constraint

$$\wedge \mathcal{G} \implies \bar{x}_{i-1=v} \geq 1; \quad (37)$$

logged during earlier search/propagation (when this node was the first layer of the recursion); or else there is a constraint

$$\bar{r}_{i-1k} + x_{i-1=v} \geq 1; \quad (38)$$

present in the PB model. In all cases, the negation of (35) will result in  $\bar{x}_{i-1=v}$  for every  $v \in \Sigma$ , giving a contradiction.

Once all the constraints in the form of (35) have been derived, the desired constraint in the form of (34) will definitely follow by RUP, as its negation would then propagate  $\bar{r}_{i-1k}$  for each  $k \in Q$ , contradicting one of the constraints on the second line of Encoding 2.

Putting all of this together, we can emit full justification using RUP for every edge elimination performed by **Regular**, and thus we are able to derive the constraint in the form of (33) required by the proof logging invariant.

It should be noted that the same justifications apply when building the graph for the first time, as it can be conceptually viewed as eliminating edges from the complete layered graph with  $n + 1$  layers and  $|Q|$  nodes in each layer. The deletion occurs in two passes. Firstly it eliminates all outgoing edges from every node in layer 0 that does not correspond to the initial state, recursing for each subsequent layer. Then secondly, it eliminates all incoming edges from every node in layer  $n + 1$  that does not correspond to a final state, similarly recursing for each previous layer.

Note finally that stronger encodings of the **Regular** constraint are possible, and these would not require as much proof logging during search. However, these encodings are not so obviously correct, which is a drawback since the constraint encoding process is not (currently) verified. It is not even clear that such encodings would give more efficient proof verification, since they would involve having a larger set of active PB constraints for the entire verification process.

### 4.3 A Complete Worked Example for Regular Propagation

Once again, to demonstrate how the proof logging procedure works more concretely, we will now show a worked example, including the initial building of the graph and a round of domain-consistent propagation. We will take the simple Example 1 from Pesant [19], of a regular language membership constraint on five variables  $X_1, \dots, X_5$  each with domain  $\{0, 1, 2\}$ , and using the DFA  $M$  as shown in Figure 2a.

For the PB encoding, we will omit the constraints encoding the domains and equals flags and focus on the constraints present in Encoding 2. We can see that we have five states numbered  $0 \dots 4$ , and five variables in the sequence, so the model would first define:

$$r_{00} + \dots + r_{04} \geq 1; \quad \dots \quad r_{50} + \dots + r_{54} \geq 1; \quad (39)$$

$$-r_{00} - \dots - r_{04} \geq -1; \quad \dots \quad -r_{50} - \dots - r_{54} \geq -1; \quad (40)$$

saying that we have to have exactly one of the state-position flags set for each position. Next for each position  $i$  we would define eight PB constraints corresponding to the eight valid transitions:

$$\begin{aligned} \bar{r}_{i0} + \bar{x}_{i=0} + r_{i+11} \geq 1; & \quad \bar{r}_{i0} + \bar{x}_{i=2} + r_{i+14} \geq 1; & \quad \bar{r}_{i1} + \bar{x}_{i=0} + r_{i+11} \geq 1; \\ \bar{r}_{i1} + \bar{x}_{i=1} + r_{i+12} \geq 1; & \quad \bar{r}_{i2} + \bar{x}_{i=0} + r_{i+13} \geq 1; & \quad \bar{r}_{i2} + \bar{x}_{i=1} + r_{i+12} \geq 1; \\ \bar{r}_{i3} + \bar{x}_{i=0} + r_{i+13} \geq 1; & \quad \bar{r}_{i4} + \bar{x}_{i=2} + r_{i+14} \geq 1; \end{aligned} \quad (41)$$

along with constraints of the form

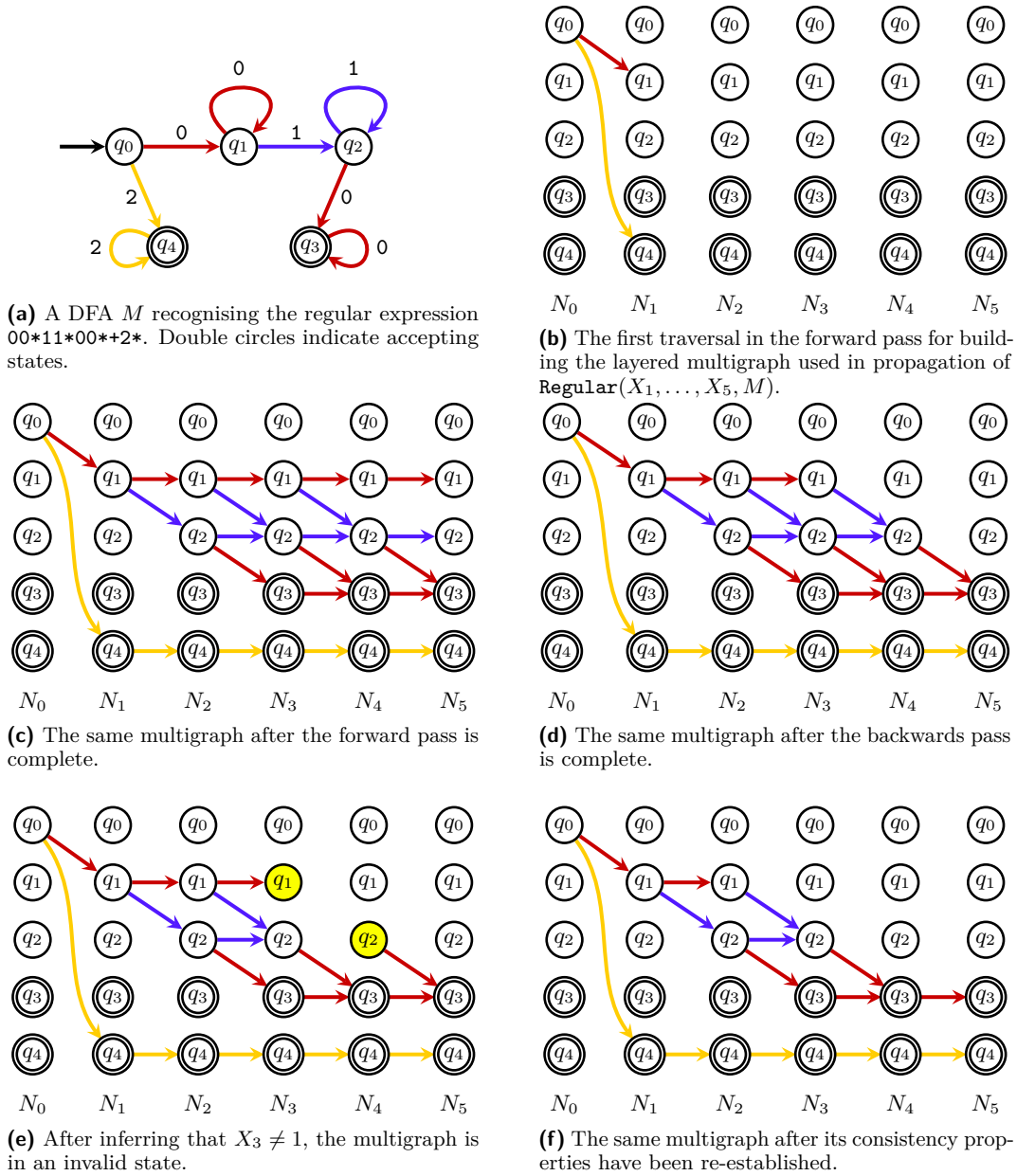
$$\bar{r}_{iq} + \bar{x}_{i=v} \geq 1; \quad (42)$$

for the remaining seventeen invalid transitions. Finally, we would have constraints saying that we have to start in an initial state and end in a final state:

$$r_{00} \geq 1; \quad r_{53} + r_{54} \geq 1. \quad (43)$$

Propagation depends on us having first built a valid graph, with nodes as shown in Figure 2b. As mentioned, this occurs in two passes, with the forward pass collecting nodes that can be reached from the initial state. This corresponds to removing outgoing edges only reachable from nodes other than the initial state.

So to begin with, we traverse the two edges incident to node  $q_0^0$ , reaching nodes  $q_1^1$  and  $q_4^1$  and thereby eliminating all other edges. These are the two edges shown in Figure 2b. Since we are eliminating outgoing edges we are effectively in the third case of the proof logging



■ **Figure 2** Propagation of the regular expression  $00^*11^*00^{**}2^*$  on the variables  $X_1, \dots, X_5$ , showing the initial state, and then the effect of propagating  $X_3 \neq 1$ .

## 26:14 Proof Logging for Smart Extensional Constraints

procedure from the previous section, however, as there are no previous states to eliminate we simply need to log constraints in the form of (34) for each of the six eliminated edges:

$$\begin{aligned} \bar{r}_{01} + \bar{x}_{0=0} &\geq 1; & \bar{r}_{01} + \bar{x}_{0=1} &\geq 1; & \bar{r}_{02} + \bar{x}_{0=0} &\geq 1; \\ \bar{r}_{02} + \bar{x}_{0=1} &\geq 1; & \bar{r}_{03} + \bar{x}_{0=0} &\geq 1; & \bar{r}_{04} + \bar{x}_{0=2} &\geq 1; \end{aligned} \quad (44)$$

and these will all follow by RUP because we have  $r_{00} \geq 1$  in the PB model. Continuing, we recursively collect edges reachable from the initial node, consequently eliminating all outgoing edges for any node that is not reached. The result of this is shown in Figure 2c.

At each layer, we can follow this same proof logging procedure for eliminating outgoing edges. For example, we do not collect  $(q_3^2, q_3^3)$  at layer 2, since  $q_3^2$  is not reached as part of the forward pass. So we would like to log

$$\bar{r}_{23} + \bar{x}_{2=0} \geq 1; \quad (45)$$

for this eliminated edge, but this does not follow by RUP. Despite us having logged constraints in the form of (34) for each eliminated incoming edge to  $q_2^3$ , none of these constraints contain either  $r_{23}$  or  $x_{2=0}$ , as they concern the previous layer, and so no further propagation takes place from the negation of (45). Hence, as discussed, we first log

$$\bar{r}_{10} + \bar{r}_{23} \geq 1; \quad \bar{r}_{11} + \bar{r}_{23} \geq 1; \quad \bar{r}_{12} + \bar{r}_{23} \geq 1; \quad \bar{r}_{13} + \bar{r}_{23} \geq 1; \quad \bar{r}_{14} + \bar{r}_{23} \geq 1; \quad (46)$$

which all do follow by RUP, and then our constraint (45) can be logged.

After this process is complete, the backwards pass takes place, first eliminating incoming edges from any nodes in the last layer corresponding to non-final states, and then recursively eliminating all incoming edges from any node that lost all of its outgoing edges. The complete, correctly initialised state of the graph after this backwards pass is shown in Figure 2d.

Following the proof logging procedure for eliminating incoming edges, in this direction we simply need to log a constraint in the form of (34) for each, and these will follow by RUP, as discussed.

At this point, some basic inferences about variable value pairs can already be made, such as  $X_0 \neq 1$  and  $X_4 \neq 1$ . Due to all the constraints that have already been logged, eliminating all possible corresponding edges from consideration, the required proof logging invariant constraint for each will follow by RUP. This concludes the graph initialisation for the **Regular** propagator, and from this point onwards the structure can be updated incrementally and restored upon backtrack, as described in detail by Pesant [19].

We will now demonstrate one such incremental update, corresponding to an execution of domain-consistent propagation for **Regular**. Suppose through the course of the computation, the assignment  $X_3 = 1$  loses support. This means the edges  $(q_1^3, q_2^4)$  and  $(q_2^3, q_2^4)$  are immediately removed from the graph, and we log the RUP constraints:

$$\bar{r}_{13} + \bar{x}_{3=1} \geq 1; \quad \bar{r}_{23} + \bar{x}_{3=1} \geq 1. \quad (47)$$

At this point the state of the graph is as shown in Figure 2e, with the two highlighted nodes having lost all of their outgoing and incoming edges respectively.

The recursive incremental update is then executed for each of these, further removing the edge  $(q_1^2, q_1^3)$  in a backwards pass, and so logging the RUP constraint

$$\wedge \mathcal{G} \implies \bar{r}_{31} + \bar{x}_{2=0}. \quad (48)$$

The edge  $(q_2^4, q_3^5)$  is also removed, but because this is removed in a forward pass, we first log

$$\wedge \mathcal{G} \implies \bar{r}_{30} + \bar{r}_{42} \geq 1; \quad \wedge \mathcal{G} \implies \bar{r}_{31} + \bar{r}_{42} \geq 1; \quad \dots \quad \wedge \mathcal{G} \implies \bar{r}_{34} + \bar{r}_{42} \geq 1; \quad (49)$$

before logging the required

$$\wedge \mathcal{G} \implies \bar{r}_{42} + \bar{x}_{2=0}. \quad (50)$$

No further nodes lose their last incoming or outgoing edge as a result of this, and so the required properties (and hence domain consistency) have been re-established on the graph. The final consistent state is shown in Figure 2f.

## 5 Implementation and Validation

We have implemented both of these proof logging propagation algorithms inside the open-source *Glasgow Constraint Solver* [17]. Our implementation adds two new constraints `Regular` and `SmartTable`, with specific functionality to produce the required encodings and justifying propagators. They can therefore be used in conjunction with the rest of the constraint types already available in the solver.

We validated the implementations by first modelling some key examples. For `SmartTable`, these included the representation of lexicographic ordering problems, and problems where at most one variable takes a value, as given by Mairy et al. [16], as well as the illustrative examples from Section 3. For `Regular`, we implemented both examples 1 and 2 from Pesant [19].

We carried out further experimental validation by generating random (acyclic) smart tables, and random DFAs on sequences of up to five variables and solving the corresponding single-constraint problems. In all tests and examples, the solver produced an OPB model and proof files, and we checked these using the VeriPB proof checker. We also found that although proof logging incurs an obvious performance cost, the observed overheads were not unreasonable, giving a slowdown factor of between 2 and 10. As proofs are written currently written directly to disk, this can be very hardware dependent, and also dependent on how optimised the propagator implementation is. We decided to leave further engineering and optimisation of proof writing to future work.

During development some very subtle bugs in both propagator implementations that had eluded conventional testing were caught immediately by proof logging. For example, in an incremental version of the `Regular` propagator, an unsound inference was being made due to mixing up variable names, but in such a way that a situation where this unsound inference would actually lead to an incorrect solution was extremely rare (only in specially constructed instances, created once we were made aware of the bug due to proof logging). After correcting bugs such as these, all proofs were certified by VeriPB as being correct.

## 6 Conclusion

We have shown that we can efficiently justify all the reasoning that could possibly be carried out for two families of smart extensional constraints. An interesting observation is that justifying this reasoning required only the RUP rule, and this rule was only to provide hints of algorithmic steps that were already being carried out by the propagators. In effect, we are logging a sequence of “lookahead to see immediate contradiction” steps. We did not require any explicit cutting planes derivations, and although we relied upon pseudo-Boolean

constraints to make it simple to express reifications and negations, in principle everything we did should also be possible in a weaker proof encoding and proof system such as CNF and DRAT. The only caveat is that we *do* rely upon strong propagation properties for encoded integer variables, which would limit approaches based upon Boolean satisfiability to integer variables with very small domains. This is in contrast to constraints like `AllDifferent` and `Linear`, which cannot be logged efficiently in resolution-based approaches [8, 10].

We expect that other global constraints, even those with complex propagation algorithms will be similarly feasible to proof log using this technique. Richer smart tables, such as those with offset or ternary restrictions [4], also fit into the framework given in Section 3.2, although the justification of the filtering inferences for each restriction may require additional cutting planes steps. Furthermore, the propagation of “Multi-Valued Decision Diagram”-based constraints [14] can be viewed as a generalisation of the techniques used for the regular language membership constraint, and so it seems very plausible that a similar proof logging methodology as demonstrated in this paper could work here. This bodes well for being able to provide auditable solving for most global constraints that might occur in a modern constraint solver.

---

## References

- 1 Fahiem Bacchus. GAC Via Unit Propagation. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, Lecture Notes in Computer Science, pages 133–147, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-74970-7\_12.
- 2 Seulkee Baek, Mario Carneiro, and Marijn J. H. Heule. A flexible proof format for SAT solver-elaborator communication. In *Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part I*, pages 59–75, Berlin, Heidelberg, 2021. Springer-Verlag. doi:10.1007/978-3-030-72016-2\_4.
- 3 Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified Symmetry and Dominance Breaking for Combinatorial Optimisation, March 2022. doi:10.48550/arXiv.2203.12275.
- 4 Frédéric Boussemart, Christophe Lecoutre, and Cédric Piette. XCSP3: an integrated format for benchmarking combinatorial constrained problems. *CoRR*, abs/1611.03398, 2016. arXiv:1611.03398.
- 5 Chiu Wo Choi, Warwick Harvey, J. H. M. Lee, and Peter J. Stuckey. Finite domain bounds consistency revisited. In *AI 2006: Advances in Artificial Intelligence, 19th Australian Joint Conference on Artificial Intelligence, Hobart, Australia, December 4-8, 2006, Proceedings*, pages 49–58, 2006. doi:10.1007/11941439\_9.
- 6 W. Cook, C.R. Coullard, and Gy. Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987. doi:10.1016/0166-218X(87)90039-4.
- 7 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 220–236, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-63046-5\_14.
- 8 Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying All Differences Using Pseudo-Boolean Reasoning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1486–1494, April 2020. doi:10.1609/aaai.v34i02.5507.
- 9 Stephan Gocht. *Certifying Correctness for Combinatorial Algorithms by Using Pseudo-Boolean Reasoning*. PhD thesis, Lund University, Sweden, 2022. URL: <https://lup.lub.lu.se/record/3550cb96-83d5-4fc7-9e62-190083a3c10a>.

- 10 Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An Auditable Constraint Programming Solver. In Christine Solnon, editor, *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CP.2022.25.
- 11 E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for cnf formulas. In *2003 Design, Automation and Test in Europe Conference and Exhibition*, pages 886–891, 2003. doi:10.1109/DATE.2003.1253718.
- 12 Marijn J. H. Heule, Warren A. Hunt, and Nathan Wetzler. Verifying refutations with extended resolution. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24*, pages 345–359, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-38574-2\_24.
- 13 Marijn J.H. Heule, Warren A. Hunt, and Nathan Wetzler. Trimming while checking clausal proofs. In *2013 Formal Methods in Computer-Aided Design*, pages 181–188, 2013. doi:10.1109/FMCAD.2013.6679408.
- 14 Samid Hoda, Willem-Jan van Hoeve, and J. N. Hooker. A systematic approach to MDD-Based constraint programming. In David Cohen, editor, *Principles and Practice of Constraint Programming – CP 2010*, pages 266–280, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-15396-9\_23.
- 15 Mikael Z Lagerkvist and Gilles Pesant. Modeling irregular shape placement problems with regular constraints. In *First workshop on bin packing and placement constraints BPPC’08*, 2008.
- 16 Jean-Baptiste Mairy, Yves Deville, and Christophe Lecoutre. The Smart Table Constraint. In Laurent Michel, editor, *12th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2015)*, Lecture Notes in Computer Science, pages 271–287, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-18008-3\_19.
- 17 Ciaran McCreesh and Matthew McIlree. The Glasgow Constraint Solver. GitHub repository, 2023. URL: <https://github.com/ciaranm/glasgow-constraint-solver>.
- 18 Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation = lazy clause generation. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 544–558. Springer, 2007. doi:10.1007/978-3-540-74970-7\_39.
- 19 Gilles Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In Mark Wallace, editor, *10th International Conference on Principles and Practice of Constraint Programming (CP 2004)*, Lecture Notes in Computer Science, pages 482–495, Berlin, Heidelberg, 2004. Springer. doi:10.1007/978-3-540-30201-8\_36.
- 20 H el ene Verhaeghe. *The extensional constraint*. PhD thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2021.
- 21 H el ene Verhaeghe, Christophe Lecoutre, Yves Deville, and Pierre Schaus. Extending Compact-Table to Basic Smart Tables. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming*, volume 10416, pages 297–307. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-66158-2\_19.








# Improving Conflict Analysis in MIP Solvers by Pseudo-Boolean Reasoning

Gioni Mexi   

Zuse Institute Berlin, Germany

Timo Berthold   

Fair Isaac Deutschland GmbH, Berlin, Germany

TU Berlin, Germany

Ambros Gleixner   

HTW Berlin, Germany

Zuse Institute Berlin, Germany

Jakob Nordström   

University of Copenhagen, Denmark

Lund University, Sweden

---

## Abstract

Conflict analysis has been successfully generalized from Boolean satisfiability (SAT) solving to mixed integer programming (MIP) solvers, but although MIP solvers operate with general linear inequalities, the conflict analysis in MIP has been limited to reasoning with the more restricted class of clausal constraint. This is in contrast to how conflict analysis is performed in so-called pseudo-Boolean solving, where solvers can reason directly with 0–1 integer linear inequalities rather than with clausal constraints extracted from such inequalities.

In this work, we investigate how pseudo-Boolean conflict analysis can be integrated in MIP solving, focusing on 0–1 integer linear programs (0–1 ILPs). Phrased in MIP terminology, conflict analysis can be understood as a sequence of linear combinations and cuts. We leverage this perspective to design a new conflict analysis algorithm based on mixed integer rounding (MIR) cuts, which theoretically dominates the state-of-the-art division-based method in pseudo-Boolean solving.

We also report results from a first proof-of-concept implementation of different pseudo-Boolean conflict analysis methods in the open-source MIP solver SCIP. When evaluated on a large and diverse set of 0–1 ILP instances from MIPLIB 2017, our new MIR-based conflict analysis outperforms both previous pseudo-Boolean methods and the clause-based method used in MIP. Our conclusion is that pseudo-Boolean conflict analysis in MIP is a promising research direction that merits further study, and that it might also make sense to investigate the use of such conflict analysis to generate stronger no-goods in constraint programming.

**2012 ACM Subject Classification** Theory of computation → Discrete optimization; Mathematics of computing → Solvers

**Keywords and phrases** Integer programming, pseudo-Boolean solving, conflict analysis, cutting planes proof system, mixed integer rounding, division, saturation

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.27

**Funding** The work for this article has been conducted within the Research Campus Modal funded by the German Federal Ministry of Education and Research (BMBF grant numbers 05M14ZAM, 05M20ZBM).

*Jakob Nordström*: supported by the Swedish Research Council grant 2016-00782 and the Independent Research Fund Denmark grant 9040-00389B.

**Acknowledgements** Part of this work was carried out while some of the authors participated in the extended reunion for the program *Satisfiability: Theory, Practice, and Beyond* at the Simons Institute for the Theory of Computing at UC Berkeley in the spring of 2023. This work has also benefited greatly from discussions during the Dagstuhl Seminar 22411 *Theory and Practice of SAT and Combinatorial Solving*.



© Gioni Mexi, Timo Berthold, Ambros Gleixner, and Jakob Nordström;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 27; pp. 27:1–27:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

The area of Boolean satisfiability (SAT) solving has witnessed dramatic performance improvements over the last couple of decades, and several techniques from SAT have also inspired developments for other combinatorial optimization paradigms such as SAT-based and (linear) pseudo-Boolean optimization, constraint programming, and mixed integer programming. In particular, conflict analysis as introduced in the works on *conflict-driven clause learning (CDCL)* [3, 38, 40] ushering in the modern SAT solving revolution has been picked up and generalized in different ways to these more general settings. Interestingly, precursors of this version of conflict analysis and nonchronological backtracking can be traced back all the way to early work in the AI community [47], and related ideas have been used in constraint programming for decades [24, 31]. Our focus in this paper is on conflict analysis in mixed integer programming and pseudo-Boolean optimization, which we proceed to discuss next.

### 1.1 Mixed Integer Programming and Conflict Analysis

The core method of mixed integer programming (MIP) is that a linear programming (LP) relaxation of the problem is fed to an LP solver. If the LP solver finds a solution that assigns real values to integral variables, then either additional cut constraints can be generated that eliminate such solutions, or the problem can be split into subproblems by branching on integer variables, generating new nodes in the search tree. During the solving process infeasible nodes in the search tree are pruned. Unlike in SAT, there can be different reasons for backtracking due to infeasibility of the LP relaxation, node presolving (propagation), or to the current objective value of the relaxed problem being worse than the best solution found so far (branch-and-bound). MIP solvers employ a multitude of further techniques such as symmetry detection, disjoint subtree detection, restarts, et cetera. For a comprehensive description of MIP solving we refer the reader to, e.g., [2].

The use of SAT techniques in MIP solvers has been a fruitful direction of research over the last decades. Specifically, CDCL conflict analysis has proven to be a useful tool to enhance the performance of MIP solvers by learning constraints from infeasibilities detected by propagation or from the LP relaxation [1, 44, 48]. However, SAT and MIP solvers differ fundamentally in how they explore the search space, in that SAT solvers search depth-first, maintaining only the current state of the search, whereas in MIP the search tree is generated in a “best-first” manner based on careful analysis on search statistics such as dual bounds and integrality of LP solutions to subproblems. These differences make it harder for MIP solvers to profit from conflict analysis, and so in contrast to SAT solving, for which this technique is absolutely crucial, in MIP solving it plays more of a supplemental if still highly valuable role.

Although the setting is different, the *graph-based conflict analysis* [1] used to learn from infeasibilities in MIP is very similar to the classic SAT approach. First, a partial assignment is extracted that consists of branching decisions and implications that led to the infeasibility. If the LP relaxation is infeasible, the information which bound changes led to infeasibility is gathered from the non-zero duals of the LP. Next, a directed acyclic graph is constructed that encodes information about the conflict, in that source nodes correspond to branching decisions, non-source nodes encode implications, and the sink node represents the infeasibility. Each cut in this graph that separates the source nodes from the sink is a valid constraint. It is important to note that all implications correspond to clausal constraints, and so this conflict analysis operates not on the linear constraints of the problem but on clauses extracted from these linear constraints. (There are also methods that can learn general linear constraints

from infeasibilities, one notable example being *dual-proof analysis* [48], but this technique is limited to conflicts arising from infeasibility of the LP relaxation and does not analyze or strengthen the partial assignment that led to infeasibility.)

## 1.2 Pseudo-Boolean Solving and Conflict Analysis

Pseudo-Boolean (PB) solving is another approach specific to integer linear programs with only binary variables, or 0–1 ILPs, which are referred to as (linear) pseudo-Boolean formulas in the PB solving literature. While MIP solvers find real-valued solutions and try to push such solutions closer and closer to integrality, PB solvers follow the SAT approach of considering only Boolean assignments and trying to extend partial assignments to more and more variables without violating any constraints. Just as in SAT, this search is performed in a depth-first manner.

Some PB solvers stick very closely to SAT in that they immediately translate the 0–1 ILP into conjunctive normal form (CNF) using auxiliary variables and then run a standard CDCL SAT solver [21, 39, 43]. Another approach, which is what is of interest in the context of this work, is to extend the solvers to reason natively with 0-1 linear inequalities [12, 46, 34, 23]. Such *conflict-driven pseudo-Boolean solvers* have the potential to run exponentially faster than CDCL-based solvers, since their conflict analysis method is exponentially stronger than that used in CDCL SAT solvers.

Since it is crucial for our work to understand the differences between conflict analysis in MIP and PB solvers, let us try to provide a somewhat simplified exposition of PB solving in a language that is meant to convey a MIP perspective (and where what follows below is heavily indebted to [19]). During the search phase, the pseudo-Boolean solver always first tries to extend the current partial solution with any variable assignments that are propagated by some linear inequality. When no further propagations are possible, the solver chooses some unassigned variable and makes a *decision* to assign this variable 0 or 1, after which it again turns to propagation. This cycle of decisions and propagations repeats until either a satisfying assignment is found or some 0–1 linear inequality  $C$  is violated. In the latter case, the solver switches to the conflict analysis phase, which works as follows:

1. The linear inequality  $R$  responsible for propagating the last variable  $x$  in  $C$  to the “wrong value” from the point of view of  $C$  is identified; this inequality  $R$  is referred to as the *reason constraint* for  $x$ .
2. A *division* or *saturation* rule is applied to  $R$  to generate a modified inequality  $R^*$  that propagates  $x$  tightly to its assigned value even when considered over the reals.
3. A new linear constraint  $D$  is computed as the smallest integer linear combination of  $R^*$  and  $C$  for which the variable  $x$  cancels and is eliminated. It is not too hard to show that it follows from the description above that this constraint  $D$  is violated by the current partial assignment of the solver with the value of  $x$  removed, and we can set  $C := D$  and go to step 1 again.

This continues until a termination criterion analogous to the *unique implication point (UIP)* notion used in SAT solving leads to  $D$  being declared as the learned constraint. At this point, the solver undoes further assignments in reverse chronological order until  $D$  is no longer violated, and then switches back to the search phase. We refer the reader to the chapter [11] for a more detailed description of conflict-driven pseudo-Boolean solving (and to the handbook [8] for an in-depth treatment of SAT and related topics in general).

In contrast to MIP conflict analysis, the algorithm described above is not phrased in terms of the conflict graph, but focuses on the syntactic *resolution* method [9, 17, 16, 42] employed in CDCL conflict analysis and harnesses the observation by Hooker [29, 30] that

resolution can be understood as a cut rule and extended to 0–1 integer linear inequalities. The conflict-graph-based analysis in MIP does not operate on the reason constraints  $R$  as described above, but instead on disjunctive clauses extracted from these constraints. It is not hard to prove formally (appealing to [4, 14, 28]) that this incurs an exponential loss in reasoning power compared to performing derivations on the linear constraints themselves.

In practice, however, it seems fair to say that current pseudo-Boolean solvers do not quite deliver on this promise of exponential gains in performance. Although there are specific problem domains where PB solvers outperform even commercial MIP solvers [35, 45], evaluations over larger sets of benchmarks [5, 19, 20] have demonstrated that the open-source MIP solver SCIP [7] tends to be clearly more effective in solving pseudo-Boolean optimization problems, and is also quite competitive for decision problems. This is especially so for some decision problems that are in some sense close to LP-infeasibility – such problems are almost trivial for MIP solvers, but can be extremely challenging for pseudo-Boolean solvers [22].

### 1.3 Questions Studied in This Work and Our Contributions

Since mixed integer programming solvers and pseudo-Boolean solvers approach 0–1 integer linear problems from quite different angles, and seem to have complementary performance profiles, it is natural to ask whether techniques from one of the paradigms can be used to improve solvers based on the other paradigm.

Some MIP-inspired approaches have been integrated with success in SAT and PB solvers, perhaps most recently in [19], where the PB solver ROUNDINGSAT [23] makes careful use of the LP solver SOPLEX [7] to detect infeasibility of LP relaxations and generate cut constraints (though this paper also raises many questions that would seem to merit further study). However, in the other direction we are not aware of any work trying to harness state-of-the-art techniques from pseudo-Boolean solving to improve the performance of MIP solvers.

In this work, we consider how the clausal conflict analysis in MIP solvers can be replaced by pseudo-Boolean reasoning, focusing on 0–1 integer linear programs. A key difference between the clausal and pseudo-Boolean conflict analysis methods is that in the latter algorithm the linear reason constraint  $R$  propagating a variable assignment might need to be modified, or *reduced*, to another constraint  $R^*$  that propagates tightly also over the reals (which is already guaranteed to hold if  $R$  is a clausal constraint). Viewed from a MIP perspective, this *reduction* step deriving  $R^*$  from  $R$  can be seen to be an application of one of two specific cut rules, where saturation-based reduction as in [34] corresponds to coefficient tightening and division-based reduction as in [23] uses Chvátal-Gomory cuts.

This observation raises the question of whether more general cuts could also be used to obtain other, and potentially more powerful, reduction methods for pseudo-Boolean conflict analysis. The answer turns out to be yes, and, in particular, we introduce a new reduction algorithm utilizing mixed integer rounding (MIR) cuts [27, 37]. A theoretical comparison of the MIR-based reduction rule with the reduction methods currently used in PB solvers show that MIR-based reduction dominates the division-based method that is considered to be state of the art in pseudo-Boolean solving, while saturation-based reduction and MIR-based reduction appear to be incomparable.

We implement pseudo-Boolean conflict analysis for 0–1 ILPs in the MIP solver SCIP, including all three reduction methods discussed above, and compare these different flavours of PB conflict analysis with each other as well as with clausal MIP conflict analysis on a large benchmark set consisting of pure 0–1 ILP instances from MIPLIB 2017. We find that the MIR-based pseudo-Boolean conflict analysis has the best performance, beating not only the

conflict analysis methods in the PB literature but also the standard clausal conflict analysis in SCIP. Interestingly, the new method is better measured not only in terms of number of nodes in the search tree, but also in terms of the number of instances solved, even though we only provide a proof-of-concept implementation lacking many of the optimizations that would be included in a full integration of this method into the SCIP codebase. Although our experimental data cannot provide conclusive evidence as to what causes this improved performance, we observe that the constraints learned from pseudo-Boolean conflict analysis seem more useful in that they take part more actively in propagations than constraints obtained by clausal conflict analysis.

## 1.4 Organization of This Paper

After reviewing preliminaries in Section 2, we give a detailed description of clausal and pseudo-Boolean conflict analysis for 0–1 integer linear programs in Section 3, including a discussion of the reduction methods found in the PB literature and our new version using mixed integer rounding cuts, and study how the different reduction rules compare in theory. In Section 4 we present our experimental results. We conclude the paper in Section 5 by summarizing our work and discussing direction for future research.

## 2 Preliminaries and Notation

Let  $n \in \mathbb{Z}_{>0}$ , and  $\mathcal{N} := [1, \dots, n]$ . We let  $x_i$  denote Boolean (i.e.,  $\{0, 1\}$ -valued) variables and  $\ell_i$  denote literals, which can be either  $x_i$  or its negation  $\bar{x}_i = 1 - x_i$ . A *pseudo-Boolean constraint* is a 0–1 integer linear inequality

$$\sum_{i \in \mathcal{N}} a_i \ell_i \geq b, \quad (1)$$

where we can assume without loss of generality that  $a_i \in \mathbb{Z}_{\geq 0}$  for all  $i \in \mathcal{N}$  and  $b \in \mathbb{Z}_{\geq 0}$  (so-called *normalized form*). We can convert “ $\leq$ ”-constraints with 0–1 variables to “ $\geq$ ”-constraints by multiplying the constraint by  $-1$  and normalizing, i.e., replacing the variables by literals. Moreover, equalities “ $=$ ” can be viewed as syntactic sugar for two opposing inequalities, which can also be transformed into normalized pseudo-Boolean format. In particular, every pure 0–1 integer linear program can be transformed to a normalized pseudo-Boolean representation. Note that in Section 3 we develop our theory and algorithms using normalized PB constraints for simplicity of exposition. However, in our actual implementation and experiments (described in Section 4), we directly operate on general linear constraints.

A (partial) assignment  $\rho$  is a (partial) map from variables to  $\{0, 1\}$ , which is extended to literals by respecting the meaning of negation. We call a literal  $\ell_i$  *falsified* or *false* if  $\rho(\ell_i) = 0$  and *satisfied* or *true* if  $\rho(\ell_i) = 1$ . If  $\rho$  is undefined for a literal, we call the literal *unassigned* or *free*. A constraint is satisfied under some partial assignment  $\rho$  if the respective inequality holds, independently of which values the unassigned literals take, and is falsified if no assignment to the unassigned literals can make the inequality true.

The slack of a PB constraint  $C : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$  under a partial assignment  $\rho$  is defined as  $\text{slack}(C, \rho) := \sum_{\{i \in \mathcal{N} : \rho(i) \neq 0\}} a_i - b$ . With this definition,  $C$  is falsified under  $\rho$  if and only if  $\text{slack}(C, \rho) < 0$ . For example the constraint  $C : 2\bar{x}_1 + 2x_2 + 3x_3 \geq 4$  is falsified under the partial assignment  $\rho = \{x_1 = 1, x_2 = 0\}$  since  $\text{slack}(C, \rho) = -1 < 0$ . For a non-falsified constraint  $C$  and an unassigned literal  $\ell_i$  with coefficient  $a_i$ , the constraint propagates  $\ell_i$  if and only if  $\text{slack}(C, \rho) < a_i$ . For instance, the same constraint  $C : 2\bar{x}_1 + 2x_2 + 3x_3 \geq 4$

propagates both variables  $x_2$  and  $x_3$  to 1 under the partial assignment  $\rho = \{x_1 = 1\}$  since  $\text{slack}(C, \rho) = 1$  is strictly smaller than the coefficients of each of the variables. A constraint propagates the assignment of a free variable tightly if the slack under the current partial assignment is 0. For any two pseudo-Boolean constraints  $C$  and  $C'$  and partial assignment  $\rho$  it holds that the slack is subadditive, i.e.,  $\text{slack}(C + C', \rho) \leq \text{slack}(C, \rho) + \text{slack}(C', \rho)$ . The decision level of a literal  $\ell_i$  under a partial assignment  $\rho$  is the number of decisions prior to the fixing of  $\ell_i$ . Note that the first fixing in every decision level is a decision literal.

### 3 Conflict Analysis Algorithms

For simplicity, in this section we present all algorithms in a pseudo-Boolean framework, where all coefficients and constants are integral, and the proofs of correctness that we provide also make crucial use of this fact. It is important to note that this is not the case in the actual implementation in SCIP, which operates with real-valued coefficients and constants. In fact, one of the challenges in implementing pseudo-Boolean conflict analysis in a MIP framework is that careful thought is required to rephrase the algorithms in such a way that they can deal with real-valued data but are still correct. Next, we describe the details of conflict analysis algorithms used in PB solvers and the different techniques that we consider in this paper.

#### 3.1 Clausal Conflict Analysis

To explain the idea of conflict analysis, we first consider the case where all constraints are clauses. Conflict analysis begins at the stage where a conflict clause  $C_{\text{conf}}$  is falsified by the current partial assignment  $\rho$ . Let  $\ell_r$  be the literal in  $C_{\text{conf}}$  that was last propagated to false, and let  $C_{\text{reason}}$  be the reason clause in chronological order that is responsible for the propagation, i.e., we have  $C_{\text{conf}} = C' \vee \ell_r$  and  $C_{\text{reason}} = C'' \vee \bar{\ell}_r$ . Using the resolution rule, we can derive the so-called *resolvent*  $C' \vee C''$  as a new learned clause  $C_{\text{learn}}$ .

Note that, even after removing  $\ell_r$  from the partial assignment  $\rho$ , both  $C'$  and  $C''$  remain falsified:  $C'$  because  $C_{\text{conf}} = C' \vee \ell_r$  and  $\ell_r$  were false, and  $C''$  because  $C_{\text{reason}} = C'' \vee \bar{\ell}_r$  propagated. This is the key invariant of the algorithm: At any point during the algorithm the resolvent is falsified by the remaining partial assignment  $\rho$ .

Hence, we can replace the conflict clause by the resolvent and continue this process. At each step either a propagating literal is removed from  $\rho$  or the learned clause is empty (at which point unsatisfiability is proven) or the last fixed literal is a decision literal. In the third case, we have reached a *first unique implication point* (FUIP) and conflict analysis terminates, with the final resolvent being the learned clause  $C_{\text{learn}}$ . With  $C_{\text{learn}}$  added, propagation on the previous decision level will prevent the last infeasible decisions to happen as the search continues.

It is straightforward to apply this algorithm to problems with 0–1 linear constraints. Suppose  $\sum_{i \in N} a_i \ell_i \geq b$  is the initial conflict constraint falsified under  $\rho$ , then  $\bigvee_{i: a_i > 0 \wedge \rho(\ell_i) = 0} \ell_i$  can be used as initial conflict clause. Analogously, we can extract at each step a reason clause from the linear constraint that propagated the last literal and perform resolution. After terminating at an FUIP, the learned clause can be added as linear constraint to the solver.

#### 3.2 PB Conflict Analysis

As in the clausal version, the main idea of PB conflict analysis is also to find a new constraint that explains the infeasibility of the current subproblem under a falsifying partial assignment. Algorithm 1 shows the base algorithm for all variants of PB conflict analysis considered in this

paper, using the first unique implication point (FUIP) learning scheme. It is initialized with a falsifying partial assignment  $\rho$  and a conflicting constraint  $C_{\text{confl}}$  under  $\rho$ . First, the learned conflict constraint  $C_{\text{learn}}$  is set equal to the conflict constraint  $C_{\text{confl}}$ . In each iteration, we extract the latest literal  $l_r$  from  $\rho$ . If the literal assignment was due to propagation of a constraint and the negated literal  $\bar{l}_r$  occurs in  $C_{\text{confl}}$ , then we extract the reason constraint  $C_{\text{reason}}$  that propagated  $l_r$ . In line 6 we “reduce” the reason constraint such that the resolvent of  $C_{\text{learn}}$  and the reduced reason  $C_{\text{reason}}$  (Line 7) that cancel the last literal  $l_r$  is still falsified under the remaining partial assignment  $\rho$ . The conflict constraint is set to the resolvent and we continue until we reach an FUIP ( $C_{\text{learn}}$  is *asserting*) or we prove  $C_{\text{learn}}$  makes the problem infeasible. We have reached an FUIP if  $C_{\text{learn}}$  would propagate some literal after removing at least all literal assignments in the current decision level from  $\rho$ . We have shown that the problem is infeasible if  $C_{\text{learn}}$  is falsified under an empty partial assignment  $\rho$ . At this point, the learned constraint can be added to the constraint database of our problem to prevent the solver from exploring the same search space again.

■ **Algorithm 1** Pseudo-Boolean Conflict Analysis Algorithm.

---

**Input** : conflict constraint  $C_{\text{confl}}$ , falsifying partial assignment  $\rho$   
**Output** : learned conflict constraint  $C_{\text{learn}}$

---

```

1  $C_{\text{learn}} \leftarrow C_{\text{confl}}$ 
2 while  $C_{\text{learn}}$  not asserting and  $C_{\text{learn}} \neq \perp$  do
3    $l_r \leftarrow$  literal last assigned on  $\rho$ 
4   if  $l_r$  propagated and  $\bar{l}_r$  occurs in  $C_{\text{learn}}$  then
5      $C_{\text{reason}} \leftarrow \text{reason}(l_r, \rho)$ 
6      $C_{\text{reason}} \leftarrow \text{reduce}(C_{\text{reason}}, C_{\text{learn}}, l_r, \rho)$ 
7      $C_{\text{learn}} \leftarrow \text{resolve}(C_{\text{learn}}, C_{\text{reason}}, l_r)$ 
8    $\rho \leftarrow \rho \setminus \{l_r\}$ 
9 return  $C_{\text{learn}}$ 

```

---

The key invariant of Algorithm 1 is that in each iteration the resolvent  $C_{\text{learn}}$  remains falsified. In the clausal version this holds even without the reduction step in line 6. However, for general linear constraints this is not the case, as shown by the next example.

► **Example 1.** Consider the two PB constraints  $C_{\text{reason}} = x_1 + x_2 + 2x_3 \geq 2$  and  $C_{\text{confl}} = x_1 + 2\bar{x}_3 + x_4 + x_5 \geq 3$  and the partial assignment  $\rho = \{x_1 = 0, x_3 = 1\}$  where  $x_1 = 0$  is a decision, and  $x_3 = 1$  is propagated by  $C_{\text{reason}}$ . Under  $\rho$  the constraint  $C_{\text{confl}}$  is falsified. Applying generalized resolution to cancel  $x_3$  yields the constraint  $2x_1 + x_2 + x_4 + x_5 \geq 3$  which is not falsified under  $\rho$ .

In the following sections, we present three different reduction techniques for Algorithm 1 that operate directly on PB constraints. The main idea is to apply valid operations on the reason constraint to reduce the slack and ensure that the resolvent will have negative slack. The two main ingredients of the reduction techniques are *weakening* and *cutting planes* and are applied to the reason constraint until the invariant is fulfilled.

Weakening a literal in a PB constraint simply sets it to 1. For example, weakening a constraint  $C : x_1 + x_2 + 2x_3 \geq 2$  on  $x_1$  yields  $x_2 + 2x_3 \geq 1$ . Weakening is a valid operation since it simply adds a multiple of the valid bound constraint  $\bar{x}_1 \geq 0$  to  $C$ . Note that weakening entails a loss of information. However, as we will see, it is a necessary operation to reduce the slack of the reason constraint. Note that whenever weakening is applied on non-falsified literal, it does not change the slack of the constraint. See Section 3.7 for more details on weakening.



Our main focus in this paper, however, is the second necessary ingredient of the reduction algorithm: cutting planes (cuts). Cuts are applied to the “weakened” version of the reason constraint in order to reduce its slack. We first present two well-documented cuts from existing literature, namely *Saturation* (Section 3.3) and *Division* (Section 3.4). Both ensure the reduction of the slack of the reason constraint to 0 at least after weakening all non-falsified literals in the original reason constraint. In Section 3.5, we introduce a new cut based on the *Mixed Integer Rounding (MIR)* procedure and prove that it has the same property. In Section 3.6 we show that the reduction algorithm using MIR always returns an equally strong or stronger reason constraint than the reduction using Division.

### 3.3 Saturation-based Reduction

First, we present the *Saturation* cut. Then, we provide details about the *Saturation-based Reduction* algorithm and demonstrate how the reduction ensures that the key invariant of conflict analysis holds.

► **Definition 2** (Saturation Cut). *Let  $C : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$ . The **Saturation Cut** of  $C$  is given by the constraint*

$$\sum_{i \in \mathcal{N}} \min\{a_i, b\} \ell_i \geq b.$$

Saturation is a valid cut known as coefficient tightening cut in the MIP literature [10] and does not entail a loss of information. Algorithm 2 is used to reduce the reason constraint  $C_{\text{reason}}$  before applying generalized resolution. Similar to the implementation in [12], in each iteration, the algorithm picks a non-falsified literal in the reason constraint different from the literal we are resolving on and weakens it. Then it applies the Saturation cut to the resulting constraint. The algorithm terminates when the slack of the resolvent becomes negative.

■ **Algorithm 2** Saturation-based Reduction Algorithm.

---

<b>Input</b>	: conflict constraint $C_{\text{conf}}$ , reason constraint $C_{\text{reason}}$ , literal to resolve $\ell_r$ , partial assignment $\rho$
<b>Output</b>	: reduced reason $C_{\text{reason}}$
1	<b>while</b> $\text{slack}(\text{resolve}(C_{\text{reason}}, C_{\text{conf}}, \ell_r), \rho) \geq 0$ <b>do</b>
2	$\ell_j \leftarrow$ non falsified literal in $C_{\text{reason}} \setminus \{\ell_r\}$
3	$C_{\text{reason}} \leftarrow \text{weaken}(C_{\text{reason}}, \ell_j)$
4	$C_{\text{reason}} \leftarrow \text{saturate}(C_{\text{reason}})$
5	<b>return</b> $C_{\text{reason}}$

---

For completeness, we prove the following well-known fact that demonstrates that the slack of the reason constraint will be reduced to 0 at the latest after weakening the last non-falsified literal and applying the Saturation cut. Since the slack is subadditive, the resolvent’s slack becomes negative and the resolvent is thus falsified.

► **Lemma 3.** *Let  $\rho$  be a partial assignment, and  $C_{\text{reason}} : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$  a constraint propagating a literal  $\ell_r$  to 1. Further, assume that  $\text{slack}(C_{\text{reason}}, \rho) > 0$ . Then, after weakening all non-falsified literals in  $C_{\text{reason}}$  (except for  $\ell_r$ ) and applying Saturation on  $C_{\text{reason}}$ , the slack of the reduced reason constraint is 0.*

**Proof.** First, we rewrite the constraint  $C_{\text{reason}}$  as

$$\sum_{j:\rho(j)=0} a_j \ell_j + \sum_{i \neq r: \rho(i) \neq 0} a_i \ell_i + a_r \ell_r \geq b.$$

Since  $\text{slack}(C_{\text{reason}}, \rho) := \sum_{i \neq r: \rho(i) \neq 0} a_i + a_r - b > 0$ , it holds that

$$a_r > b - \sum_{i \neq r: \rho(i) \neq 0} a_i. \quad (2)$$

After weakening all literals from  $\{i \neq r : \rho(i) \neq 0\}$  the constraint  $C_{\text{reason}}$  becomes

$$\sum_{j:\rho(j)=0} a_j \ell_j + a_r \ell_r \geq \tilde{b} := b - \sum_{i \neq r: \rho(i) \neq 0} a_i. \quad (3)$$

Applying Saturation on (3) sets  $a_r$  to  $\tilde{b}$  because of (2). Therefore the slack of the reduced reason constraint becomes  $\tilde{b} - \tilde{b} = 0$ . ◀

### 3.4 Division-based Reduction

A very competitive alternative to Saturation in the reduction algorithm is based on Division cuts. Division is also a valid cut known as Chvátal-Gomory cut in the MIP literature [13].

► **Definition 4** (Division Cut). *Let  $C : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$ . The **Division Cut** of  $C$  with divisor  $d \in \mathbb{Z}_{>0}$  is given by the constraint*

$$\sum_{i \in \mathcal{N}} \left\lceil \frac{a_i}{d} \right\rceil \ell_i \geq \left\lceil \frac{b}{d} \right\rceil.$$

To see why this procedure is valid, we can think of it as three steps: dividing by  $d$  maintains the validity of the constraint; rounding up coefficients on the left-hand side relaxes the constraint and is hence valid; the validity of rounding up the right-hand side follows from the integrality of the left-hand side coefficients and literals.

In the Division-based reduction algorithm, the divisor  $d$  used is the coefficient of the literal  $\ell_r$  we are resolving on. As proven in [23], it suffices to weaken non-falsified variables with a coefficient that is not a multiple of  $a_r$ , i.e., from the index set  $W := \{i \in \mathcal{N} : \rho(i) \neq 0 \text{ and } a_i \nmid a_r\}$ . After weakening all literals in  $W$  and applying Division on  $C_{\text{reason}}$ , the slack of the reduced reason constraint is 0, which for completeness we include in Lemma 6 below.

### 3.5 MIR-based Reduction

Next, we define a new cut for the reduction algorithm based on the *Mixed Integer Rounding* formula [37], which is a generalization of Gomory's mixed integer cuts [27].

► **Definition 5** (Mixed Integer Rounding Cut). *Let  $C : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$ . The **Mixed Integer Rounding (MIR) Cut** of  $C$  with divisor  $d \in \mathbb{Z}_{>0}$  is given by the constraint*

$$\sum_{i \in I_1} \left\lceil \frac{a_i}{d} \right\rceil \ell_i + \sum_{i \in I_2} \left( \left\lfloor \frac{a_i}{d} \right\rfloor + \frac{f(a_i/d)}{f(b/d)} \right) \ell_i \geq \left\lceil \frac{b}{d} \right\rceil, \quad (4)$$

where

$$I_1 = \{i \in \mathcal{N} : f(a_i/d) \geq f(b/d) \text{ or } f(a_i/d) \in \mathbb{Z}\},$$

$$I_2 = \{i' \in \mathcal{N} : f(a_{i'}/d) < f(b/d) \text{ and } f(a_{i'}/d) \notin \mathbb{Z}\},$$

and  $f(\cdot) = \cdot - \lfloor \cdot \rfloor$ . To obtain a normalized version of the MIR cut, we multiply both sides of the constraint by  $(b \bmod d)$ .

The proof that applying MIR to a constraint is a valid procedure can be found in [37]. Similar to the Division-based reduction, it suffices to weaken non-falsified variables with a coefficient that is not a multiple of  $a_r$  before applying MIR in order to reduce the slack of the reason constraint to at most 0. This is shown in the following lemma.

► **Lemma 6.** *Let  $\rho$  be a partial assignment and  $C_{\text{reason}} : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$  a constraint propagating a literal  $\ell_r$  to 1. Then, after weakening all non-falsified literal in  $W := \{i \in \mathcal{N} : \rho(i) \neq 0 \text{ and } a_r \nmid a_i\}$  and applying Division or MIR on  $C_{\text{reason}}$  with  $d = a_r$ , the slack of the reduced reason is at most 0.*

**Proof.** After weakening all literals in  $W$ , the constraint  $C_{\text{reason}}$  becomes

$$a_r \ell_r + \sum_{j \in \mathcal{N} \setminus W} a_j \ell_j \geq \tilde{b} := b - \sum_{i \in W} a_i. \quad (5)$$

Its slack is

$$\text{slack}(C_{\text{reason}}, \rho) = a_r + \sum_{\substack{j \in \mathcal{N} \setminus W: \\ \rho(j) \neq 0}} a_j - \tilde{b} = a_r + \sum_{\substack{j \in \mathcal{N}: \\ \rho(j) \neq 0, a_r \nmid a_j}} a_j - \tilde{b}.$$

Since weakening does not affect the slack, we have  $\text{slack}(C_{\text{reason}}, \rho) < a_r$ .

1. After applying the Division cut to (5) with  $d = a_r$ , the slack becomes

$$\text{slack}(C_{\text{reason}}, \rho) = 1 + \sum_{\substack{j \in \mathcal{N}: \\ \rho(j) \neq 0, a_r \nmid a_j}} \left\lceil \frac{a_j}{a_r} \right\rceil - \left\lceil \frac{\tilde{b}}{a_r} \right\rceil \leq 1 + \sum_{\substack{j \in \mathcal{N}: \\ \rho(j) \neq 0, a_r \nmid a_j}} \frac{a_j}{a_r} - \frac{\tilde{b}}{a_r} < \frac{a_r}{a_r} = 1. \quad (6)$$

Because  $C_{\text{reason}}$  contains only integer coefficients after applying the division rule, its slack is integer; hence, it must be at most 0.

2. Applying the MIR cut to (5) with  $d = a_r$  results in the same slack as in (6). This is because all left-hand side coefficients in the slack computation are divisible by  $d$ , hence they fall into the index set  $I_1$  and are transformed the same way as by the Division cut. ◀

### 3.6 Dominance Relationships

In this section, we would like to discuss briefly known dominance relationships between the different reduction techniques. The ultimate goal is to find a reduction technique that yields the strongest possible reason constraint to use in the resolution step of conflict analysis. The following lemma states the well-known fact that constraints from Saturation-based reduction are always at least as strong as the resolvents created during clausal conflict analysis as described in Section 3.1.

► **Lemma 7.** *Let  $\rho$  be a partial assignment and  $C_{\text{reason}} : \sum_{i \in \mathcal{N}} a_i \ell_i \geq b$  be a PB constraint which propagates literal  $\ell_r$  to 1. Let  $C'_{\text{reason}}$  and  $C''_{\text{reason}}$  be the constraints obtained by clausal and Saturation-based reduction, respectively. Then  $C''_{\text{reason}}$  implies  $C'_{\text{reason}}$ .*

**Proof.** Under the current partial assignment  $\rho$ , the disjunctive clause reason is given by  $\ell_r \bigvee_{j: \rho(\ell_j)=0} \ell_j$ , which can be linearized as

$$C'_{\text{reason}} : \ell_r + \sum_{j: \rho(\ell_j)=0} \ell_j \geq 1.$$

Now let  $W$  be the set of all non-falsified literals, except  $\ell_r$ . After weakening all literals in  $W$  and applying Saturation, we obtain the constraint

$$C''_{\text{reason}} : \min\{a_r, b - \sum_{i \in W} a_i\} \ell_r + \sum_{j: \rho(\ell_j)=0} \min\{a_j, b - \sum_{i \in W} a_i\} \ell_j \geq b - \sum_{i \in W} a_i.$$

As in the proof of Lemma 3, it holds that  $\min\{a_r, b - \sum_{i \in W} a_i\} = b - \sum_{i \in W} a_i$ . Now, after scaling  $C''_{\text{reason}}$  by  $b - \sum_{i \in W} a_i$  we see that  $C''_{\text{reason}}$  has the same right-hand side as  $C'_{\text{reason}}$ , but smaller or equal coefficients on the left-hand side. ◀

In [26] the authors show that using Division instead of Saturation can be exponentially stronger, and that a single Saturation step can be simulated by an exponential number of Division steps.

The dominance of MIR cuts over Chvátal-Gomory cuts is a well-known fact in the MIP literature. The following lemma shows essentially the same result as in [15], but in the context of conflict analysis for pseudo-Boolean problems.

► **Lemma 8.** *Let  $\rho$ ,  $C_{\text{reason}}$ ,  $\ell_r$  be given as in Lemma 7. Let  $C'_{\text{reason}}$  and  $C''_{\text{reason}}$  be the constraints obtained by Division-based and MIR-based reduction, respectively. Then  $C''_{\text{reason}}$  implies  $C'_{\text{reason}}$ .*

**Proof.** Let  $C'_{\text{reason}}, C''_{\text{reason}}$  be constraints as in Definition 4 and 5, respectively, with divisor  $d = a_r$ . The constraints have the same right-hand side and the same coefficients for all literals  $\ell_i$  with  $i \in I_1$ . For  $i \in I_2$  the coefficient of literal  $\ell_i$  in  $C'_{\text{reason}}$  is given by  $\left\lfloor \frac{a_i}{a_r} \right\rfloor$  and in  $C''_{\text{reason}}$  by  $\left\lfloor \frac{a_i}{a_r} \right\rfloor + \frac{f(a_i/a_r)}{f(b/a_r)}$ . The coefficients of the literals  $\ell_i$  in  $C''_{\text{reason}}$  are always greater than or equal to the coefficients in  $C'_{\text{reason}}$ , since by definition of the set  $I_2$  it holds that  $f(a_i/a_r)/f(b/a_r) < 1$ . Therefore  $C''_{\text{reason}}$  implies  $C'_{\text{reason}}$ . ◀

As an example, consider the partial assignment  $\rho = \{x_1 = 0, x_2 = 0, x_3 = 1\}$  and the constraint  $C_{\text{reason}} : 2x_1 + 6x_2 + 10x_3 \geq 8$  which propagates variable  $x_3$  to 1. Then the Division cut with divisor 10 is  $x_1 + x_2 + x_3 \geq 1$ . The MIR cut with the same divisor is  $\frac{0.2}{0.8}x_1 + \frac{0.6}{0.8}x_2 + x_3 \geq 1$ . Multiplying with  $8 \bmod 10 = 8$  gives the normalized MIR cut  $2x_1 + 6x_2 + 8x_3 \geq 8$ . The normalized MIR cut is stronger than the Division cut, which can be easily seen after scaling the Division cut by 8.

### 3.7 Practical Aspects of Weakening

While the evaluation of different weakening strategies is not the focus of this paper, we would like to discuss briefly some practical aspects of weakening literals. In our implementation we consider the following iterative weakening strategy: weaken free literals first followed by implied literals. We stop as soon as the resolvent is falsified under the remaining partial assignment. Intuitively, this order is motivated by the fact that free literals are not relevant for the propagation of literals in the reason constraint and do not affect the falsification of the conflict constraint.

However, the optimal order in which to weaken literals is not yet fully understood, and remains an open research question. Possible approaches include weakening literals in order of increasing or decreasing coefficient size. In [33] the authors conducted experiments with various weakening techniques, including partial weakening of literals and applying weakening on the conflict constraint, but the results did not yield a conclusive “best” weakening strategy.

A simple alternative is to weaken literals in a single sweep. For all three reduction algorithms, we can weaken the entire candidate set of literals as stated in Lemma 3 and Lemma 6 at once. Weakening literals all at once leads to a faster reduction algorithm

since repeated slack computations are avoided and only one cut is applied in each iteration. However, this may result in the constraint being less informative due to unnecessary weakening of literals.

## 4 Experiments

It is well known in the SAT and PB communities that efficient conflict-driven search requires substantial amounts of very careful engineering. In this first work, our focus has been on importing and adapting the pseudo-Boolean conflict analysis to a MIP setting – which is a nontrivial task in its own right – leaving further optimizations as future work.

All techniques from Section 3 have been implemented in the open source MIP solver SCIP 8.0.3 [7] and we conducted extensive experiments to compare the different reduction techniques in isolation. Obtaining accurate performance results for MIP solvers requires a carefully designed experimental setup since even small changes to algorithms or the input data can have a large impact on the behavior and the performance of the solver. This is a well-known fact in the MIP literature known as *performance variability* [36]. To lessen the effects of performance variability and obtain a fair comparison of the different reduction techniques in the context of MIP solving, we use a fairly large and diverse testset of instances and different permutations of each instance, see, e.g., [25]. Our experiments were carried out on all pure 0–1 models from the MIPLIB 2017 collection [25]. After removing numerically unstable models (with the tag “numerics”) our testset consists of 195 instances permuted by 5 different random seeds, giving a total of 975 measurements per run. For the remainder of this paper, we will refer to the combination of a model and a permutation as an *instance*. All experiments are conducted on a cluster with Intel Xeon Gold 6338 CPUs with a limit of 16GB of RAM.

It’s worth noting that SCIP, along with its underlying LP solver, is based on floating-point arithmetic. Implementing a Pseudo-Boolean Optimization solver using a limited-precision LP-based branch-and-cut framework comes with some technical challenges which are discussed, e.g., in [5, 6]. From a theoretical standpoint, switching between reals and integers (rather than between limited and arbitrary precision) is straightforward:

All the algorithms presented in Section 3 can be naturally extended to the case of 0–1 constraints with coefficients that are real numbers instead of nonnegative integers. The Chvátal-Gomory procedure, MIR cutting, and coefficient tightening algorithm were originally designed for MIP with real coefficients.

However, in practice, floating-point arithmetic may cause numerical issues due to imprecise representations of real numbers and cancellation effects. To mitigate the risk of numeric instability, many components of SCIP, such as MIR-cut generation, utilize double-double precision arithmetic [18], which could be also employed in conflict analysis. Currently, for constraints generated in conflict analysis, we use the following standard techniques:

- We terminate conflict analysis if the coefficients of the constraints span too many orders of magnitude. Specifically, if the quotient of the largest to smallest coefficient is large (in our implementation,  $10^6$ ), we stop conflict analysis.
- We remove variables from the conflict constraint if their coefficients are too small (in our implementation,  $10^{-9}$ ), thereby relaxing the constraint slightly.

The latter threshold is a common default value for the zero tolerance in MIP solvers, and the former is a common modeling recommendation for MIP.

■ **Table 1** Average percentage of true or unassigned literals that should be weakened to preserve the conflict analysis invariant. This experiment is conducted on the test set with 3 random seeds.

Setting	avg(%) literals weakened
Division	98.0
Saturation	99.7
MIR	97.3

#### 4.1 Pre-Experiment: Weaken-All-At-Once vs. Weaken-Iteratively

As noted earlier, the weakening rule can be applied iteratively or in a single sweep. In preliminary experiments, we noticed that in almost all cases, most unassigned or true literals must be weakened to achieve the conflict analysis invariant that the resolved constraint has a negative slack. Table 1 summarizes this finding for different reduction techniques: Over all instances and all conflict analysis calls, an average between 97.3% (MIR) and 99.7% (Saturation) of all literals had to be weakened. Furthermore, for most instances both weakening variants did not lead to different execution paths.

In this case, weakening all literals at once avoids the overhead of iterative use of cuts and expensive slack computations. Consequently, we decided to always weaken all literals at once and apply the cut rule on the reason side only once for the remaining experiments presented in this paper.

#### 4.2 Main Experiments: Comparing Different Reasoning Techniques

In the following, we compare all different reduction techniques from Section 3 to SCIP without any conflict analysis.

In our comparisons, we report for each technique the number of optimally solved instances, as well as the shifted geometric means of the number of processed nodes and the CPU time in seconds. The shifted geometric mean, a standard performance aggregator in the MIP literature, of the values  $t_1, \dots, t_n$  is defined as

$$\left( \prod_{i=1}^n (t_i + s) \right)^{1/n} - s, \quad (7)$$

for some  $s > 0$ . We set the shift  $s$  to 1 second for the CPU time and to 100 nodes for the number of nodes. Our base of comparison is SCIP without conflict analysis (“No Conflicts”). We report absolute values for the shifted means, and also quotients comparing them to our base setting. A factor below 1 means that a setting was faster (or needed less nodes), and a factor greater than 1 means that it was detrimental.

In Table 2 we report the results of our experiments. The table is split in four parts. We show results for “all” instances, as well as for three subsets of instances: (i) instances that are “affected” by conflict analysis, hence where the execution path of at least one setting differs from the others, (ii) “[100, limit]” instances, which take at least 100 seconds to solve to optimality or hit the time limit and (iii) “all-optimal”, which are instances solved by all settings. Note that the number of nodes can only be fairly compared on the “all-optimal” subset, since the number of nodes when hitting a time limit is hard to interpret and hard to aggregate with the same statistics on instances that are solved to optimality.

The variant of SCIP with clausal conflict analysis is referred to as “Clausal-CA”. For a fair comparison of the different strategies, we disabled the upgrading of constraints to specialized types, i.e., all generated conflicts are treated as linear constraints, and further

■ **Table 2** Main results.

	Setting	solved	time(s)	# nodes	time quot	nodes quot
all(975)	No Conflicts	394	656.75	784	1.0	1.0
	Clausal-CA	405	603.55	682	0.92	0.87
	Division	419	601.4	683	0.92	0.87
	MIR	420	599.37	677	0.91	0.86
	Saturation	418	599.76	692	0.91	0.88
affected(295)	No Conflicts	259	160.46	1096	1.0	1.0
	Clausal-CA	270	122.64	776	0.76	0.71
	Division	284	119.24	707	0.74	0.65
	MIR	285	118.29	700	0.74	0.64
	Saturation	283	118.09	735	0.74	0.67
[100, limit](218)	No Conflicts	182	667.14	2056	1.0	1.0
	Clausal-CA	193	486.45	1466	0.73	0.71
	Division	207	486.23	1345	0.73	0.65
	MIR	208	485.26	1336	0.73	0.65
	Saturation	206	491.98	1428	0.74	0.69
all-optimal(374)	No Conflicts	374	46.16	320	1.0	1.0
	Clausal-CA	374	40.58	259	0.88	0.81
	Division	374	40.75	244	0.88	0.77
	MIR	374	40.40	241	0.88	0.75
	Saturation	374	40.24	246	0.87	0.77

only generated one conflict per call. Conversely, we accept PB reasoning conflicts only if the number of nonzeros is less than 15% of the original problem variables, as in the default clausal implementation in SCIP. Our preliminary experiments confirmed that in our implementation, it is indeed detrimental to accept too-long conflicts. We did, however, add a fallback strategy, of applying weakening on the conflict constraint if the constraints are too long. This happens for about 9% of the instances.

We observe that all conflict analysis variants solved more instances than SCIP without conflict analysis, needed significantly less nodes on the all-optimal set, and less time on all four instance sets. Note that on average, the time spent in conflict analysis is only about 0.1% of the total run time. The three PB conflict analysis variants could solve more instances than the clausal variant, and needed significantly less nodes. The difference in time was less pronounced.

The performance of the PB conflict analysis variants is quite similar in all three cases. Nevertheless, MIR-based reduction could solve the most instances and needed the least nodes on the all-optimal set. When looking at the seemingly identical time-wise performance in more detail, it turns out that MIR also slightly improves on the other settings in this measure. There are 104 instances for which the path differs between Saturation-based resolution and MIR-based resolution and MIR was on average 1.1% faster on those. There are 86 instances for which the path differs between Division-based resolution and MIR-based resolution and MIR was on average 3.6% faster on those. Consequently, we decided to concentrate on MIR-based resolution for our next statistic.

Ultimately, the purpose of conflict constraints is to restrict the future search space by propagating literal assignments and pruning the search tree. Hence we analyzed how many conflicts each of the methods generates in shifted geometric mean, how large these conflicts are on average, and how many of them lead to propagations down the road. Table 3 shows

■ **Table 3** Shifted geometric mean of number of conflicts, average percentage of conflict constraints that propagate at least once and average length of learned conflicts.

Setting	mean # conflicts	avg % prop. conflicts	avg # literals
Clausal-CA	290.77	34.54	82.45
MIR	169.61	58.54	80.20

the results on the set of all instances that have a search tree of at least 100 nodes (to get a decent chance of conflict generation and propagation) and for which at least one conflict was generated with one of the methods. We consider only instances where the two settings have the same execution path. We observe that our MIR-based conflict analysis generated about a third less conflicts, but at the same time, they are much more likely to propagate: for the classic clausal conflict analysis of SCIP, about a third of the generated conflicts are used for propagation later on, while for our MIR-based variant, slightly more than half (58.54%) of all conflicts propagate at least once. At the same time, MIR-based conflicts are about the same size as clausal conflicts.

At first glance, this might appear as a contradiction, given that, as a rule of thumb, shorter conflicts tend to propagate more often and one might expect similar-sized conflicts to be similarly likely to propagate. Note, however, that the conflicts are of a quite different nature in the two cases. On the one hand, clausal conflicts are always logic clauses that only propagate when all but one literal are assigned. On the other hand, MIR-based conflicts are general pseudo-Boolean constraints, which might propagate some assignments (of literals with large coefficients) even when a majority of literals are still unassigned. This goes nicely together with the above observation that the reduction in the number of nodes is more pronounced than the reduction in runtime. As a final remark, integrating PB conflict analysis in a production-grade MIP solver would require substantially more work, but should also be expected to provide substantial further improvements measured in wallclock time.

## 5 Conclusion

In this work, we study how to integrate pseudo-Boolean conflict analysis for 0–1 integer linear programs into a MIP solving framework. In contrast to standard MIP conflict analysis, the pseudo-Boolean method operates directly on the linear constraints, rather than on clauses extracted from these constraints, and this makes it exponentially stronger in terms of reasoning power. Viewing PB conflict analysis from a MIP perspective is also helpful since it provides a view of the algorithm as a sequence of linear combinations and cuts, and we use this to strengthen the pseudo-Boolean conflict analysis further by developing a new conflict analysis method using the powerful mixed integer rounding (MIR) cuts.

We have made a first proof-of-concept implementation of our new pseudo-Boolean conflict analysis method, as well as methods from the PB literature based on saturation [34] and division [23], in the open-source MIP solver SCIP, and have run experiments on 0–1 ILP instances from MIPLIB 2017 comparing the different methods with each other and with standard clause-based MIP conflict analysis. We find that solving 0–1 ILPs with MIR-based pseudo-Boolean conflict analysis performs better than other methods, not only in the sense that it reduces the size of the search tree, but also in that our implementation can beat the highly optimized MIP conflict analysis currently used in SCIP in terms of actual running time. In our opinion, this demonstrates convincingly that pseudo-Boolean conflict analysis in MIP is a research direction that should be worth pursuing further, and that similar proof-of-concept studies could also be relevant to investigate for other combinatorial solving paradigms such as constraint programming.



As already noted above, an obvious direction of future work is to provide a more carefully engineered version of pseudo-Boolean conflict analysis that could deliver more fully on the potential for improved performance identified by our experiments. In addition to optimizing the existing code, however, it would be valuable to develop a better understanding of how and why the conflict analysis works and of ways in which the reasoning could be improved.

Pseudo-Boolean conflict analysis alternates between weakening constraints (to eliminate seemingly less relevant variables) and strengthening them by applying cut rules (to get tighter propagation on the variables that remain). The interplay between these two operations is quite poorly understood even for pseudo-Boolean solvers, and so both PB solvers and MIP solvers could gain from a careful study of how to strike the right balance. Since PB conflict analysis can be performed with several different reduction methods, and since different reduction methods can be employed independently in consecutive steps in one and the same conflict analysis, it would also be good to be able to assess the quality of constraints derived during conflict analysis, so as to select the most promising candidate at each step to pass on to the next step in the conflict analysis.

Arguably the most interesting research question, though, is whether pseudo-Boolean conflict analysis could be extended beyond 0–1 ILPs to 0–1 mixed linear problems, and/or to general integer linear programs. It is worth noting that the latter has been attempted in [32, 41], but so far with quite limited success. It is clear that the algorithms presented in this paper *cannot* work for 0–1 mixed LPs or general ILPs if generalized in the obvious, naive way, and so additional, new ideas will be needed.

---

## References

- 1 Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, March 2007.
- 2 Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007. Available at [https://opus4.kobv.de/opus4-zib/files/1112/Achterberg\\_Constraint\\_Integer\\_Programming.pdf](https://opus4.kobv.de/opus4-zib/files/1112/Achterberg_Constraint_Integer_Programming.pdf).
- 3 Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.
- 4 Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, December 2004. Preliminary version in *IJCAI '03*.
- 5 Timo Berthold, Stefan Heinz, and Marc Pfetsch. Solving Pseudo-Boolean problems with SCIP, 2008.
- 6 Timo Berthold, Stefan Heinz, and Marc E Pfetsch. Nonlinear Pseudo-Boolean optimization: relaxation or propagation? In *Theory and Applications of Satisfiability Testing-SAT 2009: 12th International Conference, SAT 2009, Swansea, UK, June 30-July 3, 2009. Proceedings 12*, pages 441–446. Springer, 2009.
- 7 Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, et al. The SCIP Optimization Suite 8.0. *arXiv preprint arXiv:2112.08872*, 2021.
- 8 Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, February 2021.
- 9 Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.
- 10 AL Brearley, Gautam Mitra, and H Paul Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical programming*, 8:54–83, 1975.

- 11 Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 7, pages 233–350. IOS Press, 2nd edition, February 2021.
- 12 Donald Chai and Andreas Kuehlmann. A fast pseudo-Boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):305–317, March 2005. Preliminary version in *DAC '03*.
- 13 Vasek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete mathematics*, 4(4):305–337, 1973.
- 14 William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.
- 15 Gérard Cornuéjols and Yanjun Li. Elementary closures for integer programs. *Operations Research Letters*, 28(1):1–8, 2001. doi:10.1016/S0167-6377(00)00067-5.
- 16 Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.
- 17 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.
- 18 Theodorus Jozef Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971.
- 19 Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 26(1–4):26–55, October 2021. Preliminary version in *CPAIOR '20*.
- 20 Jo Devriendt, Stephan Gocht, Emir Demirović, Jakob Nordström, and Peter Stuckey. Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3750–3758, February 2021.
- 21 Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, March 2006.
- 22 Jan Elffers, Jesús Giráldez-Cru, Jakob Nordström, and Marc Vinyals. Using combinatorial benchmarks to probe the reasoning power of pseudo-Boolean solvers. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 75–93. Springer, July 2018.
- 23 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1291–1299, July 2018.
- 24 Matthew L Ginsberg. Dynamic backtracking. *Journal of artificial intelligence research*, 1:25–46, 1993.
- 25 Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 13:443–490, 2021. doi:10.1007/s12532-020-00194-3.
- 26 Stephan Gocht, Jakob Nordström, and Amir Yehudayoff. On division versus saturation in pseudo-Boolean solving. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI '19)*, pages 1711–1718, August 2019.
- 27 Ralph E. Gomory. An algorithm for the mixed integer problem. Technical Report P-1885, The RAND Corporation, June 1960.
- 28 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.
- 29 John N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12(1):217–239, December 1988.

- 30 John N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6(1):271–286, March 1992.
- 31 Yuejun Jiang, Thomas Richards, and Barry Richards. Nogood backmarking with min-conflict repair in constraint satisfaction and optimization. In *Principles and Practice of Constraint Programming: Second International Workshop, PPCP'94 Rosario, Orcas Island, WA, USA, May 2–4, 1994 Proceedings*, pages 21–39. Springer, 1994.
- 32 Dejan Jovanovic and Leonardo de Moura. Cutting to the chase solving linear integer arithmetic. *Journal of Automated Reasoning*, 51(1):79–108, June 2013. Preliminary version in *CADE-23*.
- 33 Daniel Le Berre, Pierre Marquis, and Romain Wallon. On weakening strategies for PB solvers. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT '20)*, volume 12178 of *Lecture Notes in Computer Science*, pages 322–331. Springer, July 2020.
- 34 Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, July 2010.
- 35 Vincent Liew, Paul Beame, Jo Devriendt, Jan Elffers, and Jakob Nordström. Verifying properties of bit-vector multiplication using cutting planes reasoning. In *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design (FMCAD '20)*, pages 194–204, September 2020.
- 36 Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. In *Theory driven by influential applications*, pages 1–12. INFORMS, 2013.
- 37 Hugues Marchand and Laurence A Wolsey. Aggregation and mixed integer rounding to solve mips. *Operations research*, 49(3):363–371, 2001.
- 38 João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.
- 39 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, July 2014.
- 40 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.
- 41 Robert Nieuwenhuis. The IntSat method for integer linear programming. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP '14)*, volume 8656 of *Lecture Notes in Computer Science*, pages 574–589. Springer, September 2014.
- 42 John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.
- 43 Masahiko Sakai and Hidetomo Nabeshima. Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. *IEICE Transactions on Information and Systems*, 98-D(6):1121–1127, June 2015.
- 44 Tuomas Sandholm and Robert Shields. Nogood learning for mixed integer programming. In *Workshop on Hybrid Methods and Branching Rules in Combinatorial Optimization, Montréal*, volume 20, pages 21–22, 2006.
- 45 Buser Say, Jo Devriendt, Jakob Nordström, and Peter Stuckey. Theoretical and experimental results for planning with learned binarized neural network transition models. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 917–934. Springer, September 2020.
- 46 Hossein M. Sheini and Karem A. Sakallah. Pueblo: A hybrid pseudo-Boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):165–189, March 2006. Preliminary version in *DATE '05*.

- 47 Richard M Stallman and Gerald J Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial intelligence*, 9(2):135–196, 1977.
- 48 Jakob Witzig, Timo Berthold, and Stefan Heinz. Computational aspects of infeasibility analysis in mixed integer programming. Technical Report 19-54, ZIB, Takustr. 7, 14195 Berlin, 2019.



# Using Canonical Codes to Efficiently Solve the Benzenoid Generation Problem with Constraint Programming

Xiao Peng

Univ Lyon, INSA Lyon, Inria, CITI, EA3720, 69621 Villeurbanne, France

Christine Solnon

Univ Lyon, INSA Lyon, Inria, CITI, EA3720, 69621 Villeurbanne, France

---

## Abstract

The Benzenoid Generation Problem (BGP) aims at generating all benzenoid molecules that satisfy some given properties. This problem has important applications in chemistry, and Carissan et al (2021) have shown us that Constraint Programming (CP) is well suited for modelling this problem because properties defined by chemists are easy to express by means of constraints. Benzenoids are described by hexagon graphs and a key point for an efficient enumeration of these graphs is to be invariant to rotations and symmetries. In this paper, we introduce canonical codes that uniquely characterise hexagon graphs while being invariant to rotations and symmetries. We show that these codes may be defined by means of constraints. We also introduce a global constraint for ensuring that codes are canonical, and a global constraint for ensuring that a pattern is included in a code. We experimentally compare our new CP model with the CP-based approach of Carissan et al (2021), and we show that it has better scale-up properties.

**2012 ACM Subject Classification** Mathematics of computing → Graph enumeration; Computing methodologies → Artificial intelligence

**Keywords and phrases** Benzenoid Generation Problem, Canonical Code, Hexagon Graph

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.28

**Acknowledgements** We want to thank authors of [3] who helped us reproduce their results.

## 1 Introduction

Benzenoids are hydrocarbon molecules whose carbon atoms are forming cycles of size 6, i.e., hexagons. An important chemistry problem concerns the generation of all benzenoids that satisfy some given properties [5]. This problem is called the Benzenoid Generation Problem (BGP) in [2]. As benzenoids are regular tilings with hexagonal faces, they may be represented by hexagon graphs such that a vertex is associated with every hexagonal face and an edge with every pair of vertices corresponding to adjacent faces [1, 2]. For example, we display in Figure 1 a benzenoid composed of five hexagonal faces and its associated hexagon graph. These hexagon graphs are always connected.

Different benzenoids may be represented with isomorphic hexagon graphs. For example, let us consider the hexagon graph displayed in Figure 1. The subgraph induced by  $c$ ,  $d$ , and  $e$  is isomorphic to the subgraph induced by  $a$ ,  $c$ , and  $d$  whereas their associated molecules are different because hexagons  $c$ ,  $d$ , and  $e$  are not aligned whereas hexagons  $a$ ,  $c$ , and  $d$  are aligned. To overcome this problem, we take into account edge directions, considering the six directions defined in Figure 1. In this case, the subgraph induced by  $c$ ,  $d$ , and  $e$  is no longer isomorphic to the subgraph induced by  $a$ ,  $c$ , and  $d$  because edges  $(c, d)$  and  $(d, e)$  have different directions (0 and 1), whereas edges  $(a, c)$  and  $(c, d)$  have the same direction (0).



© Xiao Peng and Christine Solnon;

licensed under Creative Commons License CC-BY 4.0

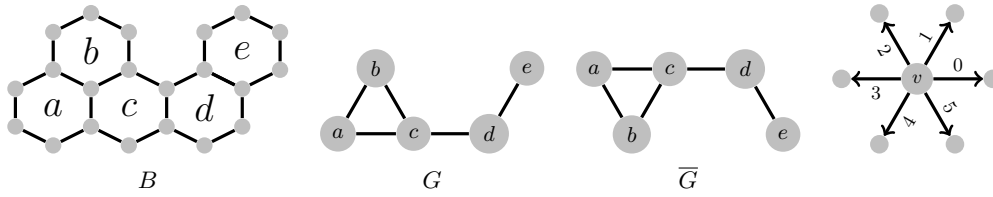
29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 28; pp. 28:1–28:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Example of benzenoid  $B$  and its associated hexagon graph  $G$ , and symmetrical graph  $\bar{G}$ . Right: Edge directions from a vertex  $v$  to each of its 6 possible neighbours.

To solve the BGP, we have to enumerate hexagon graphs that satisfy some given properties and these properties are usually easily expressed by means of constraints (such as, for example, the absence of cliques of order three, or the inclusion of some patterns). Two main approaches may be used:

- The dedicated approach of [1] uses canonical codes to represent hexagon graphs. These codes are invariant to rotations and symmetries and they are used to decide whether two graphs are isomorphic or not in linear time. However, these codes cannot represent benzenoids with holes (called coronoids). Furthermore, this approach is not declarative and it does not allow one to easily add constraints on the graphs to be enumerated.
- The Constraint Programming (CP)-based approach of [2, 3] basically searches for all connected subgraphs within an initial hexagon graph, and it is implemented in Choco [11] using graph variables. Using CP allows one to easily add constraints and this approach is able to generate all kinds of benzenoids, including coronoids. However, it is less efficient than the dedicated approach.

In this paper, we introduce an approach which is both efficient and declarative: like [1], it is based on canonical codes but these canonical codes can represent all benzenoids (including coronoids); like [2, 3], it is based on CP so that one may easily add constraints to specify structural properties. In Section 2, we introduce our new canonical code and study some of its properties that are used to efficiently generate canonical codes with CP. In Section 3, we extend canonical codes to the case where benzenoids are constrained to contain some given patterns. In Section 4, we introduce CP models for solving BGPs. In Section 5, we report experimental results and compare our approach with the CP-based approach of [2, 3].

## Notations

Given two integer values  $i$  and  $j$ , we note  $[i, j]$  the set of all integer values ranging from  $i$  to  $j$ . Given a hexagon graph  $G$ , we note  $\bar{G}$  the symmetrical graph obtained by mirroring  $G$  with respect to the  $x$ -axis, as displayed in Figure 1. Given a hexagon graph  $G = (V, E)$  and a subset of vertices  $S \subseteq V$ , we note  $G_{\downarrow S}$  the subgraph of  $G$  induced by  $S$ , i.e.,  $G_{\downarrow S} = (S, E \cap S \times S)$ .

## 2 Representation of Hexagon Graphs with Canonical Codes

A key point for an efficient enumeration of hexagon graphs is to be invariant to rotations and symmetries. For example, the graph displayed in Figure 1 is isomorphic to any other graph obtained by rotating it of  $k * 60^\circ$  with  $k \in \mathbb{N}$  and/or mirroring it with respect to the  $x$ -axis. In [3], constraints are added to break these symmetries and avoid generating several times the same graph. In this paper, we propose to use another approach based on canonical codes, i.e., integer sequences that uniquely characterise graphs and that are invariant to rotations and symmetries. We introduce our canonical code in Section 2.1. We study its properties in Section 2.2. We compare it with related work in Section 2.3.

■ **Algorithm 1**  $\text{BFS}_G(v_0, v_1)$ .

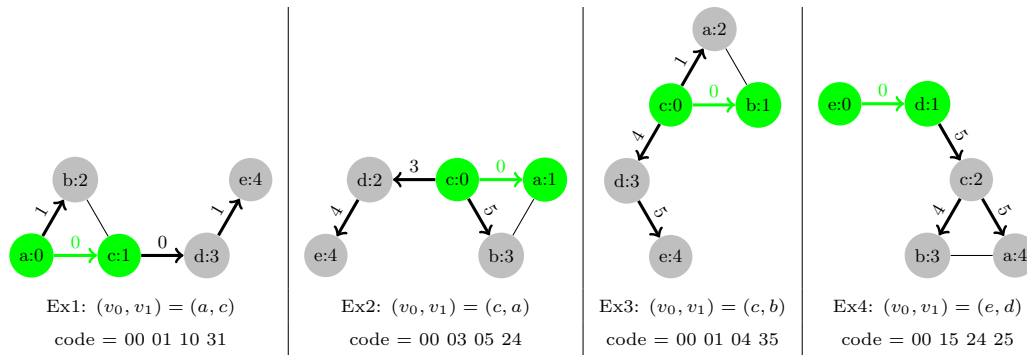
---

```

Input: A hexagon graph  $G$  and an initial edge  $(v_0, v_1)$  of  $G$ 
Output: The code associated with a BFS of  $G$  started from  $(v_0, v_1)$ 
1 rotate  $G$  so that the direction of edge  $(v_0, v_1)$  is equal to 0
2 for each vertex  $v_i$  of  $G$  do initialise  $\text{num}[v_i]$  to  $-1$ ;
3 set  $\text{num}[v_0]$  to 0 and initialise a counter  $c$  to 1
4 let  $q$  be an empty FIFO queue; add  $v_0$  in  $q$ 
5 initialise  $\text{code}$  to an empty sequence
6 while  $q$  is not empty do
7     remove from  $q$  its oldest vertex  $v_i$ 
8     for each edge  $(v_i, v_j)$  of  $G$  taken by order of increasing direction do
9         if  $\text{num}[v_j] < 0$  then
10             add  $v_j$  in  $q$ 
11             set  $\text{num}[v_j]$  to  $c$  and increment  $c$ 
12             add  $\text{num}[v_i]$  and the direction of edge  $(v_i, v_j)$  at the end of  $\text{code}$ 
13 return  $\text{code}$ 

```

---



■ **Figure 2** Examples of runs of  $\text{BFS}_G(v_0, v_1)$  when  $G$  is the graph of Figure 1 and  $(v_0, v_1)$  is equal to  $(a, c)$  for Ex1,  $(c, a)$  for Ex2,  $(c, b)$  for Ex3, and  $(e, d)$  for Ex4. For each vertex  $v_i \in \{a, b, c, d, e\}$ , we display  $v_i : \text{num}[v_i]$  in circles. Each edge  $(v_i, v_j)$  used lines 10-12 is displayed in bold and its direction is displayed on top of it (the initial edge  $(v_0, v_1)$  is displayed in green).

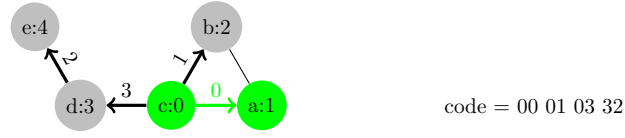
### 2.1 Definition of Canonical Codes

Given a hexagon graph  $G$  with  $n + 1$  vertices, a code is a sequence of  $n$  couples of integer values which is associated with a Breadth First Search (BFS) of  $G$  starting from a given initial edge  $(v_0, v_1)$ , as described in Algorithm 1. The initial edge is used to define the orientation of the graph (line 1): we always consider the orientation such that  $v_1$  is on the right of  $v_0$ . This allows us to be invariant to rotations.

► **Example 1.** We display in Figure 2 different runs of BFS on the hexagon graph of Figure 1, starting from different edges. In Ex1, we start from edge  $(a, c)$  which already has direction 0 so that we do not have to rotate the graph. When starting from edge  $(c, a)$  (resp.  $(c, b)$  and  $(e, d)$ ), we have to rotate the graph of  $180^\circ$  (resp.  $240^\circ$  and  $60^\circ$ ).

BFS assigns a different number  $\text{num}[v_i]$  to each vertex  $v_i$  corresponding to the order vertices are discovered:  $v_0$  is numbered with 0 (line 3) and when a vertex  $v_j$  is reached for the first time it is numbered with the next non-assigned value  $c$  (line 11). Each time the search discovers a new vertex  $v_j$  using an edge  $(v_i, v_j)$ ,  $\text{num}[v_i]$  and  $d(v_i, v_j)$  are added at the end of the code, where  $d(v_i, v_j)$  is the direction of edge  $(v_i, v_j)$  (line 12).





■ **Figure 3** Run of  $BFS_{\overline{G}}(c, a)$  where  $\overline{G}$  is the graph symmetrical to the graph  $G$  of Figure 1.

► **Example 2.** In Figure 2 (Ex1), we illustrate a run of  $BFS_G(a, c)$  when  $G$  is the graph of Figure 1. In this case, the code is 00 01 10 31: the first couple is 00 because vertex 1 ( $c$ ) has been reached from vertex 0 ( $a$ ) and  $d(a, c) = 0$ ; the second couple is 01 because vertex 2 ( $b$ ) has been reached from vertex 0 ( $a$ ) and  $d(a, b) = 1$ ; the third couple is 10 because vertex 3 ( $d$ ) has been reached from vertex 1 ( $c$ ) and  $d(c, d) = 1$ ; the fourth couple is 31 because vertex 4 ( $e$ ) has been reached from vertex 3 ( $d$ ) and  $d(d, e) = 1$ .

Note that the loop lines 8-12 considers edges outgoing from  $v_i$  by order of increasing directions. This ensures that we always compute the same code provided that the direction of the first edge  $(v_0, v_1)$  is fixed to 0.

Given a code  $p_1d_1 p_2d_2 \dots p_nd_n$  computed from a graph with  $n + 1$  vertices, we can draw this graph, starting from vertex number 0: for each  $i \in [1, n]$ ,  $p_i$  gives the number of the predecessor of vertex number  $i$  and  $d_i$  gives the direction of the edge  $(p_i, i)$ . Once all vertices have been drawn, missing edges can be added as every vertex is connected to all its neighbours. For example, in Ex1, we add an edge between vertices 1 and 2 because they have neighbour positions.

There exist different possible codes for a given graph. Each code corresponds to a different BFS starting from a different initial edge  $(v_0, v_1)$ . We define a total order on the set of all possible codes that may be associated with a given graph by considering a lexicographic order. Among all the possible codes for a graph, the smallest one according to this order is called the canonical code of this graph and it is unique.

So far, our canonical code is invariant to rotations but not to symmetries. To become invariant to symmetries, we must also compute all codes starting from all possible edges while considering the symmetrical graph  $\overline{G}$ . This leads us to the following definition.

► **Definition 3** (Canonical code  $cc(G)$ ). *The canonical code of a hexagon graph  $G = (V, E)$  is:  $cc(G) = \min\{BFS_G(v_0, v_1), BFS_{\overline{G}}(v_0, v_1) : (v_0, v_1) \in E\}$  when considering a lexicographic order to compare codes.*

► **Example 4.** The smallest code that may be computed for the graph  $G$  of Figure 1 is  $BFS_G(c, b)$ , displayed in Figure 2. However, if we consider the symmetrical graph  $\overline{G}$ ,  $BFS_{\overline{G}}(c, a)$  is smaller than  $BFS_G(c, b)$  (see Figure 3). This code is the smallest one for both  $G$  and  $\overline{G}$ , i.e.,  $cc(G) = 00 01 03 32$ .

## 2.2 Properties of Canonical Codes

The next two theorems are used to efficiently enumerate canonical codes with CP in Section 4.

► **Theorem 5.** *Given a code  $p_1d_1 p_2d_2 \dots p_nd_n$ , we have:*

**Property a:**  $\forall i \in [1, n - 1], p_i \leq p_{i+1}$ ;

**Property b:**  $\forall i \in [1, n - 1], (p_i = p_{i+1}) \Rightarrow (d_i < d_{i+1})$ .

**Proof.** Property a is a consequence of the fact that (i)  $q$  is a FIFO queue, (ii) vertices are added in  $q$  by order of increasing number, and (iii) when a vertex  $v_i$  is removed from  $q$  we add to *code* all couples  $p_id_i$  such that  $p_i = num[v_i]$  and the vertex at direction  $d_i$  from  $v_i$  is

not yet numbered. Hence, all couples added to *code* during one iteration of the loop lines 6-12 have the same value for  $p_i$ . Property b is a straightforward consequence of the fact that the loop lines 8-12 considers edges by order of increasing directions. ◀

► **Theorem 6.** *Let  $c = p_1d_1 p_2d_2 \dots p_nd_n$  be a canonical code. For each  $i \in [1, n - 1]$ , the prefix  $c_i = p_1d_1 p_2d_2 \dots p_id_i$  of  $c$  is also a canonical code.*

**Proof.** Let  $G$  and  $G_i$  be the graphs associated with  $c$  and  $c_i$ , where each vertex is numbered with  $num[v_i]$ . As  $c_i$  is a prefix of  $c$ ,  $G_i$  is the subgraph of  $G$  induced by the vertices numbered from 0 to  $i$ . Let us assume that  $c_i$  is not canonical, i.e., there exists another code  $c'_i = p'_1d'_1 p'_2d'_2 \dots p'_id'_i$  that is smaller than  $c_i$  while representing the same subgraph  $G_i$ . Let  $(v_0, v_1)$  be the edge of  $G_i$  such that  $c'_i = \text{BFS}_{G_i}(v_0, v_1)$  (resp.  $c'_i = \text{BFS}_{\overline{G_i}}(v_0, v_1)$  if  $c'_i$  is computed in the symmetrical graph). Let  $c' = \text{BFS}_G(v_0, v_1)$  (resp.  $c' = \text{BFS}_{\overline{G}}(v_0, v_1)$ ) be the code obtained by performing a search of  $G$  (resp.  $\overline{G}$ ) that starts from the same initial edge as the one used to obtain  $c'_i$ . Let  $j$  be the smallest vertex number for which  $c'$  and  $c'_i$  have different values, i.e.,  $\forall k < j$ , vertex number  $k$  has the same edges in both  $G$  and  $G_i$ , whereas vertex number  $j$  has more edges in  $G$  than in  $G_i$ . More precisely, let  $S$  (resp.  $S_i$ ) be the set of edges outgoing from  $j$  in  $G$  (resp.  $G_i$ ). We have  $S_i \subset S$  as  $G_i$  is a subgraph of  $G$ . We have to distinguish two different cases.

**Case 1:** the largest edge direction in  $S_i$  is equal to the largest edge direction in  $S$ . In this case,  $c'$  is smaller than  $c'_i$  because the sequence of couples associated with the successors of  $j$  in  $c'$  is necessarily smaller than the sequence of couples associated with the successors of  $j$  in  $c'_i$  (e.g., if edge directions in  $S_i$  are  $\{1, 5\}$  whereas edge directions in  $S$  are  $\{1, 2, 5\}$ , then  $j1 j2 j5 < j1 j5$ ).

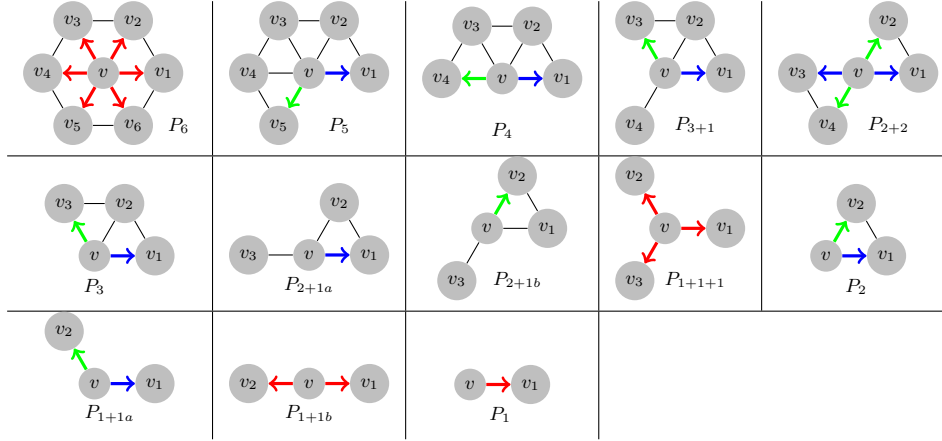
**Case 2:** the largest edge direction in  $S_i$  is smaller than the largest edge direction in  $S$ . In this case, either  $c'_i$  is a prefix of  $c'$  (if  $j$  is the greatest vertex with an outgoing edge), or  $c'$  is smaller than  $c'_i$  (e.g., if edge directions in  $S_i$  are  $\{1, 2\}$  whereas edge directions in  $S$  are  $\{1, 2, 5\}$ ,  $j1 j2 j5 < j1 j2 k$  because  $k > j$ , as stated in Property a).

In both cases, this implies that  $c' < c$  which contradicts the fact that  $c$  is canonical. ◀

**Computational complexity.** The canonical code of a graph with  $n$  vertices is computed in  $\mathcal{O}(n^2)$ : BFS is run twice for each edge and the number of edges is in  $\mathcal{O}(n)$  as a vertex cannot have more than six neighbours; the time complexity of BFS is  $\mathcal{O}(n)$  as the loop lines 6-12 is iterated  $n$  times and the loop lines 8-12 is iterated at most six times.

We may speed-up the computation by avoiding some calls to BFS. To this aim, we display in Figure 4 the 13 different possible patterns for the neighbourhood of a vertex  $v$  (up to rotations). For each pattern  $P_i$  such that  $v$  has  $k_i$  successors, every code computed by BFS with  $v_0 = v$  starts with a prefix  $0d_1 0d_2 \dots 0d_{k_i}$ , and we give below the smallest possible prefix composed of  $2k_i$  integers, denoted  $\pi_i$ , the set  $S$  of edges  $(v, v_k)$  of  $G$  such that  $\text{BFS}_G(v, v_k)$  starts with  $\pi_i$ , and the set  $\overline{S}$  of edges  $(v, v_k)$  of  $G$  such that  $\text{BFS}_{\overline{G}}(v, v_k)$  starts with  $\pi_i$  (these edges are highlighted in Figure 4).

$$\begin{array}{lll} \pi_6 = 00\ 01\ 02\ 03\ 04\ 05 & S = \{(v, v_i) : i \in [1, 6]\} & \overline{S} = \{(v, v_i) : i \in [1, 6]\} \\ \pi_5 = 00\ 01\ 02\ 03\ 04 & S = \{(v, v_1)\} & \overline{S} = \{(v, v_5)\} \\ \pi_4 = 00\ 01\ 02\ 03 & S = \{(v, v_1)\} & \overline{S} = \{(v, v_4)\} \\ \pi_{3+1} = 00\ 01\ 02\ 04 & S = \{(v, v_1)\} & \overline{S} = \{(v, v_3)\} \\ \pi_{2+2} = 00\ 01\ 03\ 04 & S = \{(v, v_1), (v, v_3)\} & \overline{S} = \{(v, v_2), (v, v_4)\} \end{array}$$



■ **Figure 4** Different neighbourhood patterns for a vertex  $v$ . We highlight in blue (resp. green and red) every edge  $(v, v_k)$  such that  $\text{BFS}_G(v, v_k)$  (resp.  $\text{BFS}_{\overline{G}}(v, v_k)$  and both  $\text{BFS}_G(v, v_k)$  and  $\text{BFS}_{\overline{G}}(v, v_k)$ ) starts with the smallest possible prefix.

$\pi_3 = 00\ 01\ 02$	$S = \{(v, v_1)\}$	$\overline{S} = \{(v, v_3)\}$
$\pi_{2+1a} = 00\ 01\ 03$	$S = \{(v, v_1)\}$	$\overline{S} = \emptyset$
$\pi_{2+1b} = 00\ 01\ 03$	$S = \emptyset$	$\overline{S} = \{(v, v_2)\}$
$\pi_{1+1+1} = 00\ 03\ 05$	$S = \{(v, v_1), (v, v_2), (v, v_3)\}$	$\overline{S} = \{(v, v_1), (v, v_2), (v, v_3)\}$
$\pi_2 = 00\ 01$	$S = \{(v, v_1)\}$	$\overline{S} = \{(v, v_2)\}$
$\pi_{1+1a} = 00\ 02$	$S = \{(v, v_1)\}$	$\overline{S} = \{(v, v_2)\}$
$\pi_{1+1b} = 00\ 03$	$S = \{(v, v_1), (v, v_2)\}$	$\overline{S} = \{(v, v_1), (v, v_2)\}$
$\pi_1 = 00$	$S = \{(v, v_1)\}$	$\overline{S} = \{(v, v_1)\}$

► **Example 7.** Let us consider the graph  $G$  of Figure 1. The pattern of vertex  $a$  in  $G$  is  $P_2$  (with  $a = v$ ,  $c = v_1$ , and  $b = v_2$ ). Therefore, the smallest possible code computed with  $v_0 = a$  starts with the prefix  $\pi_2 = 00\ 01$ , and the two codes that start with this prefix are  $\text{BFS}_G(a, c)$  and  $\text{BFS}_{\overline{G}}(a, b)$ . Also, the pattern of vertex  $c$  in  $G$  is  $P_{2+1b}$  (with  $c = v$ ,  $a = v_2$ ,  $b = v_1$ , and  $d = v_3$ ). Therefore, the smallest possible code computed with  $v_0 = c$  starts with the prefix  $\pi_{2+1a} = 00\ 01\ 03$ , and the only code that starts with this prefix is  $\text{BFS}_{\overline{G}}(c, a)$ .

To further reduce the number of searches, we define a total order among patterns.

► **Definition 8** (Order between patterns). Let  $P_i$  and  $P_j$  be two patterns (as listed in Figure 4), and let  $\pi_i1$  and  $\pi_j1$  be the sequences obtained by adding “1” at the end of  $\pi_i$  and  $\pi_j$ , respectively. We define  $P_i < P_j$  (resp.  $P_i = P_j$ ) if  $\pi_i1$  is lexicographically smaller than (resp. equal to)  $\pi_j1$ . We obtain the following order:

$$P_6 < P_5 < P_4 < P_{3+1} < P_3 < P_{2+2} < P_{2+1a} = P_{2+1b} < P_2 < P_{1+1+1} < P_{1+1a} < P_{1+1b} < P_1$$

Given two vertices  $v_i$  and  $v_j$ , if the pattern of  $v_i$  is smaller than the pattern of  $v_j$ , then we know that every code computed from  $v_i$  is smaller than every code computed from  $v_j$ . Hence, to compute the canonical code of a hexagon graph  $G$ , we first search for the smallest pattern  $P_k$  in  $G$ . Then, for each vertex  $v_i$  such that the pattern of  $v_i$  in  $G$  is equal to  $P_k$ , we compute all codes from the starting edges highlighted in Figure 4. Finally, we return the smallest computed code.

► **Example 9.** Let us consider the graph  $G$  of Figure 1. The pattern of vertex  $a$  (resp.  $b$ ,  $c$ ,  $d$ , and  $e$ ) is  $P_2$  (resp.  $P_2$ ,  $P_{2+1b}$ ,  $P_{1+1a}$ , and  $P_1$ ). The smallest pattern is  $P_{2+1b}$ . Therefore, the canonical code is obtained by running  $\text{BFS}_{\overline{G}}(c, a)$ , and it is useless to run  $\text{BFS}_G$  or  $\text{BFS}_{\overline{G}}$  from any other edge.

Note that this does not change the time complexity as in the worst case we may have  $\mathcal{O}(n)$  vertices with a smallest pattern. For example, when we have an horizontal chain of vertices, all vertices have pattern  $P_{1+1b}$ , except the two endpoints which have pattern  $P_1$ . In this case, we must run  $\text{BFS}_G$  and  $\text{BFS}_{\bar{G}}$  for every edge outgoing from vertices with pattern  $P_{1+1b}$  (i.e., all vertices but the two endpoints).

### 2.3 Comparison with other canonical codes

In the general case, building the canonical code of a graph is a problem which is not known to be in  $\mathcal{P}$  nor to be  $\mathcal{NP}$ -complete (it is isomorphic-complete). There exist rather efficient algorithms such as Nauty [9], but these algorithms have an exponential time complexity in the worst case. Canonical codes are widely used in graph mining tools such as gSpan [12] or Gaston [10]. When graphs are embedded in a 2D space, canonical codes may be computed in polynomial time [8]. These canonical codes may be simplified when considering grid graphs such that all faces are squares [6]. Canonical codes defined in [12, 10, 6] share some similarities with our canonical codes as they are computed by performing graph traversals and assigning numbers to vertices according to the order of discovery. However, in [12, 10, 6], traversals are Depth First Searches (DFSs). Performing BFSs instead of DFSs allows us to avoid some calls to BFS (as explained in Section 2.2). It also allows us to exploit properties  $a$  and  $b$  to define codes by means of constraints (see Section 4).

In [1], canonical codes are introduced for benzenoid structures. These codes only describe boundary cycles and assume that there is no hole within these cycles, thus preventing the representation of coronoids. Our codes fully describe hexagon graphs, including vertices inside the boundary cycle and, therefore, we can represent coronoids.

## 3 Canonical Codes in Case of Required Patterns

Many BGPs involve enumerating hexagon graphs that contain some given patterns, where patterns both specify mandatory and forbidden vertices [3]. More precisely, a pattern is defined by a couple  $(P, F)$  such that  $P$  is a hexagon graph that specifies mandatory vertices, and  $F$  is a set of couples that specify forbidden vertex positions, i.e., for each couple  $(p_i, d_i) \in F$ , vertex  $p_i$  must not have an outgoing edge in direction  $d_i$ .

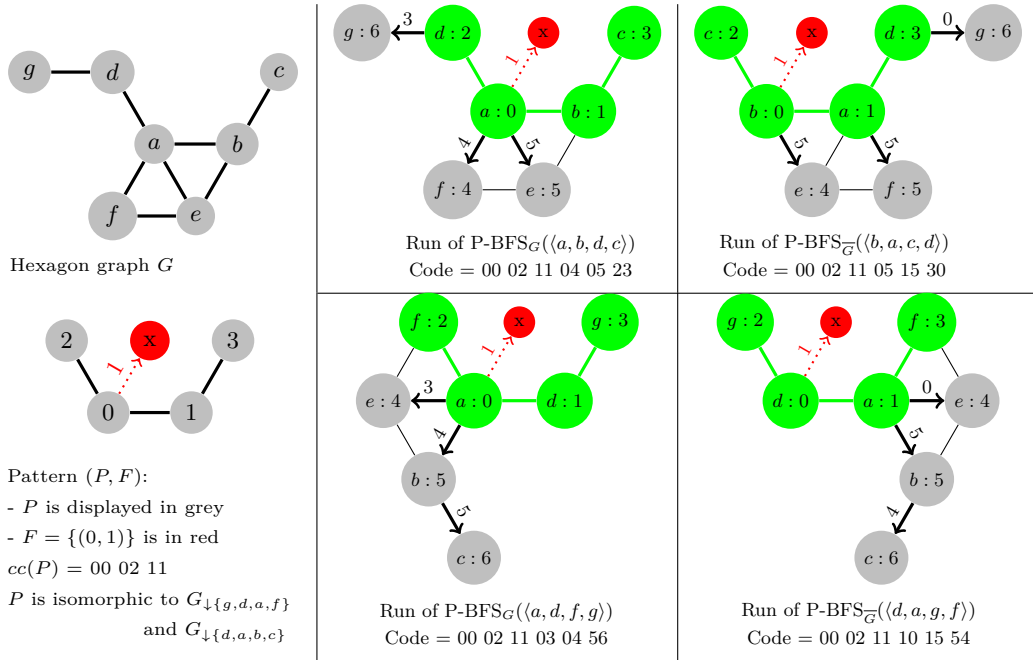
► **Example 10.** The pattern  $(P, F)$  displayed in Figure 5 has four mandatory vertices in grey and one forbidden vertex in red.  $(P, F)$  occurs twice in  $G$  as  $P$  is isomorphic to  $G_{\downarrow\{d,a,b,c\}}$  (resp.  $G_{\downarrow\{g,d,a,f\}}$ ) and  $a$  has no outgoing edge in direction 1 (resp.  $a$  has no outgoing edge in direction 1 when rotating  $G$  so that  $G_{\downarrow\{g,d,a,f\}}$  has the same orientation as  $P$ ).

To efficiently generate graphs that contain a pattern, the idea is to constrain canonical codes to start with this pattern so that all generated graphs contain it by construction. We introduce this new canonical code in Section 3.1, and we study its properties in Section 3.2.

### 3.1 Definition of P-canonical codes

To ensure that codes start with the canonical code of a given pattern  $(P, F)$ , we perform BFSs of  $G$  and  $\bar{G}$  from a given P-sequence that corresponds to an occurrence of  $(P, F)$  in  $G$  or  $\bar{G}$ . More precisely, this P-sequence is defined as follows.

► **Definition 11** (P-sequence of  $(G, P, F)$ ). *Let  $G$  be a hexagon graph with  $n$  vertices, and  $(P, F)$  be a pattern with  $1 < k < n$  mandatory vertices. A P-sequence of  $(G, P, F)$  is a sequence  $s = \langle v_0, v_1, \dots, v_{k-1} \rangle$  of  $k$  vertices of  $G$  such that*



■ **Figure 5** Left: Example of hexagon graph  $G$  and pattern  $(P, F)$ . Right: To compute  $cc_{P,F}(G)$ , we run  $P\text{-BFS}_G$  (resp.  $P\text{-BFS}_{\overline{G}}$ ) with the P-sequences  $\langle a, b, d, c \rangle$  and  $\langle a, d, f, g \rangle$  (resp.  $\langle b, a, c, d \rangle$  and  $\langle d, a, g, f \rangle$ ). The P-canonical code of  $G$  is  $cc_{P,F}(G) = 00\ 02\ 11\ 03\ 04\ 56$ .

- (i) the subgraph of  $G$  induced by  $\{v_0, \dots, v_{k-1}\}$ , i.e.,  $G_{\downarrow s}$ , is isomorphic to  $P$ ;
- (ii)  $\forall (p_i, d_i) \in F$ ,  $G_{\downarrow s}$  has no outgoing edge from the vertex corresponding to  $p_i$  in direction  $d_i$  when rotating  $G$  so that  $G_{\downarrow s}$  and  $P$  have the same orientation;
- (iii) the code returned by  $BFS_{G_{\downarrow s}}(v_0, v_1)$  is canonical;
- (iv) for each  $i \in [0, k - 1]$  the number assigned by  $BFS_{G_{\downarrow s}}(v_0, v_1)$  to  $v_i$  is equal to  $i$ , i.e.,  $num[v_i] = i$  at line 13 of Algorithm 1.

Each P-sequence corresponds to an occurrence of  $(P, F)$  in  $G$ , and  $(P, F)$  may occur more than once in  $G$ . To compute all P-sequences of  $(G, P, F)$ , we iterate on each edge of  $G$  and try to build a P-sequence that starts with this edge using Algorithm 2: given the canonical code  $c$  of  $P$ , an edge  $(v_0, v_1)$  of  $G$ , and a set  $F$  of forbidden positions,  $seq(G, v_0, v_1, c, F)$  returns the P-sequence of  $(G, P, F)$  that starts with  $\langle v_0, v_1 \rangle$  if it exists, and it returns *null* otherwise. To build the P-sequence, it uses the canonical code  $c$  to associate vertices of  $G$  to vertex numbers used in the canonical code: for each vertex number  $i \in [0, k - 1]$ ,  $vertex[i]$  is the vertex of  $G$  that corresponds to  $i$ , and for each couple  $p_i d_i$  in the canonical code, we ensure that vertex  $i$  in  $P$  corresponds to the vertex of  $G$  that is at direction  $d_i$  from  $vertex[p_i]$ , if it exists (line 5); if it does not exist, then there is no P-sequence that starts from  $(v_0, v_1)$  and we return *null* (line 4). When a P-sequence is completed, we check that there are no vertices at forbidden positions (line 6).

To be invariant to symmetries, we must also compute all P-sequences of  $(\overline{G}, P, F)$  by running  $seq(\overline{G}, v_0, v_1, c, F)$  for each edge  $(v_0, v_1)$  of  $\overline{G}$ .

► **Example 12.** Let us consider the graph  $G$  and the pattern  $(P, F)$  of Figure 5. The canonical code of  $P$  is 00 02 11, and it may be computed either by  $BFS_P(0, 1)$  or by  $BFS_{\overline{P}}(1, 0)$  because  $P$  is symmetrical. Also, this pattern is isomorphic to two subgraphs of  $G$ . Hence, the four runs of  $seq$  that return a P-sequence corresponding to an occurrence of  $(P, F)$  are:

■ **Algorithm 2**  $\text{seq}(G, v_0, v_1, c, F)$ .

---

**Input:** A hexagon graph  $G$ , an edge  $(v_0, v_1)$  of  $G$ , the canonical code  $c = p_1 d_1 \dots p_{k-1} d_{k-1}$  of a pattern  $P$ , and the set  $F = \{(p'_1, d'_1), \dots, (p'_f, d'_f)\}$  of forbidden vertices

**Output:** A P-sequence of  $(G, P, F)$  starting with  $\langle v_0, v_1 \rangle$ , or *null* if such a P-sequence does not exist

- 1 rotate  $G$  so that the direction of edge  $(v_0, v_1)$  is equal to 0
- 2  $\text{vertex}[0] \leftarrow v_0$ ;  $\text{vertex}[1] \leftarrow v_1$
- 3 **for**  $i \in [2, k-1]$  **do**
- 4     **if**  $\text{vertex}[p_i]$  does not have an outgoing edge in direction  $d_i$  **then return null**;
- 5      $\text{vertex}[i] \leftarrow$  vertex at direction  $d_i$  from  $\text{vertex}[p_i]$
- 6 **if**  $\exists (p'_i, d'_i) \in F$  s.t.  $\text{vertex}[p'_i]$  has an outgoing edge in direction  $d'_i$  **then return null**;
- 7 **return**  $\langle \text{vertex}[0], \text{vertex}[1], \dots, \text{vertex}[k-1] \rangle$

---

- $\text{seq}(G, a, b, 00\ 02\ 11, \{(0, 1)\}) = \langle a, b, d, c \rangle$  corresponding to  $0 = a, 1 = b, 2 = d, 3 = c$ ;
  - $\text{seq}(\overline{G}, b, a, 00\ 02\ 11, \{(0, 1)\}) = \langle b, a, c, d \rangle$  corresponding to  $0 = b, 1 = a, 2 = c, 3 = c$ ;
  - $\text{seq}(G, a, d, 00\ 02\ 11, \{(0, 1)\}) = \langle a, d, f, g \rangle$  corresponding to  $0 = a, 1 = d, 2 = f, 3 = g$ ;
  - $\text{seq}(\overline{G}, d, a, 00\ 02\ 11, \{(0, 1)\}) = \langle d, a, g, f \rangle$  corresponding to  $0 = d, 1 = a, 2 = g, 3 = f$ .
- All other runs of  $\text{seq}$  return *null*.

In Figure 6, the graph  $G$  has only one P-sequence for pattern  $P$  (i.e.,  $\text{seq}(G, 0, 1, cc(P), F) = \langle 0, 1, 2, 3 \rangle$ ).  $\text{seq}(G, 2, 4, cc(P), F) = \text{null}$  because  $G$  already has a vertex at direction 4 from 2 when rotating  $G$  so that the direction of edge  $(2, 4)$  is equal to 0.

Given a hexagon graph  $G$  and a P-sequence  $s = \langle v_0, v_1, \dots, v_{k-1} \rangle$ , we define an algorithm, called  $\text{P-BFS}_G(s)$ , which performs a BFS of  $G$  that starts from  $s$ , and which is obtained from Algorithm 1 by replacing lines 3 and 4 by the following lines:

---

display the canonical code of  $G_{\downarrow s}$   
let  $q$  be an empty FIFO queue  
**for**  $i \in [0, k-1]$  **do** set  $\text{num}[v_i]$  to  $i$ , and add  $v_i$  in  $q$   
initialise the counter  $c$  to  $k$

---

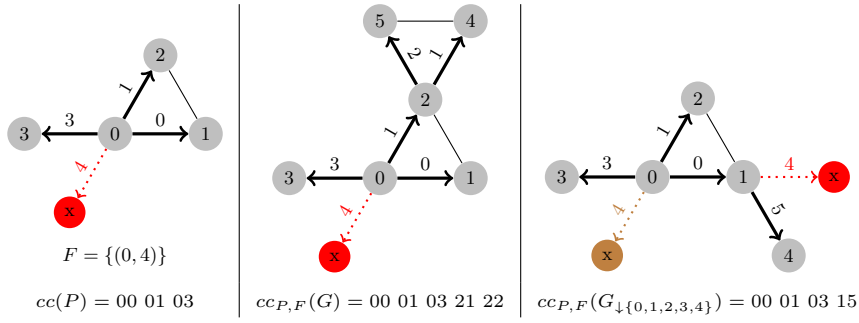
These lines ensure that the code returned by P-BFS starts with the canonical code of the pattern associated with  $s$  so that this pattern is included in the returned code by construction. The numbers assigned to the vertices of  $s$  correspond to those assigned by BFS when computing the canonical code of the pattern and we add the vertices of  $s$  in  $q$  to ensure that these vertices are treated first.

The P-canonical code of  $(G, P, F)$  is the smallest code obtained from any P-sequence.

► **Definition 13** (P-canonical code  $(cc_{P,F}(G))$ ). *The P-canonical code of  $G = (V, E)$  with respect to a pattern  $(P, F)$  is  $cc_{P,F}(G) = \min \{P\text{-BFS}_G(s) : s \in S\} \cup \{P\text{-BFS}_{\overline{G}}(s) : s \in \overline{S}\}$  where*

- $S = \{\text{seq}(G, v_0, v_1, cc(P), F) : (v_0, v_1) \in E \wedge \text{seq}(G, v_0, v_1, c, F) \neq \text{null}\}$ ,
- $\overline{S} = \{\text{seq}(\overline{G}, v_0, v_1, cc(P), F) : (v_0, v_1) \in E \wedge \text{seq}(\overline{G}, v_0, v_1, c, F) \neq \text{null}\}$ .

► **Example 14.** Let us consider the graph  $G$  and the pattern  $(P, F)$  of Figure 5. As seen in Example 12, there are four different candidate P-sequences  $s$  from which we may compute a code. We display in Figure 5 the runs of P-BFS from these four P-sequences.



■ **Figure 6** Example of P-canonical code with a non P-canonical prefix. Left: Pattern  $(P, F)$ . Middle: Graph  $G$ . Right:  $G_{\downarrow\{0,1,2,3,4\}}$ . The prefix 00 01 03 21 of  $cc_{P,F}(G)$  corresponds to  $G_{\downarrow\{0,1,2,3,4\}}$  and it is not P-canonical as  $cc_{P,F}(G_{\downarrow\{0,1,2,3,4\}})$  is smaller.

### 3.2 Properties of P-canonical codes

The time complexity for computing  $cc_{P,F}(G)$  when  $G$  has  $n$  vertices,  $P$  has  $k$  vertices (with  $k < n$ ), and  $F$  has  $f$  couples is  $\mathcal{O}(n(n+f))$ :

- the canonical code of  $P$  is computed in  $\mathcal{O}(k^2)$ ;
- Algorithm 2 is in  $\mathcal{O}(k+f)$ , and it is called  $\mathcal{O}(n)$  times to build all P-sequences;
- P-BFS is in  $\mathcal{O}(n)$  and it is called once per P-sequence, i.e.,  $\mathcal{O}(n)$  times.

Properties of Theorem 5 are no longer valid as illustrated in Figure 5: the code returned by  $\text{P-BFS}_G(\langle a, d, f, g \rangle)$  does not satisfy Property a as  $p_3 = 1$  and  $p_4 = 0$ ; the code returned by  $\text{P-BFS}_{\bar{G}}(\langle d, a, g, f \rangle)$  does not satisfy Property b as  $p_3 = p_4 = 1$  whereas  $d_3 = 1$  and  $d_4 = 0$ . This comes from the fact that the canonical code of  $P$  is inserted at the beginning of the sequence, before starting the search. If a vertex of  $G$  corresponding to a pattern vertex has some extra edges outgoing from it in  $G$  but not in  $P$ , then the integer couple associated with this edge is added after  $cc(P)$  and it may violate Properties a or b. However, we can easily show that these properties are satisfied in the second part of the code, corresponding to the vertices that do not correspond to pattern vertices, i.e., given a code  $p_1 d_1 \dots p_n d_n$  returned by  $\text{P-BFS}_G(\langle v_0, \dots, v_{k-1} \rangle)$ , we have:

**Property a'**:  $\forall i \in [k, n-1], p_i \leq p_{i+1}$ ;

**Property b'**:  $\forall i \in [k, n-1], (p_i = p_{i+1}) \Rightarrow (d_i < d_{i+1})$ .

When there are no forbidden vertices (i.e.,  $F = \emptyset$ ), Theorem 6 is still valid, i.e., every prefix of a canonical code is also canonical (the proof is similar to the proof of Theorem 6 and it is omitted due to space limits).

However, this theorem is no longer valid whenever  $F \neq \emptyset$ , i.e., the prefix of a P-canonical code may not be P-canonical, as shown in the following example.

► **Example 15.** In Figure 6, the occurrence of  $(P, F)$  in  $G$  corresponds to  $G_{\downarrow\{0,1,2,3\}}$ . The prefix 00 01 03 21 of  $cc_{P,F}(G)$  is not P-canonical: this prefix corresponds to  $G_{\downarrow\{0,1,2,3,4\}}$  and  $(P, F)$  occurs twice in it (once in  $G_{\downarrow\{0,1,2,3\}}$  with the forbidden vertex displayed in brown, and once in  $G_{\downarrow\{0,1,2,4\}}$  with the forbidden vertex displayed in red). If we extend  $G_{\downarrow\{0,1,2,3,4\}}$  by adding a vertex on the brown position, we obtain the graph  $G$  that contains only one occurrence of  $(P, F)$  and whose P-canonical code starts with a prefix greater than 00 01 03 15. If we extend  $G_{\downarrow\{0,1,2,3,4\}}$  by adding a vertex on the red position, we obtain another graph (not isomorphic to  $G$ ) that also contains only one occurrence of  $(P, F)$ .

This example shows us that a non P-canonical code may be extended to a P-canonical code. This comes from the fact that a pattern may have more occurrences in the graph associated with a prefix of a code  $c$  than in the graph associated with  $c$ , due to forbidden vertices. However, when the pattern is symmetrical (including its forbidden vertices), we may discard some non P-canonical codes. More precisely, we define below the dominated codes that may be discarded because they cannot be extended to P-canonical codes.

► **Definition 16.** (*Dominated code*) Let  $(P, F)$  be a symmetrical pattern,  $G = (V, E)$  be a hexagon graph that contains at least one occurrence of  $(P, F)$ , and  $W \subset V$  be a subset of vertices such that  $G_{\downarrow W}$  is isomorphic to  $P$ . Let  $C$  be the set of codes computed from a  $P$ -sequence  $s = \langle v_0, v_1, \dots, v_k \rangle$  such that  $(v_0, v_1)$  is an edge of  $G_{\downarrow W}$ . A code  $c \in C$  is dominated if there exists another code  $c' \in C$  such that  $c' < c$ .

► **Theorem 17.** For each P-canonical code  $c$  and each prefix  $c'$  of  $c$ ,  $c'$  is not dominated.

**Proof.** This is a straightforward consequence of the fact that all P-sequences that start from an edge  $(v_0, v_1)$  of  $G_{\downarrow W}$  correspond to different automorphisms of  $(P, F)$  and are interchangeable. ◀

► **Example 18.** In Figure 5,  $(P, F)$  is symmetrical. The dominated codes are  $\text{P-BFS}_{\overline{G}}(\langle b, a, c, d \rangle)$  (when  $W = \{a, b, c, d\}$ ) and  $\text{P-BFS}_{\overline{G}}(\langle d, a, g, f \rangle)$  (when  $W = \{a, d, f, g\}$ ). However, we cannot discard  $\text{P-BFS}_G(\langle a, b, d, c \rangle)$  (though it is larger than  $\text{P-BFS}_G(\langle a, d, f, g \rangle)$ ) as it may lead to a P-canonical code if a vertex is added on the forbidden position for  $W = \{a, d, f, g\}$ .

## 4 CP Models for Enumerating Hexagon Graphs

As hexagon graphs may be uniquely represented with canonical codes, we solve BGPs by enumerating canonical codes, and we propose to use CP to achieve this enumeration as this allows us to easily add constraints on the structures. In Section 4.1, we introduce a first CP model for enumerating consistent codes. In Section 4.2, we introduce a global constraint for ensuring that codes are canonical. In Section 4.3, we introduce a CP model for enumerating all structures that contain some given patterns.

### 4.1 CP Model for Enumerating Consistent Codes

We say that a code is consistent if there exist a hexagon graph  $G$  and an edge  $(v_0, v_1)$  of  $G$  such that  $\text{BFS}_G(v_0, v_1)$  or  $\text{BFS}_{\overline{G}}(v_0, v_1)$  return this code. Obviously, the code must satisfy Properties a and b listed in Theorem 5. However, this is not enough because we must also ensure that all vertices have different positions in the plane. To this aim, we associate 2D coordinates with vertices. We assume that all edges have a length equal to two and that vertex 0 has coordinates  $(0, 0)$ . For every other vertex  $i$ , its coordinates depend on the coordinates of its predecessor  $p_i$  and the direction  $d_i$  of edge  $(p_i, i)$ : if the coordinates of  $p_i$  are  $(x, y)$  and  $d_i = 0$  (resp. 1, 2, 3, 4, and 5), then vertex  $i$  is at coordinates  $(x + 2, y)$  (resp.  $(x + 1, y + \sqrt{3})$ ,  $(x - 1, y + \sqrt{3})$ ,  $(x - 2, y)$ ,  $(x - 1, y - \sqrt{3})$ , and  $(x + 1, y - \sqrt{3})$ ).

► **Example 19.** Let us consider the code 00 02 24. Vertex 1 is at coordinates  $(2, 0)$  (because the edge that reaches 1 from 0 has direction 0), vertex 2 is at coordinates  $(1, \sqrt{3})$  (because the edge that reaches 2 from 0 has direction 1), and vertex 3 is at coordinates  $(0, 0)$  (because the edge that reaches 3 from 2 has direction 4). As vertices 0 and 3 have the same coordinates, this code is not consistent.



## 28:12 Using Canonical Codes to Efficiently Solve the BGP with CP

For each vertex  $i \in [1, n]$ , our CP model uses four integer variables  $x_i, y_i, p_i, d_i$ :

$x_i$  is the abscissa of  $i$ , and its domain is  $D(x_i) = [-2n, 2n]$ ;

$y_i$  is the ordinate of  $i$  divided by  $\sqrt{3}$ , and its domain is  $D(y_i) = [-n, n]$ ;

$p_i$  is the vertex which has discovered  $i$ , and its domain is  $D(p_i) = [0, i - 1]$  ( $p_i < i$  because  $i$  cannot be discovered by a vertex  $j$  which has not yet been discovered);

$d_i$  is the direction of edge  $(p_i, i)$ , and its domain is  $D(d_i) = [0, 5]$ .

For vertex 0, we define  $x_0 = 0, y_0 = 0, p_0 = -1$ , and  $d_0 = -1$ . The code associated with these variables is  $p_1 d_1 \dots p_n d_n$

To ensure the consistency of the generated codes, we post the following constraints:

$C_1$ : all vertices have different coordinates, i.e.,  $\forall i, j \in [0, n], i \neq j \Rightarrow x_i \neq x_j \vee y_i \neq y_j$ ;

$C_2$ : Properties a and b (defined in Theorem 5) are satisfied, i.e.,

$$\forall i \in [1, n - 1], p_i \leq p_{i+1} \wedge p_i = p_{i+1} \Rightarrow d_i < d_{i+1};$$

$C_3$ : For each edge  $(p_i, i)$ , coordinates of  $i$  and  $p_i$  are consistent with direction  $d_i$ , i.e.,

$\forall i \in [1, n], (d_i, x_i, x_{p_i}, y_i, y_{p_i}) \in T$  where  $T$  is the table defined as follows:

$$\begin{aligned} T &= \{(0, x + 2, x, y, y) : x \in [-2n, 2n - 2], y \in [-n, n]\} \\ &\cup \{(1, x + 1, x, y + 1, y) : x \in [-2n, 2n - 1], y \in [-n, n - 1]\} \\ &\cup \{(2, x - 1, x, y + 1, y) : x \in [-2n + 1, 2n], y \in [-n, n - 1]\} \\ &\cup \{(3, x - 2, x, y, y) : x \in [-2n + 1, 2n], y \in [-n, n]\} \\ &\cup \{(4, x - 1, x, y - 1, y) : x \in [-2n + 1, 2n], y \in [-n + 1, n]\} \\ &\cup \{(5, x + 1, x, y - 1, y) : x \in [-2n, 2n - 1], y \in [-n + 1, n]\}. \end{aligned}$$

### 4.2 Global Constraint for Ensuring Canonicity

We must ensure that codes are canonical to become invariant to rotations and symmetries. To this aim, we introduce a new global constraint defined below.

► **Definition 20.** *Given an integer value  $n \geq 1$  and a tuple of four integer variables  $(p_i, d_i, x_i, y_i)$  for each  $i \in [1, n]$ , the constraint  $\text{canonical}(\{(p_i, d_i, x_i, y_i) : i \in [1, n]\})$  is satisfied if the code  $p_1 d_1 \dots p_n d_n$  is canonical.*

To propagate this constraint, we maintain a vector  $pat$  such that, for each vertex  $i$ ,  $pat[i]$  is equal to the current pattern of  $i$  (with respect to the edges that have been added so far). We also maintain a matrix  $M$  such that for each vertex  $i$  and each direction  $d \in [0, 5]$ ,  $M[i][d]$  is either equal to the vertex at direction  $d$  from  $i$ , if such a vertex exists, or to -1 otherwise.  $pat$  and  $M$  are updated each time a new couple  $(p_i, d_i)$  is instantiated. This may be done in constant time by exploiting vertex coordinates.

We trigger a failure whenever there is a vertex  $i > 0$  such that  $pat[i] < pat[0]$ , as we have seen in Section 2.2 that the pattern of  $i$  cannot be smaller than the pattern of 0 (according to the order of Definition 8).

Also, we trigger a failure whenever assigned variables define a code prefix which is not canonical, because Theorem 6 tells us that a non-canonical prefix cannot be extended to a canonical code. More precisely, when all variables that occur in a code prefix  $p_1 d_1 \dots p_k d_k$  with  $k \leq n$  are assigned, we check that this prefix is canonical and, if it is not the case, we trigger a failure. To check whether a prefix is canonical or not, we try to build smaller codes by running BFS from other edges than  $(0, 1)$ . We limit the number of BFSs to be performed by exploiting patterns, as explained in Section 2.2.

Finally, we ensure that the degree of every vertex is upper bounded by a value  $g$  which depends on  $pat[0]$ :  $g = 5$  when  $pat[0] = P_5$ ;  $g = 4$  when  $pat[0] \in \{P_4, P_{3+1}, P_3, P_{2+2}\}$ ;  $g = 3$  when  $pat[0] \in \{P_{2+1a}, P_{2+1b}, P_2, P_{1+1+1}\}$ ;  $g = 2$  when  $pat[0] = P_{1+1}$ ; and  $g = 1$  when  $pat[0] = P_1$ . When the degree of a vertex  $i$  reaches  $g$ , we remove  $i$  from the domain of every non-assigned variable  $p_k$  in order to prevent  $i$  from having more outgoing edges than  $g$ .

### 4.3 CP Model for Enumerating Graphs with a Given Pattern

Given a pattern  $(P, F)$  with  $m$  mandatory vertices, the CP model for enumerating all graphs with  $n + 1 > m$  vertices that contain  $(P, F)$  uses four integer variables  $x_i, y_i, p_i$ , and  $d_i$  for each vertex  $i \in [0, n]$ , like in Section 4.1. However, for each pattern vertex  $i \in [0, m - 1]$ , we assign the variables  $x_i, y_i, p_i$ , and  $d_i$  according to the canonical code  $cc(P)$ .

► **Example 21.** For the pattern  $P$  displayed in Figure 6, we set  $x_0 = y_0 = 0, p_1 = p_2 = p_3 = 0, d_1 = 0, d_2 = 1, d_3 = 3, x_1 = 2, y_1 = 0, x_2 = 1, y_2 = 1, x_3 = -2$ , and  $y_3 = 0$ .

Like in Section 4.1, we post constraint  $C_1$  to ensure that all vertices have different coordinates. We modify constraint  $C_2$  as Properties of Theorem 5 are satisfied only for vertices that are not in the pattern, i.e.,  $\forall i \in [m, n - 1], p_i \leq p_{i+1} \wedge p_i = p_{i+1} \Rightarrow d_i < d_{i+1}$ . We post constraint  $C_3$  but with a modified table  $T'$  to ensure that no vertex is at a forbidden position. More precisely, we compute the set  $X_F$  of coordinates of forbidden positions in  $F$  and define  $T'$  from table  $T$  as follows:  $T' = \{(d, x, x', y, y') \in T : (x, y) \notin X_F \wedge (x', y') \notin X_F\}$ .

Finally, we post a global constraint to ensure that the generated codes are P-canonical. This constraint is defined as follows.

► **Definition 22.** *Given the canonical code  $cc(P)$  of a pattern  $P$  with  $m$  vertices, an integer value  $n > m$  and a tuple of four integer variables  $(p_i, d_i, x_i, y_i)$  for each  $i \in [1, n]$ , the constraint  $P\text{-canonical}_{cc(P)}(\{(p_i, d_i, x_i, y_i) : i \in [1, n]\})$  is satisfied if the code  $p_1d_1 \dots p_nd_n$  is P-canonical.*

To propagate this constraint, we cannot trigger a failure whenever assigned variables define a code prefix which is not P-canonical as we have seen in Section 3.2 that Theorem 6 is no longer valid. However, we exploit Theorem 17 to trigger a failure whenever the current prefix is dominated. Also when all  $p_i$  and  $d_i$  variables are assigned, we check that the code is P-canonical and trigger a failure whenever this is not the case.

Some BGPs involve enumerating graphs that contain two given patterns  $(P, F)$  and  $(P', F')$ . Without loss of generality, we assume that  $\#P \geq \#P'$  where  $\#G$  denotes the number of vertices of a graph  $G$ . In this case, we post the constraint  $P\text{-canonical}_{cc(P)}$  to ensure that all generated codes contain pattern  $(P, F)$  by construction. To ensure that the graph  $G$  associated with the generated code also contains pattern  $(P', F')$ , we post a global constraint that ensures that  $G_{\downarrow[k, \#G]}$  contains an occurrence of  $(P', F')$ , where  $k = \#P$  whenever the two patterns must be disjoint, whereas  $k = 0$  otherwise. This constraint is defined below.

► **Definition 23.** *Given a pattern  $(P, F)$ , two integer values  $k$  and  $n$  such that  $k + \#P \leq n$ , and a tuple of two integer variables  $(p_i, d_i)$  for each  $i \in [0, n]$ , the constraint  $\text{subgraph}_{P,k}(\{(p_i, d_i) : i \in [0, n]\})$  is satisfied if there exists  $S \subseteq [k, n]$  such that  $G_{\downarrow S}$  is isomorphic to  $P$  and there is no vertex at a forbidden position.*

We have implemented a very basic propagator for this constraint: it is not propagated during the search, and we only check that it is entailed when all variables have been assigned. This is done by searching for an edge  $(v_0, v_1)$  of  $G_{\downarrow[k, n]}$  such that  $\text{seq}(G_{\downarrow[k, n]}, v_0, v_1, cc(P'), F') \neq \text{null}$ . If we do not find such an edge, then the constraint is not satisfied; otherwise we have found an occurrence of  $(P', F')$  in  $G_{\downarrow[k, n]}$ .

■ **Table 1** Results of BENZAI and CCODE for BGP1 and BGP2. For each  $n \in [2, 10]$  and each problem, we report the number of solutions (#sol) and the time in seconds of BENZAI and CCODE.

	BGP1			BGP2		
	#sol	BENZAI	CCODE	#sol	BENZAI	CCODE
$n = 2$	1	0.00	0.01	1	0.01	0.01
$n = 3$	3	0.00	0.01	2	0.01	0.01
$n = 4$	7	0.05	0.03	5	0.01	0.02
$n = 5$	22	0.05	0.11	12	0.01	0.07
$n = 6$	81	0.14	0.22	36	0.07	0.15
$n = 7$	331	0.67	0.30	118	0.20	0.28
$n = 8$	1436	19.58	1.08	412	3.88	0.40
$n = 9$	6510	392.24	4.66	1492	19.78	2.21
$n = 10$	30129	22874.00	25.16	5587	732.75	7.90

## 5 Experimental Evaluation

In this section, we experimentally evaluate our approach on four different BGPs (see [2, 3, 4] for more details):

**BGP1** enumerates hexagon graphs without single vertex holes (where a single vertex hole is a position without vertex which is surrounded by a cycle of 6 vertices);

**BGP2** enumerates catacondensed graphs, i.e., hexagon graphs without single vertex holes and without cliques of order 3;

**BGP3** enumerates hexagon graphs without single vertex holes and that contain a given pattern (taken in the list of eight patterns used in [3] and recalled in Appendix A);

**BGP4** enumerates hexagon graphs without single vertex holes and that contain two given patterns (taken in the list of eight patterns used in [3] and recalled in Appendix A) so that the two patterns are not overlapping. (In [3], a variant is considered where patterns may share vertices; we do not display results for this variant as the conclusions are very similar.)

For each problem, the number  $n$  of vertices is given, and we report results for  $n \in [2, 10]$ .

We consider the CP models described in Section 4, implemented in Choco [11], and available at <https://gitlab.inria.fr/xipeng/bgp>. To forbid single vertex holes, we exploit *pat* and *M* data structures. To forbid cliques of order 3, we ensure that, for each vertex  $i$ ,  $pat[i] \in \{P_{1+1+1}, P_{1+1a}, P_{1+1b}, P_1\}$  as all other patterns contain a clique of order 3.

Our approach, denoted CCODE, is compared with the CP-based approach of [3], denoted BENZAI, and we used the Choco implementation available at <https://github.com/benzai-team/BenzAI>. We do not compare our approach with the dedicated approach of [1] as it cannot enumerate hexagon graphs with holes (for BGP1, this approach finds less solutions than our approach), and it is not possible to add extra constraints in a declarative way so that it cannot be used to solve BGP2, BGP3, or BGP4 without modifying the code, or performing a post-processing to remove solutions that do not satisfy the extra constraints.

All experiments are carried out on an Intel Core Xeon E5-2623v3 of 3.0GHz×16 with 32GB of RAM.

In Table 1, we report CPU times of BENZAI and CCODE on BGP1 and BGP2. For BGP1, BENZAI is faster when  $n \leq 3$ , but CCODE is faster when  $n \geq 4$  and when  $n = 10$  it is 824 times as fast. For BGP2, BENZAI is faster when  $n \leq 7$ , but CCODE is faster when  $n \geq 8$  and when  $n = 10$  it is 93 times as fast. In Table 2, we report CPU times of BENZAI

■ **Table 2** Results of BENZAI and CCODE for BGP3 and BGP4. For each  $n \in [2, 10]$  and each problem, we report the number of instances (#i) and the average, maximum and minimum time in seconds over these instances for BENZAI and CCODE. We report '-' when all runs exceed 3600s.

n	BGP3							BGP4						
	#i	BENZAI			CCODE			#i	BENZAI			CCODE		
		avg	max	min	avg	max	min		avg	max	min	avg	max	min
2	2	0.02	0.02	0.02	0.01	0.01	0.00	0	—	—	—	—	—	—
3	6	0.02	0.03	0.01	0.00	0.01	0.00	0	—	—	—	—	—	—
4	7	0.03	0.04	0.02	0.00	0.02	0.00	3	0.07	0.16	0.02	0.01	0.03	0.00
5	8	0.05	0.07	0.01	0.01	0.07	0.00	11	0.06	0.20	0.04	0.01	0.06	0.00
6	8	0.38	0.43	0.23	0.03	0.15	0.00	23	1.41	2.51	0.93	0.02	0.16	0.01
7	8	0.95	1.06	0.79	0.13	0.35	0.01	29	3.78	5.42	2.30	0.10	0.50	0.01
8	8	20.33	23.84	13.94	0.73	1.61	0.04	34	65.06	83.21	50.28	0.49	1.69	0.03
9	8	306.30	399.25	75.05	3.94	10.52	0.23	35	400.51	574.15	226.87	2.94	10.85	0.20
10	8	-	-	-	34.60	69.75	2.20	36	-	-	-	24.19	71.03	1.61

and CCODE on BGP3 and BGP4. CCODE is always faster. When  $n = 9$ , it is 78 (resp. 136) times as fast as BENZAI for BGP3 (resp. BGP4). When  $n = 10$ , BENZAI is not able to solve any of the 8 (resp. 36) instances of BGP3 (resp. BGP4) within one hour whereas CCODE never exceeds 71s.

## 6 Conclusion

We have introduced a new canonical code for representing hexagon graphs while being invariant to rotations and symmetries. This canonical code is the smallest code associated with a BFS of the graph, and we have shown how to define codes by means of constraints, thus allowing us to generate all consistent codes with CP. We have shown that every prefix of a canonical code is canonical and this property allows us to trigger a failure whenever the current code prefix is not canonical.

We have also shown how to efficiently enumerate all codes that start with a given prefix, thus ensuring that a given pattern is included in the generated graphs. However, in this case it may happen that the prefix of a canonical code is no longer canonical, due to the fact that patterns may specify forbidden vertices. In this case, we exhibit a weaker property which allows us to filter some codes that cannot be extended to a canonical code.

We have experimentally compared our approach with the CP-based approach of [3], and we have shown that it scales much better, especially when there are a lot of solutions to enumerate.

As our approach is based on CP, new properties that must be satisfied by the generated graphs are defined in a declarative way by means of constraints. To this aim, the user has access to four variables for each vertex  $i$ , i.e.,  $p_i$ ,  $d_i$ ,  $x_i$ , and  $y_i$ . In some cases, it may be convenient to be able to add constraints on vertex patterns. This is the case, for example, for forbidding cliques of order 3 (in BGP2). In our first implementation, which is a proof of concept, the pattern of a vertex is maintained with a backtrackable data structure (using *IStateInt* Choco objects). As future work, we plan to introduce integer variables that represent patterns in order to allow the user to easily define new constraints on patterns.

We also plan to improve the propagation of the global constraint *subgraph* in order to detect some inconsistencies earlier: we could adapt Algorithm 2 (seq) in order to search for the largest sequence in polynomial time, and trigger a failure whenever the size of this sequence plus the number of non assigned vertices is smaller than  $n$ .

Finally, we would like to extend our approach to other kinds of graphs such as grid graphs (where vertices are associated with square faces instead of hexagonal faces), for example. We hope this will pave the way for new applications of CP such as, for example, applications that search for patterns in cellular automata as in [7].

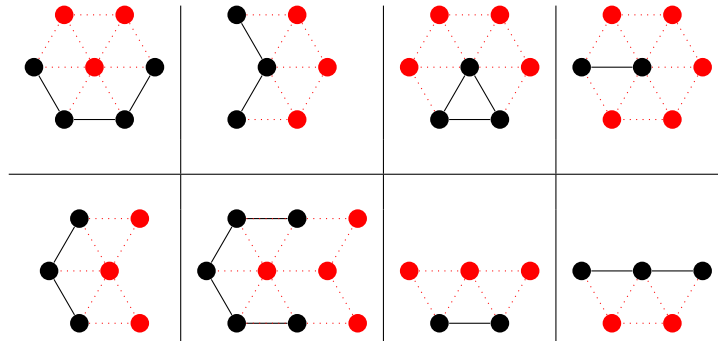
---

## References

- 1 Gunnar Brinkmann, Gilles Caporossi, and Pierre Hansen. A constructive enumeration of fusenes and benzenoids. *J. Algorithms*, 45(2):155–166, 2002.
- 2 Yannick Carissan, Denis Hagebaum-Reignier, Nicolas Prcovic, Cyril Terrioux, and Adrien Varet. Using constraint programming to generate benzenoid structures in theoretical chemistry. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP*, volume 12333 of *Lecture Notes in Computer Science*, pages 690–706. Springer, 2020.
- 3 Yannick Carissan, Denis Hagebaum-Reignier, Nicolas Prcovic, Cyril Terrioux, and Adrien Varet. Exhaustive generation of benzenoid structures sharing common patterns. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP*, volume 210 of *LIPICs*, pages 19:1–19:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 4 Yannick Carissan, Denis Hagebaum-Reignier, Nicolas Prcovic, Cyril Terrioux, and Adrien Varet. How constraint programming can help chemists to generate benzenoid structures and assess the local aromaticity of benzenoids. *Constraints An Int. J.*, 27(3):192–248, 2022.
- 5 S. J. Cyvin, J. Brunvoll, and B. N. Cyvin. Search for concealed non-kekulean benzenoids and coronoids. *Journal of Chemical Information and Computer Sciences*, 29(4):236–244, 1989.
- 6 Romain Deville, Élisabeth Fromont, Baptiste Jeudy, and Christine Solnon. Grima: A grid mining algorithm for bag-of-grid-based classification. In Antonio Robles-Kelly, Marco Loog, Battista Biggio, Francisco Escolano, and Richard C. Wilson, editors, *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR, Proceedings*, volume 10029 of *Lecture Notes in Computer Science*, pages 132–142, 2016.
- 7 Romain Deville, Élisabeth Fromont, Baptiste Jeudy, and Christine Solnon. Mining frequent patterns in 2d+t grid graphs for cellular automata analysis. In Pasquale Foggia, Cheng-Lin Liu, and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition - 11th IAPR-TC-15 International Workshop, GBRPR 2017, Anacapri, Italy, May 16-18, 2017, Proceedings*, volume 10310 of *Lecture Notes in Computer Science*, pages 177–186, 2017.
- 8 Stéphane Gosselin, Guillaume Damiand, and Christine Solnon. Efficient search of combinatorial maps using signatures. *Theor. Comput. Sci.*, 412(15):1392–1405, 2011.
- 9 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014.
- 10 Siegfried Nijssen and Joost N. Kok. The gaston tool for frequent subgraph mining. In *Proceedings of the 2nd International Workshop on Graph-Based Tools, GraBaTs 2004, Rome, Italy, October 2, 2004*, volume 127 of *Electronic Notes in Theoretical Computer Science*, pages 77–87. Elsevier, 2004. doi:10.1016/j.entcs.2004.12.039.
- 11 C. Prud'homme, J.-G. Fages, and X. Lorca. *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. URL: <http://www.choco-solver.org>.
- 12 Xifeng Yan and Jiawei Han. gspan: graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724, 2002. doi:10.1109/ICDM.2002.1184038.

### A Patterns used in BGP3 and BGP4

The eight patterns used in BGP3 and BGP4 are displayed below (mandatory and forbidden vertices are displayed in black and red, respectively).





# Distribution Optimization in Constraint Programming

Guillaume Perez ✉ 

Huawei Technologies Ltd, CSI Paris, Boulogne-Billancourt, France

Gaël Glorian ✉ 

Huawei Technologies Ltd, CSI Paris, Boulogne-Billancourt, France

Wijnand Suijlen ✉ 

Huawei Technologies Ltd, CSI Paris, Boulogne-Billancourt, France

Arnaud Lallouet ✉ 

Huawei Technologies Ltd, CSI Paris, Boulogne-Billancourt, France

---

## Abstract

Stochastic Constraint Programming introduces stochastic variables following a probability distribution to model uncertainty. In the classical setting, probability distributions are given and constant. We propose a framework in which random variables are given a set of possible distributions and only one should be selected. A solution is obtained when all variable distributions are assigned, and all decision variables are assigned too. In such a setting, a constraint on random variables limits the possible distributions its random variables may take. We generalize the notion of *chance* as the probability of satisfaction of a constraint, called *probabilization*, given variable distributions. Probabilization can be seen as a generalization of reification in a random setting whose result is a random variable. We define minimal arithmetic to work with stochastic variables having a variable distribution. Using the introduced representation, our framework can in theory save an exponential number of decisions, and represents problems that were previously not representable with finite integer domains. Finally, we model and solve two industrial problems that require this extension – virtual network configuration and assignment of chemical delivery – and show improvement in terms of quality of solution and speed.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Mathematics of computing → Solvers; Computing methodologies → Probabilistic reasoning

**Keywords and phrases** Constraint Programming, Optimization, Stochastic Optimization

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.29

## 1 Introduction

Stochastic optimization and chance-constrained programming [6, 41] are classes of problems in which uncertainty is present. In such a setting, both decision variables and random variables are present. Usually the probability distributions of the random variables are known and constant. The goal is to optimize a given objective function on these variable sets and to satisfy a set of constraints with sufficient probability. Optimal production planning, optimal power flow, textile manufacturing, vehicle sharing, and parcel delivery services are just a few of the many industrial areas where stochastic optimization is required [28, 29, 34, 55, 21].

Learning probabilistic distributions, or distribution learning, is a machine learning framework that consists of learning the probability distribution that could generate a given set of samples [22, 24]. In the usual settings, the input is a set of samples drawn from an unknown distribution and the goal is to uncover this unknown distribution. Since, many methods have been proposed, for example using assumptions on the class of probability distribution to be a mixture of Gaussian or a Poisson law etc. [10, 9], or directly using neural networks [1].



© Guillaume Perez, Gaël Glorian, Wijnand Suijlen, and Arnaud Lallouet;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 29; pp. 29:1–29:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Consider the virtual network functions design problem [12, 13, 16, 48]. The main part of this problem consists of selecting the settings of different nodes of a network function, such that the global computation latency is robust. Robustness here implies that the total latency (i.e. the sum of the latency of each node) is smaller than a given value, with a probability of at least  $\gamma$ . Locally, for each node the latency distribution is given by a random variable whose distribution is conditioned by the settings of the node. Different settings will lead to different probability distributions for the nodes.

In this paper, we propose to work on *probability distribution optimization*. In this setting, the input is a stochastic constraint optimization problem, and the goal is to find both the assignment of the decision variables and the distribution of the random variables. This can be seen as a generalization of distribution learning in the sense that it is not restricted to the fitting constraints.

Constraint Programming (CP) is an expressive optimization framework often used to solve combinatorial problems such as scheduling. In CP, optimization under chance, confidence, probability, or statistical constraints is a prolific research area [32, 40, 38, 37, 19, 27]. The early and impacting works on stochastic constraint programming defined the basics [14, 53]. Then, optimization methods for chance constraints, Markov, or sampling probability distribution constraints etc. have been proposed [49, 18, 46, 39, 30]. These works have focused either on the multi-stage framework or on one global constraint to extend the optimization process to a stochastic context. But they all consider random variables with fixed distributions.

In this paper, the stochastic CP framework is extended to handle distribution optimization. It is another set of stochastic problems where random variables are given a domain of possible distributions, from which only *one* should be selected. The abstract object *probability distribution variable* is introduced for modeling purposes. It represents the variability of its sample space and its probability. A random variable is *assigned* when its associated probability distribution is known and fixed. *Distribution variables* are now one of the many variables of the problem to solve. A solution is found when all the decision variables are assigned, and when the distribution of all the random variables is fixed. In addition, we propose the definition of *distribution constraints*, which are constraints involving distribution variables. In the *hard* case, a distribution constraint restricts the possible distributions of the random variables it involves. Using the introduced representation, our framework can in theory save an exponential number of decisions, and represents problems that previously could not be represented with a finite integer domain.

Then, we propose to focus on a particular case of constraints namely *relational constraints*. These constraints represent the usual CP constraints, as they are defined by the set of allowed tuples. For these constraints, we propose two new consistency levels, namely  $P$  consistency, for probability consistency, and  $\Omega$  consistency, which is a probability distribution encoding of the usual arc consistency. A direct implication is that most existing stochastic constraints (confidence, sampling, PMF, etc.) that consider the distribution as data of the problem, can be upgraded to deal with variable distributions. That is why in this paper we extend the Confidence/Chance of relational constraints [49, 30, 36]. Furthermore, the notion of *probabilization* of a constraint is proposed. It is a generalization of the chance concept [49], closely related to the reification of constraint, but for probability purpose. The *probabilization* returns a Boolean random variable associated to a distribution variable representing the probability of satisfaction of the constraint.

Distribution variables are abstract objects that can be implemented in diverse ways. We propose an implementation of distribution variables using a direct encoding as a probability mass function. For each possible value of a random variable, we define its probability as a continuous variable. Then, we propose filtering rules defining the minimal arithmetic allowing to model distribution optimization problems.

Finally, we exploit this implementation to solve two distribution optimization problems in the experimental section. The first problem is the stochastic design of virtual network functions, where a simple, yet efficient, model allows to find optimal designs. The second problem we model is a chemical delivery application. It shows how using *variable distributions* allows to have smaller search trees and higher quality solutions.

## 2 Related Work

Chance constrained optimization [6, 31, 33] is a prolific research area. They are classes of problems in which uncertainty is present. In such setting, both decision variables and random variables are present. Usually the probability distributions of the random variables are known and constant. The first definition was given for discrete distributions and piece-wise linear functions with linear inequalities involving random variables. These inequalities had to be maintained at a given level of probability. In these problems, the random variables can be defined using a joint probability distribution, making them conditional on each other [8], which usually makes the problem harder to solve. This framework has been used for the case where the distribution of random variables was conditioned on storage levels or stream flows [20]. Moreover, joint chance constrained optimization problems are problems that contain multiple uncertain constraints on multivariate random variables. They are jointly required to be satisfied with probability exceeding a threshold [54]. However, it is uncommon to have the probability distribution defined as conditioned by the decision variables, or even constrained. The work proposed in this paper is part of chance constrained optimization and aims to use a constraint programming solver as a modeling and solving framework for problems where finding a distribution that satisfies constraints is also part of the problem.

The probabilistic graphical model community proposed Bayesian networks and influence diagram framework [35, 26] to represent the interactions between decision variables (agents) and the probability distributions of random variables. Such influence diagrams, while they tend to grow exponentially, could be powerful tools to represent the constrained distribution during the optimization process. Several methods have been defined to solve chance constrained optimization. In general, non-linear programming solvers are used [28] and dedicated models are defined. For example, dynamic programming coupled with influence diagram has been used to optimize agent decisions to maximize utility functions [50], in a similar way as reinforcement learning. Furthermore, it is not unusual to use sampling methods to approximate chance optimization [3, 33].

In CP, the *chance* constraint is defined as a policy under uncertainty, guaranteeing robustness of the assignment [53, 49]. More precisely, consider a problem where first a set  $X_1$  of decision variables are assigned. Next, some random variables  $Y_1$  reveal their value, then values are selected for the set of variables  $X_2$  and so on for as many iterations as it is required. The solution of such problem is a tree, and no longer a tuple, for which the specified fraction of the scenarios is satisfied. Since then, this work has been extended multiple times, [17, 18] proposed generic algorithms reusing existing filtering for the global chance constraint. More recently, [30] proposed to restrict the chance constraint to 2 stages optimization, as it is a widely used approach. They proposed a filtering for the conjunction of binary inequalities. Finally, [36] proposed to use MDDs as a new generic filtering for the confidence constraint. All of these are different from the work proposed in this paper. On the one hand, policies are not considered, the proposed framework is restricted to 2-stages similarly to [30, 36]. On the other hand, the proposed framework is the first introducing variable distribution inside of constraint programming. Note that it is one of the possible extensions to stochastic CP

as mentioned in the *Extensions* section of [53]. Finally, a recent trend called randomness optimization [23, 2, 15]. is used to optimize the randomness of the solution of a decision problem. For example randomness of solution is important for the design of systems to prevent reverse engineering. Random solution as a constraint is already part of the constraint programming framework [42].

### 3 Distribution Optimization

#### 3.1 Preliminaries

A sample space  $\Omega$  is a set representing all the possible outcomes of an experiment. In the general case, the probability of an outcome  $v \in \Omega$  is noted  $\Pr[v]$ . An event is a set of outcomes from the sample space and an event space  $\mathcal{F}$  is a set of events. Events are often useful for characterising particular subsets of outcomes. For example, the latency of a process might be a real number, but the probability will be usually defined by segments of time (i.e. intervals). Finally,  $P$  is a probability function that maps each element  $e$  of  $\mathcal{F}$  to a probability  $P(e) \in [0, 1]$ . These three elements form a *probability space*, denoted by the triplet  $\psi = (\Omega, \mathcal{F}, P)$ . A random variable  $y$  follows a probability space  $\psi$  if its distribution is defined by the probability space  $\psi$ . This will be noted  $y \sim \psi$ . When  $\Omega$  is discrete or countable, a probability mass function (PMF)  $pmf: \Omega \rightarrow \mathbb{R}$  assigns a probability  $\Pr[v]$  to each value  $v \in \Omega$ . Let  $Y = (y_1, \dots, y_r)$  be a vector of  $r$  independent random variables. By definition, the probability of a sample of independent variables (i.e. a tuple) is given by the product of the probabilities of the selected values. More precisely, the probability of a tuple  $t = (a_1, \dots, a_r)$  is defined by:  $\Pr[Y = t] = \prod_{i=1}^r \Pr[y_i = a_i]$ . When a subset  $Y$  is composed of variables that are not independent, it is possible to replace them by a single random variable  $Y^*$  representing their aggregation [7]. The domain of outcomes of this variable is the Cartesian product of the outcomes of the variables of  $Y$ . So we can assume in this paper that all random variables are independent.

In the rest of this paper,  $X = \{x_1, \dots, x_r\}$  denotes a set of variables (usually integer).  $Y = \{y_1, \dots, y_k\}$  denotes a set of random variables. Given a constraint  $C$ ,  $\mathbb{T}_C(v_1, \dots, v_n) = 1$  (resp.  $= 0$ ) implies that constraint  $C$  is satisfied by the tuple  $(v_1, \dots, v_n)$  (resp. unsatisfied).  $D(x)$  denotes the current domain of variable  $x$ . We denote by  $\overline{D(x)}$  (resp.  $\underline{D(x)}$ ) the upper bound (resp. lower bound) of variable  $x$ . we denote by  $v \in D(x)$  the fact that a value  $v$  belongs to the domain of a variable  $x$ . In a similar way, let  $\forall t \in D^\times(X) = \prod_{i=1}^n D(X_i)$  denotes all the tuples in the Cartesian product of the current domain of variables in  $X$ .

#### 3.2 Distribution Variables

In this section, we extend the CP framework with the definition of *probability distribution variables*.

► **Definition 1.** *A probability distribution variable  $r$  is a variable defining a probability space. The domain  $D(r)$  of a distribution variable  $r$  is a set of probability spaces.*

Recall that a probability space defines its sample space  $\Omega$ , its event space  $\mathcal{F}$  and the probability  $P$  of each event in it. *Distribution variables* are now one of the many variables of the problem, similar to any other decision variable. A probability distribution variable is assigned when the probability of each value in its event space is known and fixed. Let  $r$  be a random distribution variable.  $\psi \in D(r)$  denotes the membership of a probability space  $\psi$  to the current domain of the distribution variable  $r$ . We denote by  $y \sim r$  the

fact that the random variable  $y$  will be drawn following the distribution variable  $r$ , and by  $y \sim \psi$  that random variable  $y$  follows the distribution defined by the probability space  $\psi$  (a constant). Let  $y_1$  and  $y_2$  be two random variables such that  $y_1 \sim r$ ,  $y_2 \sim r$ . Such notation implies that they are independent and identically distributed (IID). We denote  $D(y)$  or  $D(r)$  the domain of the probability distribution variable of random variable  $y \sim r$ . Let  $\forall \Psi \in D^\times(Y)$  denotes all the tuples of probability spaces in the Cartesian product of the current domain of distribution variables of  $Y$ . Let  $\Omega_y$  be the set of all the outcomes such that there exists a probability space in  $D(y)$  where the outcome has a non-zero probability. Let  $\forall t_Y \in D_\Omega^\times(Y)$  denote all the tuples of outcomes (i.e. value) in the Cartesian product of the current sample space  $\Omega_y$  of random variables in  $y \in Y$ . We note by  $\overline{Pr}[Y = t]$  (resp.  $\underline{Pr}[Y = t]$ ) the upper bound (resp. lower bound) probability of tuple  $t$  to be drawn by random variables in  $Y$ . We have  $\overline{Pr}[Y = t] = \max\{Pr[Y \sim \Psi = t] \mid \Psi \in D^\times(Y)\}$  and  $\underline{Pr}[Y = t] = \min\{Pr[Y \sim \Psi = t] \mid \Psi \in D^\times(Y)\}$ . All along the paper,  $\Psi$  represents a vector of probability spaces,  $\psi$  a probability space, and  $\Psi[\{y\}]$  the probability space associated to variable  $y$  in  $\Psi$ . Finally,  $\mathbb{P}$  is the space of all the possible probability spaces (i.e.  $\forall \psi \in \mathbb{P}$ ).

It is interesting to note that integer variables can be represented by the proposed distribution variables.

► **Proposition 2.** *A integer variable always has an equivalent distribution variable.*

**Proof.** Let  $x$  be an integer variable with domain  $D(x)$ . Let  $r$  be a distribution variable such that  $D(r) = \{(\{v\}, \{\emptyset, \{v\}\}, Pr[v=1]) \mid \forall v \in D(x)\}$ . In other words, for each value  $v$  in the domain of  $x$ , there is a probability space whose sample space  $\Omega = \{v\}$  is restricted to value  $v$ . There is a one-to-one correspondence between the values in the domains of  $r$  and  $x$ , hence  $x$  and  $r$  are equivalent. Note that the converse is not true as the number of probability spaces in a distribution variable may not be countable. In the rest of this paper, without loss of generality, we assume that all the integer variables have been replaced by equivalent distribution variables for simplicity of notation. ◀

### 3.3 Distribution Constraints

A random distribution variable being a modeling object of CP, it can be constrained. In this section, we propose a basic definition of distribution constraints. We use this definition to derive a generalization of arc consistency to distribution variables. Finally, we propose to focus on a sub-part of the constraint space, namely relational constraints, that generalizes usual CP constraints.

Distribution constraints are all the constraints involving random variables with variable distribution. A distribution constraint defines the possible probability spaces of the random variables it involves. Integer constraints can always be defined by the set of satisfying tuples. This basic notion is extended to distribution constraints:

► **Definition 3.** *Let  $C$  be a constraint defined on random variables  $Y$ .  $C$  is defined by a set of tuples  $T_R$  such that  $\forall \Psi \in T_R$ ,  $\Psi$  is a tuple of valid probability spaces for the probability distribution of random variables in  $Y$ .*

► **Example 4.** Table of PMFs The following table represents a constraint on two variables: integer variable  $x$  and random variable  $y \sim r$ .

$Pr[x = 0]$	$Pr[x = 1]$	$Pr[y = 1]$	$Pr[y = 2]$	$Pr[y = 3]$	$Pr[y = 4]$	$Pr[y = 5]$	$Pr[y = 6]$
1	0	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$
0	1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$

Such a table restricts the distribution variables of  $x$  and  $y$  to two probability spaces, defined by probability mass functions.  $x$  is a integer variable encoded as a probability distribution. The possible probabilities are either 0 or 1.  $y$  is a true random variable. Depending on the assigned value of  $x$ , random variable  $y$  will behave differently. Note that PMFs are not the only way to encode probability spaces.

The notion of consistency is of utmost important in constraint programming [45]. We propose to directly map the generalized arc consistency to distribution variables. The principal difference is that the domain of a distribution variable is a set of probability spaces, and not a set of integers as for integer variables for example. Let  $C$  be a constraint defined on random variables  $Y$  defined by a set of tuples of probability spaces  $T_R$ . The consistency properties are defined by:

- A probability space  $\psi \in D(y)$  of  $y \in Y$  is consistent with  $C$  if it exists a support  $\Psi \in T_R$  such that  $\Psi[\{y\}] = \psi$  and  $\Psi \in D^\times(Y)$ .
- A distribution variable of  $y \in Y$  is consistent if each probability space in its domain is consistent.
- A constraint  $C$  is generalized arc consistent if all its variables are consistent.

### 3.4 Relational Constraints

Relational constraints are a subset of distribution constraints. They consider the sampled values of the random variables for the satisfiability check of the constraint. In other words, the constraint can always be defined by the possible assignments for the integer variables and for the random variables. They are composed of all the usual constraints of the CP framework but generalized to variable distributions.

► **Definition 5.** *Let  $C$  be a relational constraint defined on random variables  $Y$ .  $C$  is defined by a table of tuples  $T_C$ , such that  $\forall t \in T_C$  is a valid sample that can be drawn for random variables in  $Y$ .*

Note that, for each table  $T_C$ , an infinite number of probability spaces are valid. Indeed, consider the binary constraint defined by the tuples  $[(0, 0), (0, 2)]$  on variables  $x$  and  $y$ . This constraint restricts the sampled value of  $x$  to be 0, and restricts the probability distribution of  $y$  to any distribution such that  $\Pr[y = 0] + \Pr[y = 2] = 1$ . This implies that such a definition can be highly compressing, as an infinite number of distributions satisfy this equality. On the other hand, such a definition is not expressive enough to restrict to all subsets of probability spaces. For example, it is impossible to define  $E(y) = 3$  (expectation), or simply the constraint defined by the table from the example in section 3.3. That is the reason why relational constraints are a subset of distribution constraints.

► **Definition 6 (Relational Constraint).** *Let  $\mathbb{T}_C(t) = 1$  if  $t \in T_C$  and 0 otherwise be a predicate function for constraint  $C$ . We propose to use the following definition for the relational constraint:*

$$\int_{t \in D_\Omega^\times(Y)} \Pr[Y = t] \mathbb{T}_C(t) = 1 \quad ; \quad \int_{t \in D_\Omega^\times(Y)} \Pr[Y = t] \neg \mathbb{T}_C(t) = 0 \quad (1)$$

The most intuitive description of such constraint is that it does not imply that all valid tuples must be reached, but that all reachable tuples must be valid.

The notion of consistency of a probability space in the case of a relational constraint can be redefined using its definition. Given a relational constraint  $C$  defined on random variables  $Y$  by the relational table  $T_C$ .

► **Definition 7.** A probability space  $\psi \in D(y)$  of  $y \in Y$  is consistent with  $C$  if and only if:

$$\exists \Psi \in D^\times(Y), \Psi[\{y\}] = \psi \wedge \forall t_Y \in D_\Omega^\times(Y), \Pr[Y \sim \Psi = t_Y] \leq \mathbb{T}_C(t) \quad (2)$$

It is a direct rewriting of the consistency of distribution constraints using the relational definition (equation (1)). Then, the domain of probability variable  $y \in Y$  is consistent with constraint  $C$  if all the probability spaces in it are consistent with  $C$ . Finally, a constraint  $C$  is consistent if all the variables in it are consistent.

Let  $\psi$  be a probability space and  $Y_y^\psi$  be a random distribution variable vector where variable  $y$  is restricted to probability space  $\psi$ . A weaker consistency can be defined for relational constraints and distribution variables. This consistency is close to bound-consistency. It considers the bounds of the probability of events given the current domain of a distribution variable.

► **Definition 8.** A probability space  $\psi \in D(y)$  of  $y \in Y$  is P-bound consistent with  $C$  if:

$$\forall t \in D_\Omega^\times(Y_y^\psi), \Pr[Y_y^\psi = t] \leq \mathbb{T}_C(t) \quad (3)$$

Then, the domain of probability variable  $y \in Y$  is P-bound consistent with constraint  $C$  if all the probability spaces in it are P-bound consistent with  $C$ . Finally, a constraint  $C$  is P-bound consistent if all the variables in it are P-bound consistent:  $\forall t \in D_\Omega^\times(Y), \Pr[Y = t] \leq \mathbb{T}_C(t)$ . Now that the P-bound consistency is defined, we propose to focus on the case  $\mathbb{T}_C(t) = 0$  as it is the only one able to invalidate probability spaces. Let  $Y_{\setminus\{y\}}$  be the vector of variables  $Y$  without variable  $y$ . Let  $t_{\setminus\{y\}}$  be the tuple  $t$  without the value at the position of variable  $y$ . Propagating the following logical implication is enough to enforce P-bound consistency constraint  $C$ :

$$\forall y \in Y, \forall t \in D_\Omega^\times(Y), \Pr[Y_{\setminus\{y\}} = t_{\setminus\{y\}}] > \mathbb{T}_C(t) \implies \Pr[y = t_y] = 0 \quad (4)$$

This pruning rule implies that all the probability spaces that could lead to an invalid tuple should be removed. It will later be derived to extract filtering rules for the implementation of this framework.

► **Definition 9.** Given a relational constraint  $C$  defined on random variables  $Y$  by the relational table  $T_C$ . A probability space  $\psi \in D(y)$  of  $y \in Y$  is  $\Omega$ -consistent with  $C$  if:

$$\exists t \in D_\Omega^\times(Y_y^\psi), \mathbb{T}_C(t) \quad (5)$$

$\Omega$ -consistency ensures that it exists at least one tuple with non-zero probability to satisfy the predicate given the other domains.  $\Omega$ -consistency, when all the variables are integer variables encoded as distribution variables, is equivalent to global arc consistency as it finds a support for each value in the domain of each variable.

Consider a total order on the outcomes of  $\Omega_y$ . Let  $\overline{\Omega_y}$  (resp.  $\underline{\Omega_y}$ ) be the largest (resp. smallest) element of  $\Omega_y$  with respect to the total order. Let  $D_\Omega^\times(Y)$  denote the Cartesian product of the intervals  $[\underline{\Omega_y}, \overline{\Omega_y}]$ ,  $\forall y \in Y$ .

► **Definition 10.** A probability space  $\psi \in D(y)$  of  $y \in Y$  is  $\Omega$ -bound-consistent with  $C$  if:

$$\exists t \in D_\Omega^\times(Y_y^\psi), \mathbb{T}_C(t) \quad (6)$$

When all the variables are integer variables encoded as distribution variables and the total order is defined by the operator  $<$ ,  $\Omega$ -bound-consistency is equivalent to bound-consistency.

Consider the Confidence constraint that restricts the decision variables to values such that the probability of satisfaction is greater than a given threshold [49, 30, 36]. We propose to generalize it by considering variable distributions for random variables.

► **Definition 11.** Given a relational constraint  $C(Y)$ , with  $Y$  a vector of random variables with variable distributions. Given a confidence threshold  $\gamma$ . The generalized confidence is:

$$\int_{t \in D_{\Omega}^{\times}(Y)} Pr[Y = t] T_C(t) > \gamma \quad (7)$$

Previously, propagation of chance constraint was only removing values from integer variables that could not lead to a robust enough solution. The new propagation of this constraint will impact both the integer variables (here encoded as distributions) and the distribution variables as probability spaces that are not robust enough will be removed too.

### 3.5 Views on Random Variables

We propose to make a clear distinction between the three equality operators: “ $y_1 = y_2$ ”, “ $y_1 \simeq y_2$ ”, and “ $y_1 \equiv y_2 + y_3$ ”. First,  $y_1 = y_2$  is the classic equality constraint, it implies that the value of the random variables must be equal, but they are considered different random variables. Second, the constraint  $y_1 \simeq y_2$  ensures that  $y_1 \sim r_1$  and  $y_2 \sim r_2$  are *equal in law*. More precisely it ensures that  $r_1 = r_2$ . This implies that their random assignment might be different, but the constraint is satisfied if they follow the same distribution. From a probability point of view, they are independent. Finally,  $y_1 \equiv y_2 + y_3$ , where  $y_1$  is the actual summation of the two random variables  $y_2$  and  $y_3$ . From a probability point of view, they are dependent.

Views are a useful abstraction in constraint programming [47, 51]. They prevent the creation of intermediate variables representing some function on a variable ( $x' = f(x)$ ). Views can be adapted to random variables directly. Consider for example the affine view  $y_1 \equiv ay + b$ . Iterating on the values of  $y_1$  is done by iterating on the values of  $y$  and applying the affine transformation. In addition,  $Pr[y = i] = Pr[y_1 = ai + b]$ . In the rest of this paper, notations such as  $-y$  represent a view on random variable  $y$ .

### 3.6 Probability Valuation of Constraints

In constraint programming, the reification of a constraint is very useful while modeling a problem. The reification variable is usually a Boolean variable denoting the satisfiability of the related constraints. Other valuations of constraints have been proposed, such as soft constraints [52, 25] and cost-version of constraints [44]. In this paper, we introduce the probability valuation of a constraint. It is a generalization of the chance from [18] for the case of variable distributions as the result is a Boolean random variable.

► **Definition 12.** Let  $C$  be a relational constraint defined on the random variables  $Y$ . Let  $p_C$  be a Boolean random variable. The probability valuation  $p_C \equiv C(Y)$  is defined by:

$$Pr[p_C = 1] = \int_{t \in D_{\Omega}^{\times}(Y)} Pr[Y = t] T_C(t) \quad (8)$$

Note that in the general case processing the probability of satisfaction of a constraint  $C$  is hard because the table  $T_C$  of the constraint may be untractable.

► **Proposition 13.** The probabilization of a constraint is a generalization of the reification.

**Proof.** Let  $C$  be a relational constraint defined on the random variables  $Y$ . Let  $p_C \in [0, 1]$  be a Boolean random variable such that  $Pr(p_C = 1) = Pr[C(Y)]$ . The two possible values for the reification of a constraint are satisfied and unsatisfied. If  $Pr(p_C = 1) = 1$  the constraint is always satisfied, if  $Pr(p_C = 1) = 0$ , the constraint is always unsatisfied. For all the other values,  $p_C$  represents the probability of satisfaction of the constraint. ◀

Note that the probabilization can be seen as a random variable representing the projection of the distribution of the random variables on the constraint. This implies that  $p_C$  and the random variables in  $C$  are dependents. In the rest of this paper,  $p_C$  will be used to denote  $\Pr[p_C = 1]$  when no ambiguity is present.

### 3.7 Multiple Stochastic Relational Constraints

In chance-constrained optimization, applications can be made of several independent parts [54], and the joint probability must be satisfied. Those are problems for which constraints of the problem can be split into independent ones. Let  $\mathbf{1}$  be the vector of ones. More formally, these problems can be reformulated as:

$$\arg \min_{Y \in \mathbb{D}} F(Y) \quad (9)$$

$$\text{subject to } p_1 \equiv C_1(Y_1) \quad (10)$$

$$\dots \quad (11)$$

$$p_n \equiv C_n(Y_n) \quad (12)$$

$$\Pr[(p_1, \dots, p_n) = \mathbf{1}] \geq \gamma \quad (13)$$

Where  $Y_i \cap Y_j = \emptyset, \forall i, j$ , and  $\cap$  being the set intersection operator. Using the probability valuation of constraints (10-12), the global chance or confidence constraint is (13). This implies that the hard part of the chance constraining will be located into the probability valuation of the constraints.

## 4 Implementation

### 4.1 Distribution Implementation

In the rest of this paper, a possible implementation for the discrete random variables is proposed. Then, propagation algorithms are proposed using the proposed implementation. Implementing a variable distribution *random variable* inside a solver might be challenging, as probability distributions can be expressed in very diverse ways [26]. Indeed, as for set or sequence variables [11], the choice of implementation will impact the design of propagation algorithms, and the capability of representation. For known probabilistic distributions, a straightforward implementation would be the parameters that describe it. For example, it would be sufficient to have two continuous variables  $\mu$  and  $\sigma$  for a normal distribution. Indeed, with such a definition, the propagation could use the closed form of the cumulative distribution function and probability density functions. In this paper, for the implementation of the distribution, we assume that random variables are independent and discrete. We propose to represent distribution variables using probability mass function variables PMF.

► **Definition 14.** *Let  $y$  be a discrete independent random variable with sample space  $\Omega_y = \{v_1, \dots, v_d\}$ . The PMF variable of  $y$  is represented as a vector  $d^y = (d_v^y)_{v \in \Omega_y}$  of continuous variables where  $D(d_v^y) \subseteq [0, 1]$  represents the probability  $\Pr[y = v]$  and  $\sum_i d_i^y = 1$ .*

Note that using continuous or floating point variables inside of CP solver is a known topic [5, 43]. They are usually represented by intervals. For evident reasons, integer variables are considered encoded as usual.

► **Example 15.** Consider a Constraint Satisfaction Problem (CSP) with one decision variable  $x \in \{0, 1\}$  and one random variable  $y \in [1, 6]$ . The CSP contains two constraints: one distribution constraint and one relational constraint. The first one defines the possible



distributions of  $y$  and is defined by the table from the example in section 3.3. The second constraint is the constraint  $\Pr[y > 2] \geq \gamma$  with  $\gamma = 0.6$ . A solution of such a CSP is an assignment of both  $x$  and the  $d_i^y$  variables that satisfies both constraints.

Consider the propagation of the  $y > 2$  constraint. For each of the variables  $d_3^y$  to  $d_6^y$ , the lower bound is set to  $\gamma - 3\frac{1}{6} = 0.1$ . Indeed, for each of the possible values, the cumulative probability must be at least equal to  $\gamma$ . Once the propagation of constraint  $y > 2$  is done, the distribution constraint can filter out value 1 from  $x$  as the associated distribution is no longer valid ( $d_4^y = 0.1 > \frac{1}{12}$ ). Values of  $d^y$  are assigned to the tuple  $(\frac{1}{6}, \dots, \frac{1}{6})$ , and a solution is found.

► **Example 16** (Probability valuation). Consider again the problem of the previous example. Once  $y$  is assigned, the probability valuation of constraint  $y > 2$  is  $p_{y>2} = \sum_{i=3}^6 d_i^y = \frac{4}{6}$ .

## 4.2 Filtering Algorithms for Relational Constraints on Random variables

It is stated in [28] that *the major challenge towards solving chance constrained optimization problems lies in the computation of the probability and its derivatives of satisfying inequality constraints*. In this section, probability filtering is provided for several constraints. Assumptions made by these filtering algorithms are that all random variables are independent and implemented using the distribution variables proposed in this paper. Given  $p_C \equiv C(X, Y)$  the probability valuation of a constraint, the filtering algorithm should filter inconsistent values from  $X$ , from the distributions of  $Y$  and  $p_C$ .

Let  $C$  be a relational constraint defined on the random variables  $Y$ . We propose here a few filtering rules deriving from equations (4) and (8).

► **Example 17** (Unary Equal). Let  $p_{y=c} \equiv (y = c)$  be the probability valuation of the *equal* constraint. In such settings,  $\Pr[p_{y=c} = 1] = d_c^y$ . In addition for all the other values, another filtering can be defined by  $\forall v \neq c, d_v^y \leq 1 - \Pr[p_{y=c} = 1]$ . Note that this filtering should be directly done by the domain definition of the probability distribution.

► **Example 18** (Unary Greater Than). Let  $p_{y>c} \equiv (y > c)$  be the probability valuation of the unary *greater-than* constraint. We propose to decompose equation (8) into two equations on the bounds of the random variable:  $p_{y>c} \geq \sum_{v_i=c+1}^{\Omega(y)} d_{v_i}^y$ ;  $p_{y>c} \leq \sum_{v_i=c+1}^{\Omega(y)} d_{v_i}^y$ . This rewriting is usual in CP, and linear propagation can be applied. Moreover, analogously to the reification, filtering can also be defined for values lower than, or equal to  $c$ . Indeed, as  $p_{y>c}$  is the probability of satisfaction of the constraint,  $p_{y \leq c} = 1 - p_{y>c}$  is the probability of violating the constraint. This implies that the two filtering rules above should be used to propagate  $d_v^y, \forall v \leq c$ .

► **Example 19** (Binary Equal). Let  $p_{y_1=y_2} = \Pr[y_1 = y_2]$  be the probability valuation of the *equal* constraint between two random variables. A filtering can be extracted from equation:  $p_{y_1=y_2} = \sum_i d_i^{y_1} d_i^{y_2}$ .

► **Example 20** (Constraints  $y \simeq y_1$  **op**  $y_2$ ).  $y \simeq y_1 + y_2$  is the definition of the distribution of variable  $y$  from the distributions of variables  $y_1$  and  $y_2$ . It is a probability-based constraint. In the general case, the probability distribution of the sum of two independent variables is given by the convolution:  $\Pr[y = k] = \sum_{i=-\infty}^{\infty} \Pr[y_1 = k - i] \Pr[y_2 = i]$ . This convolution can be used to extract propagation rules for the ternary constraint. For each value  $v_k$  of the possible values of  $y$ , a constraint of the form of  $d_{v_k}^y = \sum_i d_{v_k-i}^{y_1} d_{v_i}^{y_2}$  is posted. Note that the same equation can be defined for the multiplication, subtraction, division, and modulo constraints.

► **Example 21** (Constraints  $y = y_1$  **op**  $y_2$ ). For example,  $y = y_1 + y_2$  is the test of equality between the sampled values of  $y$  and the sum of the sampled values of  $y_1$  and  $y_2$ . This constraint can be expressed using the two constraints above. First, let  $y_{1+2} \equiv y_1 + y_2$  be the distribution of the addition of  $y_1$  and  $y_2$ . Second, let  $p_{y=y_1+y_2}$  be the probability valuation of  $y = y_{1+2}$ . Finally,  $p_{y=y_1+y_2} = \Pr[y = y_{1+2}]$ .

### 4.3 Filtering Relational Constraints on Integer and Random Variables

Consider a relational constraint  $C$  on decision variables  $X$  and on random variables  $Y$ .

► **Proposition 22.** *Set the variable  $d_{v_i}^{y_i}$  from  $y_i \in Y$  for constraint  $C$  to 0 if:*

$$\forall t_X \in D^\times(X), \exists t_Y \in D_\Omega^\times(Y), t_{Y_i} = v_i \wedge \prod_{\forall v_j \in t_Y \setminus \{v_i\}} d_{v_j}^{y_j} > \mathbb{T}_C(t_X, t_Y) \quad (14)$$

This proposition is a direct rewriting of equation equations (4) and (8). In addition to the classical filtering of values in the decision variables if they do not have a valid tuple, another filtering should be defined.

► **Proposition 23.** *Filter value  $v_i$  from the domain of  $x_i \in X$  for constraint  $C$  if:*

$$\exists t_Y \in D_\Omega^\times(Y), \forall t_X = (\dots, v_i, \dots) \in D^\times(X), \prod_{\forall v_j \in t_Y} d_{v_j}^{y_j} > \mathbb{T}_C(t_X, t_Y) \quad (15)$$

Let  $p_{C(X,Y)} \equiv C(X, Y)$  be the probability valuation of the constraint. A direct simple filtering can be extracted from equation (8). In addition, the filtering should be propagated back to the decision variables, for example:

► **Proposition 24.** *Filter value  $v_i$  from the domain of  $x_i \in X$  for constraint  $C$  if:*

$$\forall t_x = (\dots, v_i, \dots) \in D^\times(X), \Pr[C(t_x, Y)] \neq p_C \quad (16)$$

Here, equation (8) is applied to each tuple containing the value  $v_i$ . If none of them satisfies the probability valuation, then the value can be safely removed. Recall that  $p_C$  represents  $\Pr[p_C = 1]$ . Moreover, in general, the  $\neq$  will be checked for  $>$  and  $<$  to filter the bounds of the variables. The exact rule depends on the expression of the satisfaction of the constraint in terms of distribution variables, as shown in the following examples.

► **Example 25** (Lower Than). Let  $p_{x < y} \equiv (x < y)$  be the probability valuation of the *lower-than* constraint. This constraint is one of the most used in chance optimization as it can be used to model the deviation of scheduled amounts [8]. Analogously to the equal constraint, the pruning of variable  $x$  can be defined by the following rule:  $x < \max\{v_j \in x \mid \sum_{v_i=v_j}^{\Omega(y)} \overline{D(d_{v_i}^y)} > \underline{D(p_{x < y})}\}$ . First, the opposite rule can also be defined by using  $\overline{D(p_{x < y})}$ . Then, note that this pruning rule, when both the  $d_v^y$  and  $p_{x < y}$  are fixed, is equivalent to the pruning of [30]. In addition, as the authors proposed, a closed formula may be used to directly extract the bounds of  $x$ . Now that  $x$  is propagated, the probability distribution and valuation should be filtered. This can be done by using the following sum:  $p_{x < y} = \sum_{v_i=x+1}^{\Omega(y)} d_{v_i}^y$ .

► **Example 26** (Equal). Let  $p_{x=y} \equiv (x = y)$  be the probability valuation of the *equal* constraint. First the pruning of variable  $x$  can be defined using the following rule:

$$\forall v \in \Omega(y), \overline{D(d_v^y)} < \underline{D(p_{x=y})} \vee \underline{D(d_v^y)} < \overline{D(p_{x=y})} \implies x \neq v \quad (17)$$

Then, the pruning of the probability valuation  $p_{x=y}$  can be defined using:

$$p_{x=y} \geq \min(\{D(d_v^y) | \forall v \in x\}); p_{x=y} \leq \max(\{\overline{D(d_v^y)} | \forall v \in x\}) \quad (18)$$

Finally, the pruning of the distribution of  $y$  is an adaptation of the usual propagation:

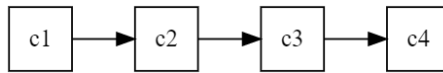
$$x = v \implies d_v^y = p_{x=y} \quad (19)$$

#### 4.4 Rewriting of constraints

In the previous sections, several constraints and propagation algorithms have been proposed. In this section, it is shown how to combine these constraints to model most use cases. Consider the not equal constraint. Even though no dedicated algorithm has been defined, propagation can be based on other ones. Let  $p_{x \neq y} \equiv (x \neq y)$  be the probability valuation of the not-equal constraint,  $p_{x \neq y} = 1 - p_{x=y}$ . This is analogous to the reification, indeed,  $(x_1 = x_2) = \neg(x_1 \neq x_2)$ . Several more constraints can be reformulated, for example the greater-less-than relationship:  $p_{x \leq y} = p_{x-1 < y}$ ;  $p_{x > y} = p_{-x < -y}$  (alternatively,  $p_{x \geq y} = p_{-x-1 < -y}$ ). The in-not in relationship:  $p_{y \notin X} = 1 - p_{y \in X}$ . And the many logical relationships such as  $p_{c_1 \wedge c_2} = p_{c_1} p_{c_2}$ ;  $p_{c_1 \vee c_2} = 1 - (1 - p_{c_1})(1 - p_{c_2})$ ; and  $p_{c_1 \oplus c_2} = p_{c_1}(1 - p_{c_2}) + (1 - p_{c_1})p_{c_2}$ . With  $\oplus$  the xor logical operator. Note that some of these reformulations are direct encodings of the probability rules, such as for the *and* and *or* constraints.

### 5 Application: Virtual Network Design

In virtual network functions design, the main performance criterion is the total latency of the virtual network [12, 13, 16, 48]. In such a graph, each node is either a machine or a virtual machine, and has to process blocks of data. The processing time of a data block by a node is following a random distribution, which is conditioned by its configuration (both hardware and software). The global latency is the sum of the processing times of the nodes of the network. The goal of the virtual network design problem is to set the configurations for all the nodes such that the global latency is below  $L$  milliseconds with probability  $\gamma$ , while the cost is minimized and other configuration constraints are satisfied.



■ **Figure 1** Example of virtual network graph.

Consider the network from Figure 1. This network contains 4 nodes ( $c_1, \dots, c_4$ ). In this graph, when a data block needs to be sent, it will go through  $c_1$ , then  $c_2$ , then  $c_3$  and finally  $c_4$ . Note that in practice, the depth of the graph is often shallow. In our industrial application the number of nodes was 3. Moreover, each node contains several settings (*cpu redundancy*, *queue maximum read times*, *queue batch pkt num*, *dpe*, etc.) The latency of a node is influenced by all these settings. In addition, each node serves a particular role, and some operations must be done in at least one node. All of these requirements are contained in the constraints on the settings variables. The influence of the setting variables on the random variables' distribution is encoded as a table  $T_s$  for each node.

## 5.1 Model

Let  $N = (n_1, \dots, n_k)$  be the nodes to configure. Let  $y_i \sim D^{y_i}$  be the random variable representing the latency of node  $n_i$ . Let  $S = (S^1, \dots, S^k)$  with  $S^i = s_1^i, \dots, s_m^i$  being the settings variables associated to each node. Let  $L$  be the maximum latency and  $\gamma$  the minimal probability. Let  $acc_i \in (acc_2, \dots, acc_k)$  be the accumulated sum of the  $i$  first random variables. Let  $c_i$  be the cost associated with node  $n_i$ .

$$\arg \min_{S \in \mathbb{Z}, Y \in \mathbb{P}} \sum_{i=1}^k c_i \quad (20)$$

$$\text{such that } \text{Table}(S^i, c_i, y_i, T_s), \quad \forall i \in [1, k] \quad (21)$$

$$acc_1 \simeq y_1 \quad (22)$$

$$acc_i \simeq acc_{i-1} + y_i, \quad \forall i \in [2, k] \quad (23)$$

$$\Pr[acc_k < L] \geq \gamma \quad (24)$$

$$\text{ValidSettings}(S) \quad (25)$$

## 5.2 Data

A large number of configurations have been heavily sampled to extract the distribution of probability. Once those were built, they have been used to extrapolate to unknown configurations. The latency is given in micro-seconds, with values in the set (50, 100, ..., 1450, 1500). Each node may be defined by around 100 configurations (valid assignment of the settings). The number of nodes varies between 2 to 5. Two algorithms are compared. First, *Variable* is the work proposed by this paper, where distribution variables are used. The second one is *Fixed*, it is an adaptation of the algorithm from [30, 36]. In *Fixed*, the distributions are considered unknown until all the setting variables used to define them are instantiated. It can be seen as a form of generate and test. The search strategy is the same for both algorithms. It starts by assigning the possible settings. Note that this is the only fair comparison with *Fixed* as the setting variables are the one influencing the possible distributions. In the general case, better strategies could be defined for *Variable*. Time out is set to 30 minutes. The integer CP solver used for all the experiments of this paper is our internal solver. The continuous part is solved by call to IBEX [4], analogously to CHOCO-solver [43].

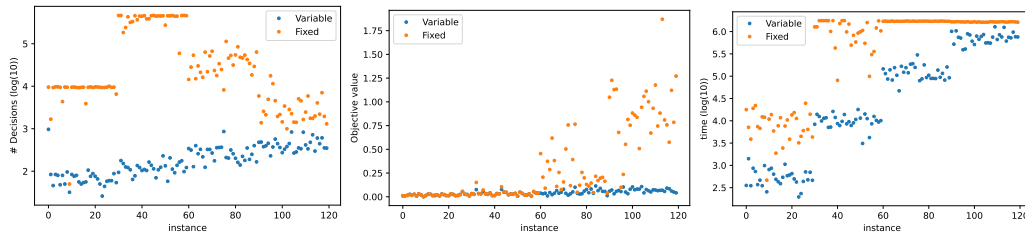
## 5.3 Results

First, consider the results from Table 1. As we can see, finding a solution for the *fixed* model is not too hard, but as the size increases, it is not able to find the optimal one, or to prove it. In contrast, for the *variable* model, the optimal solution is always found and proved. The main reason is that invalid distributions are removed by the latency constraint before any decision leading to them is taken.

■ **Table 1** For each instance set, the values are #SAT (#OPT). The last row is the mean time in seconds for proving optimality.

# Nodes	2	3	4	5
<b>Fixed</b> (baseline)	30 ( <b>30</b> )	30 (22)	30 (0)	30 (0)
<b>Variable</b>	30 ( <b>30</b> )	30 ( <b>30</b> )	30 ( <b>30</b> )	30 ( <b>30</b> )
<b>Variable</b> avg time	0.57 s	10.2 s	125.4 s	752.3 s

Moreover, consider the plots from Figure 2. Instances are sorted by size. On the middle, the comparison on the objective value is made. As we can see, the objective value of the *fixed* method is optimal for small instances, even if no proof can be reached. Then, for larger instances, it is often quite far from the optimal (objective  $\in [0, 3]$ ). In addition, on the left and right plots, it is seen that most of the time, the *fixed* method reaches the timeout. Note that the time increases exponentially with the number of nodes for both methods, but not the number of decisions for the *variable* method.



■ **Figure 2** Instances are ordered w.r.t. size. (left) Number of decision (log base 10). (middle) Best objective value comparison. (right) Running time (log base 10).

In conclusion, this experiment shows that the *variable* method proposed in this paper is better both in terms of quality of solution and in terms of time. In practice, solving the industrial instances of size 3 takes 17.75 seconds on average for the optimal solution.

## 6 Application: Chemical Deliveries

The following problem is part of a pipeline of chemicals processing and is composed of two assignment problems. First, chemicals are received by the factory every day. Once received, they must be stored into containers. These containers are restricted to some type of products. Each container already contains a known amount of products and has a maximum storage capacity. In addition, for some chemicals the total amount delivered is larger than the remaining storage capacity of the containers, given their currently stored quantities. In practice, this is not an issue as the chemicals are also processed, hence emptying the containers. The second problem concerns product assignment. Indeed, the chemicals are used to manufacture bio-sourced bases for perfumes. Those are later used by home-perfume makers to create reeds, candles, etc. A dozen of teams are spread over 4 buildings. Each building/team is specialized in a set of types of product, yet there are overlaps. Every day, in addition to the incoming deliveries, the factory must produce a given amount of several products. Selecting which team should work on which product is part of the problem to solve. The probability of emptying a container depends on the products associated to the teams in the building. A solution is an assignment of products to teams and deliveries to containers such that the stored quantity in each container minus the quantity that will be used does not exceed the maximum capacity with a high confidence.

More formally, this is an assignment problem. Each product  $p \in P$  must be produced by a team  $t \in T$ . Each team  $t \in T$  has a work capacity  $W_t$ , and work in a building  $b_t \in B$ . Each product  $w$  requires an given amount of work  $w_p$ , and is compatible to a subset of the teams  $T_p \in T$ . Each delivery  $j \in D$  stores a quantity  $q_j$  in a container  $c_i \in C$ . Each container  $c_i$  has a maximum capacity  $C_i$ . Each container  $c_i$  will be emptied in parallel of a quantity  $y_i$  unknown in advance. The exact quantity  $y_i$  is unknown, but its distribution is influenced by the products processed in its building.

## 6.1 Data

The dataset contains 40 instances, with  $|P| \in [15, 25]$ ,  $|C| \in [4, 10]$ ,  $T = 12, B = 4, |D| \in [20, 50]$ . Maximum running time is set to 30 minutes. The dataset is based on past data for the generation of probabilistic distributions. Figure 3 (left) shows an example of a set of distribution for one container. As we can see, the more the building has to make products, the higher the chance to consume.

## 6.2 Model

For each product  $p$ , variables  $p_{p,t} \forall t \in T$  indicates if the product is made by the team  $t$ . For each delivery  $d$ , variable  $d_{d,c} \forall c \in C$  indicates if the delivery is stored in container  $c$ . For each container  $c$ ,  $y_c$  is the emptying random variable. For each building  $b$ ,  $q_b$  is the quantity of product produced in the building. Let  $q = (q_1, \dots, q_{|B|})$  be a vector of all  $q_b$  variables. Let  $x_c \forall c \in C$  be the additional capacity required by container  $c$  such that it is no longer overflowing.

$$\arg \max_{x \in \mathbb{Z}, Y \in \mathbb{P}} \gamma \quad (26)$$

$$\text{Such that} \quad \sum_{t=1}^{|T|} p_{i,t} = 1, \quad \forall i \in P \quad (27)$$

$$\sum_{i=1}^{|P|} p_{i,t} w_i \leq W_t, \quad \forall t \in T \quad (28)$$

$$q_b = \sum_{i=1}^{|P|} \sum_{t \in \{t_i | b_{t_i} = b\}} p_{i,t}, \quad \forall b \in B \quad (29)$$

$$\sum_{i=1}^{|D|} d_{i,c} q_i \leq C_c + x_c, \quad \forall c \in C \quad (30)$$

$$\sum_{c=1}^{|C|} d_{i,c} = 1, \quad \forall i \in D \quad (31)$$

$$\text{Table}(q, y_c), \quad \forall c \in C \quad (32)$$

$$p_{x_c \leq y_c} \equiv x_c \leq y_c, \quad \forall c \in C \quad (33)$$

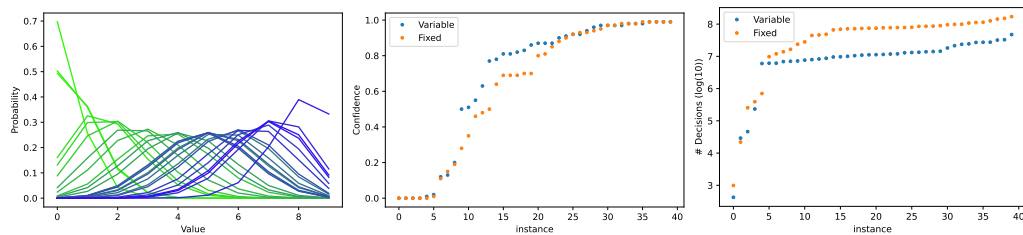
$$\Pr[(p_{x_1 \leq y_1}, \dots, p_{x_{|C|} \leq y_{|C|}}) = \mathbf{1}] \geq \gamma \quad (34)$$

## 6.3 Results

Figure 3 shows the results of the *fixed* and *variable* (this paper) methods. First, it is interesting to see that for some instances a confidence of 1 can be found. For these instances, no consumption was required to find a solution. In contrast, we can also see that for some instances, it is hard to find solutions with more than 2 percent confidence. Those instances are hard instances, where the delivered amount is too large. Then in the middle of the plot, it is shown that the confidence of solutions found by the *variable* method is higher in general than the *fixed* method. In addition, the *variable* method makes one order of magnitude less decisions to find better or equivalent solutions. When profiled, the reason why less decisions are made is the repetitive call to the continuous integrated solver. An actual hybrid CP solver would be drastically more efficient. Nevertheless, even with this drawback, in both experiments, the *variable* showed significant improvements in term both of quality of solution and time.

## 7 Conclusion and Future works

This paper proposed to extend the CP framework to distribution optimization. First, random variables have been extended to the case of distribution variables, then the CP constraints have been extended to deal with this extension. We proposed a generalization of integer



■ **Figure 3** Instances are ordered w.r.t. size. (left) Range of probability distributions for a random variable having 9 possible values. The gradient of color indicate the increase of  $q$ . (middle) Best confidence comparison. (right) Number of decisions (log base 10).

variables and new consistencies for probability distributions. In addition, generic filtering algorithms have been proposed, pruning invalid distributions. We defined an implementation based on a probability mass function decomposition and the minimal arithmetic to model most problems, together with the associated filtering algorithms. Finally, as shown in the experimental section, we used the proposed framework to solve two optimization problems where the distributions of probability are not fixed at the beginning of the problem.

The main future direction is to no longer restrict the search to a finite set of distributions using a table as done in our experiments, but to be able to search directly into the distribution space, which is closer to known methods in machine learning. Doing this would bring CP as one of the main frameworks to do constrained distribution learning, and will require hybridization of CP solvers. Other future directions include the design of specialized global constraints filtering, the generalization of the reuse of existing filtering algorithms [18], and the implementation of different types of encoding of the probability distribution variables. Another important direction is the encoding of dependent random variables. In this paper, we purposely introduced the event space of probability space to encode the dependency between random variables. In future works, constraints on the event spaces will lead to efficient dependency implementation. Finally, this paper is restricted to 2-stage policy, it will be important in the future to extend it to multi-stages.

---

## References

- 1 Eric Baum and Frank Wilczek. Supervised learning of probability distributions by neural networks. In *Neural information processing systems*, 1987.
- 2 N Bharanidharan and Harikumar Rajaguru. Improved chicken swarm optimization to classify dementia mri images using a novel controlled randomness optimization algorithm. *International Journal of Imaging Systems and Technology*, 30(3):605–620, 2020.
- 3 Giuseppe Calafiore and Marco C Campi. Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming*, 102(1):25–46, 2005.
- 4 Gilles Chabert et al. Ibex, an interval-based explorer, 2007.
- 5 Gilles Chabert and Luc Jaulin. Contractor programming. *Artificial Intelligence*, 173(11):1079–1100, 2009.
- 6 Abraham Charnes and William W Cooper. Chance-constrained programming. *Management science*, 6(1):73–79, 1959.
- 7 Abraham Charnes and William W Cooper. Deterministic equivalents for optimizing and satisficing under chance constraints. *Operations research*, 11(1):18–39, 1963.
- 8 Abraham Charnes, William Wager Cooper, and MJL Kirby. Chance-constrained programming: an extension of statistical method. In *Optimizing methods in statistics*, pages 391–402. Elsevier, 1971.

- 9 Constantinos Daskalakis, Ilias Diakonikolas, and Rocco A Servedio. Learning poisson binomial distributions. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 709–728, 2012.
- 10 Constantinos Daskalakis and Gautam Kamath. Faster and sample near-optimal algorithms for proper learning mixtures of gaussians. In *Conference on Learning Theory*, pages 1183–1213. PMLR, 2014.
- 11 Augustin Delecluse, Pierre Schaus, and Pascal Van Hentenryck. Sequence variables for routing problems. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, 2022.
- 12 Mihai Dobrescu, Katerina Argyraki, and Sylvia Ratnasamy. Toward predictable performance in software {Packet-Processing} platforms. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 141–154, 2012.
- 13 Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle. Assessing soft-and hardware bottlenecks in pc-based packet forwarding systems. *ICN 2015*, 90, 2015.
- 14 H el ene Fargier and J er ome Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty: European Conference ECSQARU'93 Granada, Spain, November 8–10, 1993 Proceedings 2*, pages 97–104. Springer, 1993.
- 15 Jakob Feldtkeller, David Knichel, Pascal Sasdrich, Amir Moradi, and Tim G uneysu. Randomness optimization for gadget compositions in higher-order masking. *Cryptology ePrint Archive*, 2022.
- 16 Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.
- 17 Brahim Hnich, Roberto Rossi, S Armagan Tarim, and Steven Prestwich. Synthesizing filtering algorithms for global chance-constraints. In *Principles and Practice of Constraint Programming-CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September 20-24, 2009 Proceedings 15*, pages 439–453. Springer, 2009.
- 18 Brahim Hnich, Roberto Rossi, S Armagan Tarim, and Steven Prestwich. Filtering algorithms for global chance constraints. *Artificial Intelligence*, 189:69–94, 2012.
- 19 JN Hooker. Stochastic decision diagrams. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 138–154. Springer, 2022.
- 20 Mark H Houck. A chance constrained optimization model for reservoir design and operation. *Water Resources Research*, 15(5):1011–1016, 1979.
- 21 Bahareh Kargar, Mir Saman Pishvaei, Hamed Jahani, and Jiuh-Biing Sheu. Organ transportation and allocation problem under medical uncertainty: A real case study of liver transplantation. *Transportation Research Part E: Logistics and Transportation Review*, 134:101841, 2020.
- 22 Michael Kearns, Yishay Mansour, Dana Ron, Ronitt Rubinfeld, Robert E Schapire, and Linda Sellie. On the learnability of discrete distributions. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 273–282, 1994.
- 23 Lin Kemeng, Wang Xiaoyan, Xia Weijie, Zhang Jiaming, et al. Optimization of the randomness in einstein which based on monte carlo algorithms. In *2019 Chinese Control And Decision Conference (CCDC)*, pages 6305–6309. IEEE, 2019.
- 24 Stefan Kern, Sibylle D M uller, Nikolaus Hansen, Dirk B uche, Jiri Ocenasek, and Petros Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms—a comparative review. *Natural Computing*, 3:77–112, 2004.
- 25 Minh Thanh Khong, Christophe Lecoutre, Pierre Schaus, and Yves Deville. Soft-regular with a prefix-size violation measure. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 333–343. Springer, 2018.
- 26 Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.



- 27 Anna LD Latour, Behrouz Babaki, Daniël Fokkinga, Marie Anastacio, Holger H Hoos, and Siegfried Nijssen. Exact stochastic constraint optimisation with applications in network analysis. *Artificial Intelligence*, 304:103650, 2022.
- 28 Pu Li, Harvey Arellano-Garcia, and Günter Wozny. Chance constrained programming approach to process optimization under uncertainty. *Computers & chemical engineering*, 32(1-2):25–45, 2008.
- 29 Xiaoxia Lin, Stacy L Janak, and Christodoulos A Floudas. A new robust optimization approach for scheduling under uncertainty:: I. bounded uncertainty. *Computers & chemical engineering*, 28(6-7):1069–1085, 2004.
- 30 Alexandre Mercier-Aubin, Ludwig Dumetz, Jonathan Gaudreault, and Claude-Guy Quimper. The confidence constraint: A step towards stochastic cp solvers. In *International Conference on Principles and Practice of Constraint Programming*, pages 759–773. Springer, 2020.
- 31 Arkadi Nemirovski and Alexander Shapiro. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):969–996, 2007.
- 32 François Pachet, Pierre Roy, Alexandre Papadopoulos, and Jason Sakellariou. Generating 1/f noise sequences as constraint satisfaction: The voss constraint. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- 33 Bernardo K Pagnoncelli, Shabbir Ahmed, and Alexander Shapiro. Sample average approximation method for chance constrained programming: theory and applications. *Journal of optimization theory and applications*, 142(2):399–416, 2009.
- 34 M Arenas Parra, A Bilbao Terol, B Pérez Gladish, and MV Rodriguez Uribe. Solving a multiobjective possibilistic problem through compromise programming. *European Journal of Operational Research*, 164(3):748–759, 2005.
- 35 Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.
- 36 Guillaume Perez, Steve Malalel, Gael Glorian, Victor Jung, Alexandre Papadopoulos, Marie Pelleau, Wijnand Suijlen, Jean-Charles Régin, and Arnaud Lallouet. Generalized confidence constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.
- 37 Guillaume Perez, Brendan Rappazzo, and Carla Gomes. Extending the capacity of 1/f noise generation. In *International Conference on Principles and Practice of Constraint Programming*, pages 601–610. Springer, 2018.
- 38 Guillaume Perez and Jean-Charles Régin. MDDs are efficient modeling tools: An application to dispersion constraints. In *Integration of AI and OR Techniques in Constraint Programming*, 2017.
- 39 Guillaume Perez and Jean-Charles Régin. Mdds: Sampling and probability constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 226–242. Springer, 2017.
- 40 Gilles Pesant. Achieving domain consistency and counting solutions for dispersion constraints. *INFORMS Journal on Computing*, 27(4):690–703, 2015. doi:10.1287/ijoc.2015.0654.
- 41 Warren B Powell. A unified framework for stochastic optimization. *European Journal of Operational Research*, 275(3):795–821, 2019.
- 42 Steven D Prestwich, Roberto Rossi, and S Armagan Tarim. Randomness as a constraint. In *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31–September 4, 2015, Proceedings 21*, pages 351–366. Springer, 2015.
- 43 Charles Prud’homme, Jean-Guillaume Fages, and Xavier Lorca. Choco solver documentation. *TASC, INRIA Rennes, LINA CNRS UMR*, 6241:13–42, 2016.
- 44 Jean-Charles Régin. Arc consistency for global cardinality constraints with costs. In *International Conference on Principles and Practice of Constraint Programming*, pages 390–404. Springer, 1999.
- 45 Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.


- 46 Roberto Rossi, Brahim Hnich, S Armagan Tarim, and Steven Prestwich. Confidence-based reasoning in stochastic constraint programming. *Artificial Intelligence*, 228:129–152, 2015.
- 47 Christian Schulte and Guido Tack. View-based propagator derivation. *Constraints*, 18(1):75–107, 2013.
- 48 Kalika Suksomboon, Nobutaka Matsumoto, Shuichi Okamoto, Michiaki Hayashi, and Yusheng Ji. Configuring a software router by the erlang- $k$ -based packet latency prediction. *IEEE Journal on Selected Areas in Communications*, 36(3):422–437, 2018.
- 49 S Armagan Tarim, Suresh Manandhar, and Toby Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11:53–80, 2006.
- 50 Joseph A Tatman and Ross D Shachter. Dynamic programming and influence diagrams. *IEEE transactions on systems, man, and cybernetics*, 20(2):365–379, 1990.
- 51 Pascal Van Hentenryck and Laurent Michel. Domain views for constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 705–720. Springer, 2014.
- 52 Willem-Jan Van Hoes, Gilles Pesant, and Louis-Martin Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, 12(4):347–373, 2006.
- 53 Toby Walsh. Stochastic constraint programming. In *ECAI*, volume 2, pages 111–115, 2002.
- 54 Weijun Xie and Shabbir Ahmed. On deterministic reformulations of distributionally robust joint chance constrained optimization problems. *SIAM Journal on Optimization*, 28(2):1151–1182, 2018.
- 55 Hui Zhang and Pu Li. Chance constrained programming for optimal power flow under uncertainty. *IEEE Transactions on Power Systems*, 26(4):2417–2424, 2011.



# The p-Dispersion Problem with Distance Constraints

Nikolaos Ploskas ✉ 

University of Western Macedonia, Kozani, Greece

Kostas Stergiou ✉ 

University of Western Macedonia, Kozani, Greece

Dimosthenis C. Tsouros ✉ 

KU Leuven, Belgium

---

## Abstract

In the (maxmin) p-dispersion problem we seek to locate a set of facilities in an area so that the minimum distance between any pair of facilities is maximized. We study a variant of this problem where there exist constraints specifying the minimum allowed distances between the facilities. This type of problem, which we call PDDP, has not received much attention within the literature on location and dispersion problems, despite its relevance to real scenarios. We propose both ILP and CP methods to solve the PDDP. Regarding ILP, we give two formulations derived from a classic and a state-of-the-art model for p-dispersion, respectively. Regarding CP, we first give a generic model that can be implemented within any standard CP solver, and we then propose a specialized heuristic Branch&Bound method. Experiments demonstrate that the ILP formulations are more efficient than the CP model, as the latter is unable to prove optimality in reasonable time, except for small problems, and is usually slower in finding solutions of the same quality than the ILP models. However, although the ILP approach displays good performance on small to medium size problems, it cannot efficiently handle larger ones. The heuristic CP-based method can be very efficient on larger problems and is able to quickly discover solutions to problems that are very hard for an ILP solver.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming

**Keywords and phrases** Facility location, distance constraints, optimization

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.30

## 1 Introduction

Maximum diversity problems arise in many practical settings from facility location to telecommunications and social network analysis [28]. Arguably, the most famous such problem is the (*maxmin*) *p*-dispersion problem (PDP) [29]. In the PDP we are given a set of candidate locations  $P = \{1, 2, \dots, n\}$  for  $p$  facilities and an  $n \times n$  matrix  $(D[i, j])$ ,  $i, j \in P$  with distances between candidate locations  $i$  and  $j$ . The goal is to select  $p$  items from  $P$  to locate the facilities such that the minimum distance between any pair of facilities is maximized.

In practice, the PDP occurs whenever a close proximity of facilities is dangerous or for other reasons undesirable. A standard application is concerned with the location of power plants, where we wish to minimize the risk of losing multiple plants in the event of an accident or an enemy attack. To achieve this, locations for the plants are desired so that interplant distances are as large as possible. Similar applications arise in the military sector, as it is common to scatter military installations in order to make it difficult for the enemy to disarm all of them. In a more peaceful context, we may wish to disperse branches of the same franchise, so that mutual competition between similar shops is minimized, or public facilities which have overlapping areas of service, e.g., schools, hospitals, electoral districts,



© Nikolaos Ploskas, Kostas Stergiou, and Dimosthenis C. Tsouros;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 30; pp. 30:1–30:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

etc. [25, 16, 28]. In telecommunications, we may wish to disperse radio transceivers to service cellular phones so as to minimize interference. Another, more recent, area of application for the PDP, is if distances are not interpreted physically but as a measure of the diversity between members of a group [33].

Another dispersion problem that has been deeply studied is the maxsum p-dispersion problem, called p-defence problem in [29], where we seek to locate p facilities so that the sum of distances between the facilities is maximized [25]. Although the maxsum variant also has many applications, it is recognized that the PDP is better suited to model situations where the close proximity of facilities must be definitely avoided (e.g. for safety reasons). This is because maximizing the sum of distances does not guarantee that no two facilities will be placed close to each other [25].

In this paper we are concerned with a variant of the PDP where hard constraints specifying minimum allowed distances exist between facilities. We call this problem *p-dispersion with distance constraints* (PDDP). Although this problem was put forward by Moon and Chaudhry who first systematically studied location problems with distance constraints and coined the term p-dispersion [29], it has been rather ignored since, despite its relevance to real applications. Distance constraints in dispersion problems can stem from operational needs and regulations, such as clearance distances for safe chemical storage [1], separation distances between packages containing radioactive materials [40], or between portable fire extinguishers in an area [41]. Motivated by such applications, a recent study by Dai et al. considers p-dispersion with distance constraints in the context of circle placement in non-convex polygons [10].

To further motivate our study of the PDDP, consider a scenario where p identical power plants need to be located in an area. Assume that due to safety reasons, any two plants must be more than  $x$  km away from one another. If this problem is modeled and solved as a PDP then either of the following two results will hold: 1) The optimal solution places the two closest facilities  $y \leq x$  km apart. Then the original problem is infeasible, as the safety requirements cannot be satisfied for all pairs of facilities, 2) The optimal solution places the two closest facilities  $y > x$  km apart. Then, as  $y$  is the minimum distance between any two plants, all the safety requirements are satisfied and the original problem has been solved. But what if all power plants to be located are not identical? What if there are differences in the plants' sizes, the volume of power generated, the waste produced, etc.?

In such cases, the safety requirements regarding the minimum allowed distances between plants may not be the same for all pairs of plants. For instance, the allowed distance between smaller and less dangerous plants will probably be smaller than between larger ones. Hence, the PDP does no longer suffice to model the problem. This is because an optimal solution that places the closest plants  $y$  km apart does not guarantee that the safety distances will be satisfied for all pairs of plants. The case of non-identical (heterogeneous) facilities has not received much attention in the p-dispersion literature, as the predominant explicit or implicit assumption is that the facilities to be located are indistinguishable (homogeneous). But in practice, not all power plants will be identical, and the same holds for the branches of a franchise, for public facilities, and almost any type of facilities that we want to disperse.

We start our study of the PDDP by giving two ILP formulations. The first one is based on the classic formulation for p-dispersion by Kuby [25], while the second is based on a state-of-the-art model proposed by Sayah and Irnich [34]. Both these formulations are extended to deal with heterogeneous facilities and to include distance constraints.

Then, we describe a generic CP model for the PDDP that can be implemented within any standard CP solver. For the purposes of this study, we have implemented this model in OR-Tools and Choco. Experimental results demonstrate that the ILP formulations,

implemented in Gurobi, are more efficient than the CP solvers, as the latter find it very hard to prove optimality, even for small problems, and are usually slower in locating solutions of the same quality as the ILP solver.

We further explore the applicability of CP technology by introducing a specialized heuristic method based on CP. Specifically, using a simpler model of the problem, with fewer variables and constraints, we have devised a specialized Branch&Bound mechanism, which has been implemented in a custom CP solver. This method tries to prune the search tree early by estimating the best cost that can be achieved if the sub-tree rooted at the currently visited node is explored. If the estimated cost does not improve the cost of the best solution found so far then the current branch is abandoned. The estimation is carried out through a simple greedy assignment of the remaining variables. This reasoning achieves significant search tree pruning, albeit by sacrificing completeness.

In our experimental analysis we first compare the ILP formulations to the CP approaches on randomly generated problems with 5-30 facilities and at most 80 potential location points. Results show that the CP model implemented in the standard CP solvers OR-Tools and Choco cannot compete with the ILP formulations implemented in Gurobi, as the latter solver is quite efficient in all but the largest class that contains 30 facilities and 80 location points. The heuristic CP approach very quickly finds solutions, often optimal ones, on all instances, including those of the hardest class. We then consider harder problems that are generated using the p-dispersion MDPLIB benchmark library as basis [28]. Results demonstrate that the ILP formulations are efficient on problems with 10 facilities and 100 potential locations, but fail to efficiently handle larger problems. On the other hand, the heuristic CP approach can trivially find solutions of good quality on smaller instances, while it can also handle larger instances that are very hard for the ILP solver, finding solutions of much better quality.

## 2 Related Work

The maxmin p-dispersion problem, which is  $\mathcal{NP}$ -hard on general networks for an arbitrary p [18], was originally mentioned by Shier, as far back as 1977 [36]. However, the term p-dispersion first appeared in the analysis of location problems with distance constraints by Moon and Chaudhry [29]. The first ILP solution was proposed by Kuby [25] while the first specialized algorithm was given by Erkut [15]. Kincaid proposed simulated annealing and tabu search methods [24], Ghosh proposed a multi-start heuristic [18] and Resende et al. applied the GRASP methodology to the maxmin problem [32]. More recently, Sayyady & Fathi [35] and Sayah & Irnich [34] proposed ILP approaches to the maxmin problem, which are able to solve large size problems, and as argued in the comprehensive review on OR methods for dispersion problems given in [28], they tend to make heuristic approaches obsolete, as they can handle problems of similar size.

Regarding distance constraints, Moon and Chaudhry were the first to systematically study location problems with distance constraints [29]. The p-dispersion problem with distance constraints was mentioned by them as a problem that can arise in real-life scenarios, but no approaches towards solving it were proposed. Recently, Dai et al. revisited this problem as part of a study on circle (i.e. facility) dispersion in non-convex polygons [10]. A heuristic method, inspired by the mechanics of the n-body problem in physics, was proposed for the plain p-dispersion problem in non-convex polygons, and this method was also adapted to the case where distance constraints exist between pairs of circles.

Distance constraints have also been considered in the context of other location problems. Some early works considered maximum distance constraints between the demand nodes and the facility locations [7, 8, 23, 38]. Tansel et al. studied the distance constrained *p-center*

problem for the case where the network is a tree [38]. Chaudhry et al. proposed heuristics for selecting a maximum-weight set of locations such that no two are closer than a given distance from each other [6]. Moon and Papayanopoulos considered the problem of locating two facilities so as to minimize the maximum of combined Euclidean distances to unweighted existing points when the facilities must be separated by at least a specified distance [30]. Comley studied the problem of locating a small number of heterogeneous semi-obnoxious facilities that interact with each other as well as with other existing facilities [9].

Berman and Huang studied the problem of locating homogeneous obnoxious facilities on a network so as to minimize the total demand covered, subject to the condition that no two facilities are allowed to be closer than a pre-specified distance [2]. Drezner et al. proposed the Weber obnoxious facility location problem where we seek to locate one facility so that the weighted sum of distances between the facility and demand points is minimized, with the additional requirement that the facility location is at least a given distance from demand points because it is obnoxious to them [13]. Drezner et al. considered a continuous multiple obnoxious facility location problem where a given number of facilities must be located in a convex polygon with the objective of maximizing the minimum distance between facilities and a given set of communities subject to distance constraints between facilities [14]. Welch and Salhi studied the location of obnoxious facilities with interactions between them [39]. Location problems with distance constraints that restrict the placement of facilities near certain demand points have also been studied, e.g. [31, 4, 27].

There are very few CP-related methods for facility location problems [17, 5, 37] and none of them concerns p-dispersion problems, with or without distance constraints. Regarding our heuristic CP-based method, there are works that follow a similar approach, i.e. sacrificing the completeness of a CP solver to solve optimization problems faster [22, 26, 19]. However, these works typically do this through a more local reasoning, e.g. by adding extra constraints.

Finally, the p-median problem with distance constraints, originally put forward in [29], is being studied in a paper that is currently under review (details are suppressed to preserve anonymity). In such a problem, there exist both facilities and clients that are serviced by the facilities. The goal is to locate  $p$  facilities so that the sum of the distances between the clients and their closest facility is minimized. As here, ILP and CP models for this problem are proposed and compared. Results demonstrate that the ILP approach is by far the most efficient on problems with homogeneous facilities, but it is outperformed by a heuristic CP approach on some classes of problems with heterogeneous facilities.

### 3 Background

In a p-dispersion with distance constraints problem (PDDP),  $p$  facilities in a set of facilities  $F$  are to be placed in an area. We assume that the set  $P$  of potential location points for the facilities is known. We also assume that the distance between each pair of potential locations  $(i, j)$  is given in a matrix  $D$ . Between each pair of facilities  $f_i$  and  $f_j$  there is a distance constraint  $dis(f_i, f_j) > d_{ij}$  specifying that the distance  $dis(f_i, f_j)$  between the points where the facilities  $f_i$  and  $f_j$  are located must be greater than  $d_{ij}$ , where  $d_{ij}$  is a constant. To summarize, in a PDDP we have:

- $P$ : the set of candidate facility locations.
- $F$ : the set of facilities to be located.
- $p$ : the number of facilities to be located.
- $D[i, j]$ : the distance between any two candidate facility locations.
- $d_{ij}$ : the lower bound in the allowed distance between each pair of facilities  $(i, j)$ , where  $i, j \in F$ .

The distance between two points  $i$  and  $j$  can be given by the straight-line (Euclidean) distance, e.g. for the location of hazardous facilities, or by the shortest path in a street network, e.g. for the location of franchises, or by any other suitable metric. The methods we propose do not depend on any particular distance measure because, as is common in the literature, we assume that the pairwise distances between all possible location points are given in a 2-d distance matrix  $D$ . However, if necessary, instead of precomputing the distances and storing them in a distance matrix, they could be computed “on the fly”, under the condition that this operation takes constant time. This holds for Euclidean or Manhattan distances given the coordinates of the points, but it does not hold for the shortest path in a network.

A common assumption in the literature on location problems with distance constraints is that the lower distance bound  $d_{ij}$  is fixed to a specific value for all the constraints between facilities. This is a reasonable assumption in the case where the facilities are homogeneous, and therefore in essence indistinguishable, but it is not always realistic, especially when the facilities have different properties, as for example in [1, 40]. In case the facilities are heterogeneous, the distance bound may vary from constraint to constraint.

The goal in a PDDP is to locate each facility to a node so that the minimum distance between any two facilities is maximized subject to the satisfaction of all the distance constraints.

## 4 ILP models

We first give an ILP model for the PDDP, based on the classic formulation of Kuby for p-dispersion [25] and then we give a model based on the state-of-the-art model of Sayah & Irnich [34]. Both formulations are extended to deal with heterogeneous facilities and to include distance constraints between facilities.

### 4.1 Kuby based model

We make use of the following additional notation:

- $C = \{(i, j, f_1, f_2) \mid i, j \in P, f_1, f_2 \in F, D[i, j] \leq d_{f_1 f_2}\}, \forall i \in P, \forall j \in P$ , and for each pair of facilities  $(f_1, f_2)$ : the set of quadruples  $(i, j, f_1, f_2)$  s.t. facilities  $f_1$  and  $f_2$  cannot be placed in facility sites  $i$  and  $j$ , respectively, because  $i$  and  $j$  are not in a safe distance between each other with respect to the allowed distance between  $f_1$  and  $f_2$ .
- $x_{ij} = 1$  if a facility  $j \in F$  is located at a facility site  $i \in P$  and 0 otherwise.
- $y_i = 1$  if any facility is located at a facility site  $i \in P$  and 0 otherwise.
- $b$  is the minimum distance between the facilities that we aim to maximize.

For any  $i \in P$ , variable  $y_i$  shows whether or not facility site  $i$  will host any facility. These are the variables that are present in Kuby’s formulation for p-dispersion. Given that facilities are considered identical in the p-dispersion literature, in Kuby’s model we only need to know whether a site will host a facility or not. But in the case of the PDDP, where facilities can be different and distance constraints exist between them, we also need to know which particular facility will be hosted by a site. Hence, we introduce  $|P| \times |F|$  variables, i.e. one variable  $x_{ij}, \forall (i, j), i \in P, j \in F$ , in order to know whether or not a specific facility  $j \in F$  is located at a facility site  $i \in P$ .



The extension of Kuby's model to capture the PDDP can be expressed as:

$$\max \quad b \quad (1)$$

$$\text{s.t.} \quad \sum_{i \in P} y_i = p \quad (2)$$

$$b \leq M(2 - y_i - y_j) + D[i, j] \quad \forall i, j \in P, j > i \quad (3)$$

$$\sum_{j \in F} x_{ij} = y_i \quad \forall i \in P \quad (4)$$

$$\sum_{j \in F} x_{ij} \leq 1 \quad \forall i \in P \quad (5)$$

$$\sum_{i \in P} x_{ij} = 1 \quad \forall j \in F \quad (6)$$

$$x_{if_1} + x_{jf_2} \leq 1 \quad \forall (i, j, f_1, f_2) \in C \quad (7)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in P, \forall j \in F \quad (8)$$

$$y_i \in \{0, 1\} \quad \forall i \in P \quad (9)$$

$$b \geq 0 \quad (10)$$

The objective function 1 aims at maximizing the shortest distance  $b$  between located facilities. Constraint 2 specifies that  $p$  facilities are to be located. Constraint 3 guarantees that  $b \leq D[i, j]$  whenever both locations  $i$  and  $j$  are chosen via  $y_i = y_j = 1$  for the location of facilities, where  $M$  represents a Big  $M$  constant. Variables  $b$  and  $y_i, i \in P$ , the objective function 1 and Constraints 2, 3, 9 and 10 form the original formulation of Kuby for the p-dispersion problem.

As we explained, in the case of the PDDP, we add variables  $x_{ij}, \forall (i, j), i \in P, j \in F$ , in order to know whether or not a specific facility  $j \in F$  is located at a facility site  $i \in P$ . These variables are linked to the  $y_i$  variables via Constraint 4, which specifies that if any facility  $j$  is located at a facility site  $i$ , then variable  $y_i$  equals 1 and 0 otherwise. To ensure that no two variables  $x_{ij}$  and  $x_{ij'}$  are set to 1 (i.e. each facility site must host at most one facility), we add Constraint 5. To ensure that no two variables  $x_{ij}$  and  $x_{i'j}$  are set to 1 (i.e. each facility must be hosted at exactly one facility site), we add Constraint 6. Finally, Constraint 7 models the distance constraints between facilities. It ensures that each facility is at a safe distance from all other facilities by not allowing two facilities  $f_1$  and  $f_2$  to be established at sites that are at a distance closer than the allowed distance between  $f_1$  and  $f_2$ . These pairwise constraints are a special case of clique constraints and are an efficient option to model distance constraints in ILP, as demonstrated in [2].

The original Kuby model for the p-dispersion problem has  $|P|+1$  variables and  $\sum_{i=1}^{|P|-1} i+1$  constraints, while our extended Kuby based model for the PDDP has  $|P| \times |F| + |P| + 1$  variables and  $2 \times |P| + |F| + \sum_{i=1}^{|P|-1} i + 1$  constraints, without considering Constraint 7 which can give  $(|P| \times |F|)^2$  constraints.

## 4.2 Sayah and Irnich based model

We now present a model for the PDDP based on the PDP model proposed by Sayah and Irnich, which utilizes the fact that the optimal distance is equal to at least one of the entries of the distance matrix [34]. Let us introduce some additional notation for this model:

- $E = \{(i, j) \in P \times P : i < j\}$ : the set of edges between any two candidate facility locations.
- $E(l) = \{(i, j) \in E : D[i, j] < l\}$ : a subset of edges given any distance  $l$ .
- $L^0 < L^1 < \dots < L^{k_{max}}$ : the different nonzero values in  $D[i, j]$ . The associate index sets are  $K = \{1, 2, \dots, k_{max}\}$  and  $K_0 = \{0\} \cup K$ . By definition,  $\emptyset = E(L^0) \subsetneq E(L^1) \subsetneq \dots \subsetneq E(L^{k_{max}}) \subsetneq E$  holds.
- $z_k = 1$  if the location decisions satisfy a minimum distance of at least  $L^k$ ,  $k \in K$  and 0 otherwise.

Similar to the Kuby model, variable  $y_i$ , for any  $i \in P$ , shows whether or not facility site  $i$  will host any facility. In addition, variable  $z_k$ , for any  $k \in K$ , indicates whether the location decisions satisfy a minimum distance of at least  $L^k$ . These are the variables that are present in the formulation of Sayah and Irnich for p-dispersion. As we explained in Kuby's model, we also need to introduce  $|P| \times |F|$  variables, i.e. one variable  $x_{ij}, \forall (i, j), i \in P, j \in F$ , in order to know whether or not a specific facility  $j \in F$  is located at a facility site  $i \in P$ .

The model for the PDDP using Sayah and Irnich's model as basis can be expressed as:

$$\max \quad D^0 + \sum_{k \in K} (L^k - L^{k-1}) z_k \quad (11)$$

$$\text{s.t.} \quad \sum_{i \in P} y_i = p \quad (12)$$

$$z_k \leq z_{k-1} \quad \forall k \in K, k > 1 \quad (13)$$

$$y_i + y_j + z_k \leq 2 \quad \forall k \in K, (i, j) \in E(L^k) \setminus E(L^{k-1}) \quad (14)$$

$$\sum_{j \in F} x_{ij} = y_i \quad \forall i \in P \quad (15)$$

$$\sum_{j \in F} x_{ij} \leq 1 \quad \forall i \in P \quad (16)$$

$$\sum_{i \in P} x_{ij} = 1 \quad \forall j \in F \quad (17)$$

$$x_{if_1} + x_{jf_2} \leq 1 \quad \forall (i, j, f_1, f_2) \in C \quad (18)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in P, \forall j \in F \quad (19)$$

$$y_i \in \{0, 1\} \quad \forall i \in P \quad (20)$$

$$z_k \in \{0, 1\} \quad \forall k \in K \quad (21)$$

The objective function 11 aims at maximizing the shortest distance between located facilities. Constraints 12, 15, 16, 17, 18, 19 and 20 are also present in the Kuby based formulation, while Constraints 13, 14 and 21 replace Constraints 3 and 10. Constraint 13 models the consistency between the  $z_k$  variables in the sense that the  $z_k$  variables are non-increasing in  $k$ , while Constraint 14 ensures that no pair  $(i, j)$  of locations with distance  $D[i, j] < L^k$  is chosen simultaneously. The consistency Constraint 13 specifies that any feasible solution fulfills that there exists a unique  $k \in K_0$  with  $z_1 = z_2 = \dots = z_k = 1$  and  $z_{k+1} = z_{k+2} = \dots = z_{k_{max}} = 0$ . Variables  $y_i, i \in P$ , and  $z_k, k \in K$ , the objective function 11 and Constraints 12, 13, 14, 20 and 21 form the original formulation of Sayah and Irnich for the p-dispersion problem.

The addition of variables  $x_{ij}, \forall (i, j), i \in P, j \in F$ , in the revised Sayah and Irnich model for the PDPP yields Constraints 15, 16, 17, and 19, as already explained in Kuby's model. In addition, Constraint 18 models the distance constraints between facilities.

The original model of Sayah and Irnich for the p-dispersion problem has  $|P| + k_{max} - 1$  variables and  $k_{max} + (k_{max} - 1) \times \sum_{i=1}^{|P|-1} i - 1$  constraints, while our extended Sayah and Irnich based model for the PDDP has  $|P| \times |F| + |P| + k_{max} - 1$  variables and  $2 \times |P| + |F| + k_{max} + (k_{max} - 1) \times \sum_{i=1}^{|P|-1} i - 1$  without considering Constraint 18 which can give  $(|P| \times |F|)^2$  constraints.

## 5 CP approaches to the PDDP

We first give a generic CP model of the PDDP and we then we describe the mechanics of a specialized heuristic CP solver. The PDDP is modeled as a Constraint Optimization Problem (COP)  $(X, Dom, C, O)$ , where  $X$  is the set of decision variables,  $Dom$  is the set

of finite domains,  $C$  is the set of hard constraints and  $O$  is the optimization function. The model is as follows:

1. For each facility  $i \in F$  there is a finite domain variable  $x_i$ . These  $p$  variables are the decision variables in the problem, meaning that  $|X| = |F| = p$ . The domain of each variable  $x_i \in X$ , denoted by  $Dom(x_i)$ , includes as values all the points where a facility can be located, i.e.  $\forall x_i \in X : Dom(x_i) = P$ .
2.  $Y$  is a set of auxiliary variables, s.t. for each pair of variables  $(x_i, x_j) \in X \times X \mid i < j$ , there is a variable  $y_{ij} \in Y$  and a constraint  $y_{ij} = D[x_i, x_j]$ . Hence, each  $y_{ij} \in Y$  models the distance between  $x_i$  and  $x_j$ . In CP solvers, this is implemented using the Element global constraint, i.e.  $y_{ij} = Element(D, [x_i, x_j])$ .
3. For each variable  $y_{ij}$ , there is a distance constraint  $y_{ij} > d_{ij}$ .
4. There is a variable  $z$ , s.t.  $z = \min(Y)$ .
5. The objective function is  $O = maximize(z)$ .

This model contains  $p + p \times (p - 1)/2 + 1$  variables, with  $p$  being decision variables and the rest auxiliary, and  $p \times (p - 1) + 1$  constraints.

We also considered the use of an AllDifferent constraint, to speed up propagation. However, the distance constraints already propagate the fact that facilities should be placed at different locations, as they all have bound greater than 0. Experiments with and without the AllDifferent constraint showed no noticeable difference in run times.

## 5.1 A heuristic CP-based method

We now propose a heuristic technique that tries to prune early the parts of the search tree that do not seem promising, i.e. it is unlikely that exploring them will improve the value of the optimization function. At each node of the search tree this method tries to estimate the best value of the optimization function that can be achieved if we explore the sub-tree rooted at that node. If this value is not better than the cost of the best solution found so far then the current branch is not further explored.

The proposed method can be embedded in a standard CP solver. However, this cannot be done at the modeling level by just specifying variables and posting constraints, because it requires writing specialized code within the solver and possibly a intervention in how the search process works. Naturally, the estimation of the bound at each node cannot always be precise (otherwise we would be able to trivially solve the PDDP), and therefore, a solver that employs this method will not be exact.

To demonstrate our heuristic method, we describe a simple CP solver, specialized for the PDDP, that applies it. This solver uses a simpler model of the problem, dropping the auxiliary variables and relevant constraints. Hence, we now have a model with only the  $p$  decision variables and  $p \times (p - 1)/2$  distance constraints. The optimization function is handled within the solver in the following way: Whenever a new solution is found, its cost is computed so as to determine if this cost is better than the current best cost. If so, then the best cost found so far is updated. If such an update occurs, it will be propagated to the decision variables, as described below.

The heuristic pruning technique works as follows: The cost of the first feasible solution found is used as the initial lower bound denoting the cost of the best solution found so far. Thereafter, at each node of the search tree, an upper bound for the best possible solution under the current assignment is computed. This upper bound gives an estimation of the best possible cost that can be achieved if the sub-tree rooted at the specific node is explored. If this is not higher than the current best cost then the current branch of the search tree is abandoned and the search moves on. Each time a solution with a higher cost than the current lower bound is found, the lower bound is updated.

The computation of the upper bound estimation at each node is performed in a greedy fashion. Specifically, assuming that  $x_i$  is the current variable,  $(x_1 \leftarrow v_1), \dots, (x_{i-1} \leftarrow v_{i-1})$  is the assignment to past variables and  $v_i$  is the value under consideration for  $x_i$ , we greedily compute the cost of the “best” assignment for the future variables  $x_{i+1}, \dots, x_p$ . That is, we visit these variables one by one, starting with  $x_{i+1}$ , and for each variable  $x_j$ ,  $i + 1 \leq j \leq p$ , and each value  $v_j \in \text{Dom}(x_j)$ , we find the minimum distance between  $v_j$  and any assignment (location) among variables (facilities)  $x_1, \dots, x_{j-1}$ . The value that maximizes this distance is then (temporarily) assigned to  $x_j$ . This is repeated until all variables have been assigned. We then compute the cost of this complete assignment, which gives the upper bound, but in fact, this may be an underestimation of the real cost. If the computed cost is equal to or lower than the current lower bound then the assignment of  $v_i$  to  $x_i$  is undone and the current branch of the search tree is abandoned, albeit risking to prune the branch leading to the optimal solution. This process is depicted by Algorithm 2.

Algorithm *PDDP\_CP\_Solver* (Algorithm 1) gives a high level description of the entire solving process.

---

■ **Algorithm 1** *PDDP\_CP\_Solver*( $X, \text{Dom}, C, O$ ).

---

```

if Propagate( $X, \text{Dom}, C$ ) = FALSE
  return NULL;
 $depth \leftarrow 1$ ;
 $best\_found \leftarrow 0$ ;
select an unassigned variable  $x_i$ ;
while  $depth \geq 1$ 
  if all values in  $\text{Dom}(x_i)$  have been tried
     $depth \leftarrow depth - 1$ ;
  else
    select a value  $a \in \text{Dom}(x_i)$  that has not been tried;
    if  $depth = n$ 
       $cur\_cost \leftarrow \text{Compute\_Solution\_Cost}(X, \text{Dom}, C)$ 
      if  $cur\_cost > best\_found$ 
         $best\_found \leftarrow cur\_cost$ ;
    else if Propagate( $X, \text{Dom}, C, x_i \leftarrow a$ ) = TRUE
      if  $best\_found \neq 0$  AND Bound( $X, \text{Dom}, C, x_i \leftarrow a, best\_found$ ) = TRUE
         $depth \leftarrow depth + 1$ ;
        select an unassigned variable  $x_i$ ;
if  $best\_found = 0$  return NULL;
return  $best\_found$ ;

```

---

Given a PDDP  $(X, \text{DOM}, C, O)$ , where  $O$  is the optimization function of the PDDP, the algorithm starts by propagating the hard constraints in  $C$ , as a typical CP solver does. Function *Propagate* enforces arc consistency on the distance constraints. If no failure (empty domain) is detected then the algorithm initializes the depth to 1 and the best cost found ( $best\_found$ ) to 0 and commences the search by selecting a variable using a variable ordering heuristic. While the depth of search is greater than 0, denoting that the search space has not been exhaustively searched, a branching decision is made, i.e. a value is selected and assigned to the currently selected variable. If all the variables have been assigned ( $depth = n$ ), meaning that a feasible solution has been found, the cost of this solution is computed and if this cost is higher than the best cost found so far then the latter is updated.

If not all variables have been assigned yet then Function *Propagate* is called to propagate the value assignment just made. If no failure occurs, the heuristic bounding mechanism described above is triggered by calling Function *Bound* (Algorithm 2), provided that at least one feasible solution has already been found. If this function succeeds, meaning that the

## 30:10 The p-Dispersion Problem with Distance Constraints

estimated cost is better than the best bound found so far then the algorithm moves forward by increasing the depth of search and selecting a new unassigned variable. On the other hand, if either propagation fails or the estimated bound is not better than the value of *best\_found* then the current branch is abandoned and a new value for the current variable is selected.

■ **Algorithm 2** *Bound*( $X, Dom, C, x_i \leftarrow v_i, best\_found$ ).

---

```
for each  $x_j, i + 1 \leq j \leq p$ 
   $val \leftarrow dis \leftarrow -1$ ;
  for each  $v_j \in Dom(x_j)$ 
     $temp \leftarrow$  shortest distance between  $v_j$  and any assigned variable  $x_1 \dots x_{j-1}$ ;
    if  $temp > dis$ 
       $dis \leftarrow temp$ ;
       $val \leftarrow v_j$ ;
   $x_j \leftarrow val$ ;
 $bound \leftarrow Compute\_Solution\_Cost(X, Dom, C)$ ;
if  $bound > best\_found$  return TRUE;
return FALSE;
```

---

If the value *best\_found* remains 0 upon termination then the algorithm has proved that the problem is infeasible and the solver returns NULL. Otherwise, the best cost found is returned. In the former case, the heuristic part of the algorithm (i.e. the bounding mechanism) will never be triggered, as no feasible solution will have been found. Hence, the search space will be systematically explored in a typical CP solver fashion until a backtrack to depth -1 occurs, proving that the problem is infeasible.

Function *Propagate* applies arc consistency on the distance constraints, taking into consideration the value of *best\_found*, i.e. the best cost found so far, to perform extra pruning, if possible. This is done in typical CP fashion for binary constraints, i.e. using a queue to insert and then process variables that have their domain filtered. Specifically, if a variable  $x_i$  is removed from the queue then for each variable  $x_j$  constrained with  $x_i$ , and each value  $v_j \in Dom(x_j)$ , we check if there exists a value  $v_i$  in  $Dom(x_i)$  s.t. the two values satisfy the distance constraint between  $x_j$  and  $x_i$  **and** the distance between the two values is greater than *best\_found*. In other words, for each possible location  $v_j$  of  $x_j$  we search for a location  $v_i$  for  $x_i$  s.t. by placing the two facilities at these locations, not only the relevant distance constraint is satisfied, but we can also improve the cost of the optimization function. If no such  $v_i$  exists then by placing  $x_j$  at  $v_j$  there is no way to locate  $x_i$  so that we can satisfy the distance constraint and improve the cost. Hence,  $v_j$  can be deleted from  $D(x_j)$ , i.e. it can be removed from consideration as a potential location point for  $x_j$ . If such a value deletion occurs then variable  $x_j$  is inserted in the queue to propagate the deletion.

Finally, note that the pruning that can be achieved by taking into consideration the current best cost can also be achieved by a standard CP solver that employs the model described further above. Such a solver will typically add a constraint  $z > best\_found$  once a new solution with better cost than the previously found solutions is located. The propagation of this constraint may result in the filtering of the  $y_{ij}$  variables' domains, which in turn will be carried over to the decision variables through the distance constraints  $y_{ij} = D[x_i, x_j]$ .

## 6 Experimental Results

We experimented with PDDP instances generated in two different ways. The first is a simple method that randomly generates a PDDP with a desired number of facilities and location points. The second uses the p-dispersion benchmark library MDPLIB 2.0 [28] as a basis to

generate PDDPs. Computations were performed on an Intel i7 CPU 8700 with 16 GB of main memory, a clock of 3.2 GHz, an L1 cache of 348 KB, an L2 cache of 2 MB, and an L3 cache of 12 MB, running under CentOS 8.4.

The ILP models were solved using Gurobi 9.0.3 [21]. The exact CP model was written in the CPMpy modeling tool [20] and compiled into OR-Tools [12]. An implementation in Choco [11] was also tried. The heuristic CP method was implemented in a custom solver programmed in C. This solver essentially implements the *CP\_Solve* algorithm (Algorithm 1) described above. Choco and the heuristic solver use the dom/wdeg heuristic for variable ordering [3] and lexicographic value ordering, while OR-Tools uses its default options. A time out of 3,600 seconds was set for each instance. We used only one thread on all solvers, to get a fair comparison.

The ILP models are stored in compressed sparse column format since the constraint matrix can sometimes be too large to be stored as a full array in memory. For example, the largest instance considered in the computational study has a constraint matrix of 5,425,061 rows, 105,038 columns and 11,212,542 nonzeros. Its compressed size is only 12MB, while its size as a full matrix is 530GB.

## 6.1 Random problems

For an initial evaluation of the proposed approaches to the PDDP, we ran experiments on problems where we try to locate  $p \in \{5, 10, 20, 30\}$  facilities in a  $10 \times 10$  grid, with  $|P| \in \{30, 80\}$  potential location points selected randomly among the 100 total points. The distances between the points are computed using the Manhattan distance metric. For each distance constraint  $dis(f_i, f_j) > d_{ij}$  between facilities  $f_i$  and  $f_j$ ,  $d_{ij}$  was set to a random integer in the interval  $[1, max]/2$ , where  $max$  is the maximum Manhattan distance between any two potential location points. Ten instances were generated and solved for each combination of parameter values.

Table 1 compares the following: Our two ILP formulations implemented in Gurobi, with  $Gurobi_k$  denoting our first model, based on that of Kuby, and  $Gurobi_s$  denoting our second model, based on that of Sayah & Irnich, and the CP solvers OR-tools and Choco. For each class, in column  $\sum_{cpu}$  we give the total cpu time taken by the corresponding solver over all 10 instances, and in brackets we give the number of instances on which the solver timed out. If the solver timed out on at least one instance then  $\sum_{cpu}$  gives a lower bound on the actual run time. Column  $cpu_o$  gives the mean time taken by the solver to locate the optimal solution. A zero means that less than 0.1 seconds were taken on average. In brackets, we give the number of instances for which the solver found the optimal solution. Note that the optimal solutions are known for all instances because at least one solver terminated within the time limit. When a solver managed to find the optimal solution on at least 8 out of the 10 instances, we compute  $cpu_o$ , excluding the instances where it timed out. Otherwise,  $cpu_o$  is left blank (-), meaning that the solver failed to find the optimal solution on too many instances for this metric to be meaningful.

From Table 1 it is clear that the ILP approach is more efficient than the CP one in this type of instances.  $Gurobi_k$  (resp.  $Gurobi_s$ ) times out on 1 (resp. 2) out of the 70 instances, whereas OR-Tools (resp. Choco) timed out on 40 (resp. 42) instances. With respect to the cases when the optimal solution was located within the time limit, even without proving optimality,  $Gurobi_k$  (resp.  $Gurobi_s$ ) found the optimal solution on 69 (resp. 68 instances), whereas OR-Tools (resp. Choco) on 59 (resp. 45) instances. This indicates that the CP solvers find the proof of optimality especially difficult. Regarding the time required to find the optimal solution (when found), all solvers are quite fast on smaller problems (with 5-10

## 30:12 The p-Dispersion Problem with Distance Constraints

■ **Table 1** Comparing exact solvers on random PDDPs. Cpu times are given in seconds.

Class ( $p,  P $ )	Gurobi <sub>k</sub> $\sum \text{cpu}$	cpu <sub>o</sub>	Gurobi <sub>s</sub> $\sum \text{cpu}$	cpu <sub>o</sub>	OR-Tools $\sum \text{cpu}$	cpu <sub>o</sub>	Choco $\sum \text{cpu}$	cpu <sub>o</sub>
(5,30)	0.7 (0)	0 (10)	1.6 (0)	0 (10)	3 (0)	0 (10)	3.5 (0)	0 (10)
(10,30)	2 (0)	0 (10)	6 (0)	0 (10)	998 (0)	1.7 (10)	>11,517 (2)	50 (10)
(20,30)	87 (0)	7 (10)	51 (0)	4 (10)	>10h (10)	69 (9)	>10h (10)	- (4)
(5,80)	7 (0)	0 (10)	20 (0)	0 (10)	13 (0)	0 (10)	3 (0)	0 (10)
(10,80)	12 (0)	0 (10)	78 (0)	0 (10)	>10h (10)	0.3 (10)	>10h (10)	0 (10)
(20,80)	727 (0)	69 (10)	725 (0)	28 (10)	>10h (10)	- (4)	>10h (10)	- (1)
(30,80)	>22,207 (1)	2,065 (9)	>10,519 (2)	414 (8)	>10h (10)	- (6)	>10h (10)	- (1)

facilities). Problems with 20 facilities and 30 potential locations are also easily manageable by the ILP models, whereas OR-Tools takes more than one minute on average to discover the optimal solution, and even fails to discover it on one instance. Choco fails even worse, failing to find the optimal solution on 6 instances, and failing to find any solution on one instance.

As the size of the problem grows, the ILP models, and even more so the CP solvers, find it harder to solve the instances. On problems with 30 facilities and 80 location points, the ILP models start giving time outs and take a long time to find the optimal solution, when they manage to do so, whereas the CP solvers, and especially Choco, fail to find the optimal on many instances from the (20,80) and (30,80) classes.

Table 2 compares the exact solvers to the heuristic CP solver (denoted CP<sub>h</sub>). For this solver, we report the total cpu time taken over the 10 instances of each class ( $\sum \text{cpu}$ ) and the mean cpu time taken to find the best solution it found within the time limit (cpu<sub>b</sub>). We also report (in brackets) the number of instances in which the solver managed to find the optimal solution. Then, in the following columns, we give the mean cpu times taken by the exact solvers to find a solution that at least matches the cost of the best solution found by the heuristic solver. In this way, we can evaluate the worth of the heuristic CP method as a heuristic for the PDDP. If the exact solvers manage to quickly match the best solution found by the heuristic one then there is not much point in using the heuristic solver. Whereas, if the heuristic solver quickly discovers a solution that the exact ones take very long to match then it is worth considering this approach for the PDDP. If an exact solver only managed to find a solution as good as that found by the heuristic solver in some instances, we give in brackets the number of times that this occurred, and we consider only these instances for the computation of the mean cpu time.

■ **Table 2** Comparing solvers on random PDDPs. Cpu times are given in seconds.

Class ( $p,  P $ )	CP <sub>h</sub> $\sum \text{cpu}$	cpu <sub>b</sub>	Gurobi <sub>k</sub>	Gurobi <sub>s</sub>	OR-Tools	Choco
(5,30)	0	0 (9)	0	0	0	0
(10,30)	0	0 (10)	0	0	1.7	50
(20,30)	2	0.2 (9)	5	1.7	65	160 (4)
(5,80)	0.1	0 (10)	0	0	0	0
(10,80)	1	0.1 (10)	0	0	0.3	0
(20,80)	2	0.2 (0)	3.5	3	6	0
(30,80)	36	3 (10)	2,218 (9)	1,051 (8)	1,760 (6)	3,453 (1)

As the results in Table 2 demonstrate, CP<sub>h</sub> is very fast as it managed to complete all 10 instances of each class in at most 2 seconds, except for the (30,80) class, on which it only took 36 seconds. Importantly, it also discovered the optimal solution in 58 out of the 70

instances, including all instances of the hard class (30,80) class. On the other hand, it did not manage to discover the optimal in any instance of the (20,80) class. Comparing the run times ( $\text{cpu}_b$ ) of  $\text{CP}_h$  to the exact solvers, results from all but the last class show that Gurobi manages to quickly match the best solution found by  $\text{CP}_h$ , even if the latter is faster on average. This does not always hold for OR-Tools which takes more than 1 minute on average in the (20,30) class. Choco takes 50 seconds on average in the (10,30) class and it manages to match the best solution of  $\text{CP}_h$  in only 4 instances of the (20,30) class. The benefits of the heuristic method in hard problems are demonstrated by the (30,80) class where it takes 3 secs on average to locate the optimal solution, whereas Gurobi<sub>k</sub> and Gurobi<sub>s</sub> require 2,218 and 1,051 secs respectively. OR-Tools found the optimal solution in only 6 out of the 10 instances of this class, taking 1,760 secs on average, while Choco managed to find the optimal in only one instance in 3,453 secs.

Finally, Table 3 takes a closer look at the performance of  $\text{CP}_h$ , with respect to the effect of the heuristic pruning method. To investigate this, we report the results obtained by the solver when the heuristic is deactivated (i.e. Function *Bound* is not called). In this case, the solver, denoted as  $\text{CP}_{-h}$ , operates as a typical CP solver. As in Table 1, we give the total cpu time taken, and in brackets the number of time outs, and the mean time taken to locate the optimal solution (the number of instances where the optimal solution was found is given in brackets). In the following column ( $\text{cpu}_h$ ) we give the mean cpu time taken by  $\text{CP}_{-h}$  to find a solution that at least matches the cost of the best solution found by  $\text{CP}_h$ , and in brackets, the number of times that it managed to do so. The next two columns give the mean numbers of visited search tree nodes for  $\text{CP}_{-h}$  and  $\text{CP}_h$  (the entry is left blank if there were time-outs). The last two columns give the average number of calls to Function *Bound* in  $\text{CP}_h$  and the percentage of fails caused by this function (i.e. the percentage of branches pruned by the heuristic).

■ **Table 3** A closer look at the performance of the custom solver, with and without the heuristic.

(p, $ P $ )	$\text{CP}_{-h} \sum \text{cpu}$	$\text{CP}_{-h} \text{cpu}_o$	$\text{CP}_{-h} \text{cpu}_h$	$\text{CP}_{-h} \text{nodes}$	$\text{CP}_h \text{nodes}$	calls	%fails
(5,30)	0.2 (0)	0 (10)	0 (10)	1,726	156	140	94
(10,30)	466 (0)	5 (10)	5 (10)	12.5M	229	218	95
(20,30)	>21,827 (3)	1 (10)	1 (10)	-	4,917	781	80
(5,80)	6 (0)	0.6 (10)	0.6 (10)	7,624	501	477	97
(10,80)	>10h (10)	- (5)	520 (5)	-	877	868	98
(20,80)	>10h (10)	- (0)	- (0)	-	835	816	98
(30,80)	>10h (10)	- (7)	1,044 (7)	-	17,207	12,995	81

Table 3 demonstrates the pruning power of our proposed heuristic. Without its use, the solver is able to handle the easier classes of problems, but not the harder ones, in accordance with standard CP solvers (Table 1). The solver is very successful compared to OR-Tools and Choco on the (20,30) class, as it times out in only 3 instances and finds the optimal solution in 1 sec on average. However, the solver fares badly on the (20,80) class where it is unable to find the optimal in any instance and actually finds worse solutions than  $\text{CP}_h$  in all instances. As for the (30,80) class,  $\text{CP}_{-h}$  finds the optimal solution in 7 instances, which is better than OR-Tools and Choco, but needs 1,044 secs on average to match the solution found by  $\text{CP}_h$ . Regarding the pruning achieved by the heuristic when it is activated, it is impressive that in most of the classes, there is a very high percentage of pruned branches over the total calls to the heuristic (up to 98%), which explains its success in speeding up search.



## 6.2 MDPLIB

The MDPLIB collects a large number of dispersion benchmarks (for both maxmin and maxsum p-dispersion) divided into various classes [28]. In the GKD, MDG, and SOM classes, the distances between the potential facility locations are given by Euclidean distances, random real numbers, and random integers, respectively. We took instances from these classes, having 100-250 potential facility points and 10-20 facilities, and we generated 10 PDDPs for each instance by randomly adding distance constraints between the facilities. For each distance constraint  $dis(f_i, f_j) > d_{ij}$  between facilities  $f_i$  and  $f_j$ ,  $d_{ij}$  was set to a random number in the interval  $[1, max]/2$ , where  $max$  is the maximum distance between any two potential location points, as specified in the base MDPLIB instance.

Table 4 compares the exact solvers on the MDPLIB-based problems. We exclude Choco as it is clearly inferior to OR-Tools. For each class we give the number of the MDPLIB instance on which it is based, and in brackets the numbers of potential locations and facilities to be located, e.g. a1(100,10) for MDG. For each solver, we report the total cpu time taken over the 10 instances ( $\sum$ cpu columns), the number of times when the optimal solution was found ( $\#opt$ ) columns, and the mean of the optimization function's value for the best solution found within the time limit. In the  $\sum$ cpu columns we give in brackets the number of instances in which the solvers timed out. In the  $\#opt$  columns we give in brackets the number of times when optimality was proved. Finally, in some cases, a solver did not manage to find any solution within the time limit. In such cases there is a subscript in the value of cost, giving the number of instances in which at least one solution was found. In such a case, the value of cost is computed over these instances only. In GKD classes with 250 points and 20 facilities, OR-Tools suffered memory exhaustion and crashed. This is denoted with MEM in the  $\sum$ cpu column.

The data in Table 4 demonstrates that the PDDPs generated using MDPLIB instances as basis can be very hard for both the ILP and CP approaches. None of the solvers terminates within the time limit on any instance with 20 facilities, while problems with 15 and 10 facilities are also quite hard. In addition, there are some instances of the larger classes (e.g. GKDD1(250,20)) where the solvers are unable to discover any solution within 1 hour of cpu time, let alone the optimal one. Comparing ILP to CP, Gurobi, with any of the two formulations, is in general more efficient than OR-Tools. Gurobi<sub>k</sub> (resp. Gurobi<sub>s</sub>) found the optimal solution in 77 (resp. 74) out of the 220 instances, and proved optimality in 73 (resp. 70) instances, whereas OR-Tools did not prove optimality in any instance (as it timed out on all of them) and found the optimal in 10 instances only. However, OR-Tools often managed to find better solutions than Gurobi in hard classes with 20 facilities, as the cost columns indicate, for instance in classes MDGa2(100,20) and MDGb2(100,20). Comparing Gurobi<sub>k</sub> to Gurobi<sub>s</sub>, there is no clear winner in terms of run times, but the latter managed to find solutions of better quality than the former in most of the classes. However, Gurobi<sub>s</sub> proved optimality or found the optimal solution in slightly fewer instances than Gurobi<sub>k</sub>.

Table 5 compares CP<sub>h</sub> to the exact solvers. For CP<sub>h</sub>, we report the total cpu time it takes over the 10 instances of each class (and the number of time-outs in brackets), the mean time it takes to find its best solution (cpu<sub>b</sub>), and the mean of the optimization function's value for the best solution it finds. For the exact solvers, we report the mean times taken to match the value of the best solution found by CP<sub>h</sub> (cpu<sub>h</sub>) columns, and the number of instances on which the solvers managed to find a solution that matches or improves the best solution found by CP<sub>h</sub> (in brackets). If this number is 0 or close to 0 then the entry in the cpu<sub>h</sub> column is left blank (-), as it is impossible or meaningless to compute the value of cpu<sub>h</sub>. If a value in the cpu<sub>b</sub> column is blank (-), e.g. GKDD1(100,20), then most of the 10 instances in this class were infeasible (in brackets we give the number of infeasible instances).

■ **Table 4** Comparing exact solvers on MDPLIB-generated PDDPs. Cpu times are given in seconds. The best mean value of the optimization function for each class is denoted in bold.

Class (p, P )	Gurobi <sub>k</sub> ∑cpu	#opt	cost	Gurobi <sub>s</sub> ∑cpu	#opt	cost	OR-Tools ∑cpu	#opt	cost
MDG									
a1(100,10)	>25,745 (3)	9 (7)	4.67	11,208 (0)	10 (10)	<b>4.68</b>	>10h (10)	7 (0)	4.59
a1(100,20)	>10h (10)	0 (0)	0.80 <sub>8</sub>	>10h (10)	0 (0)	1.16	>10h (10)	0 (0)	<b>1.17</b>
a2(100,10)	15,327 (0)	10 (9)	<b>4.74</b>	11,412 (0)	10 (10)	<b>4.74</b>	>10h (10)	1 (0)	4.36
a2(100,20)	>10h (10)	0 (0)	0.78 <sub>8</sub>	>10h (10)	0 (0)	0.83 <sub>9</sub>	>10h (10)	0 (0)	<b>1.40</b>
b1(100,10)	11,852 (0)	10 (10)	<b>460.11</b>	>29,171 (2)	10 (8)	<b>460.11</b>	>10h (10)	0 (0)	430.25
b1(100,20)	>10h (10)	0 (0)	102.42	>10h (10)	0 (0)	<b>109.35</b>	>10h (10)	0 (0)	77.33
b2(100,10)	10,305 (2)	8 (8)	459.65	27,894 (2)	8 (8)	<b>459.99</b>	>10h (10)	1 (0)	413.85
b2(100,20)	>10h (10)	0 (0)	106.77 <sub>9</sub>	>10h (10)	0 (0)	81.09	>10h (10)	0 (0)	<b>113.33</b>
GKD									
d1(100,10)	4,743 (0)	10 (10)	<b>34.06</b>	5,415 (0)	10 (10)	<b>34.06</b>	>10h (10)	0 (0)	33.04
d1(100,20)	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-
d1(250,10)	>15,448 (4)	6 (6)	<b>35.98</b>	34,753 (9)	3 (1)	35.27	>10h (10)	0 (0)	-
d1(250,20)	>10h (10)	0 (0)	9.95 <sub>4</sub>	>10h (10)	0 (0)	10.55 <sub>2</sub>	MEM	0 (0)	-
d2(100,10)	>5,998 (1)	9 (9)	34.34	2,602	10 (10)	<b>34.82</b>	>10h (10)	0 (0)	31.18
d2(100,20)	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-
d2(250,10)	>22,334 (6)	4 (4)	35.94	>33,595 (8)	2 (2)	<b>36.31</b>	>10h (10)	0 (0)	-
d2(250,20)	>10h (10)	0 (0)	-	>10h (10)	0 (0)	-	MEM	0 (0)	-
SOM									
a21(100,10)	>20,260 (1)	10 (9)	<b>5</b>	9,585 (0)	10 (10)	<b>5</b>	>10h (10)	1 (0)	4.1
a21(100,15)	>34,887 (9)	1 (1)	<b>2.2</b>	>35,371	1 (1)	<b>2.2</b>	>10h (10)	0 (0)	2
a21(100,20)	>10h (10)	0 (0)	1 <sub>6</sub>	>10h (10)	0 (0)	1 <sub>6</sub>	>10h (10)	0 (0)	1 <sub>9</sub>
a41(150,15)	>10h (10)	0 (0)	2.6	>10h (10)	0 (0)	2.6	>10h (10)	0 (0)	<b>3</b>
a41(150,20)	>10h (10)	0 (0)	1 <sub>9</sub>	>10h (10)	0 (0)	1.11 <sub>9</sub>	>10h (10)	0 (0)	1
b5(200,20)	>10h (10)	0 (0)	1.4	>10h (10)	0 (0)	<b>2</b>	>10h (10)	0 (0)	-

Results from Table 5 demonstrate the efficiency of the heuristic CP approach. Regarding run times, CP<sub>h</sub> times out in only 7 instances and terminates quickly in all instances of all classes, except for the hard GKDd1(250,20) and GKDd2(250,20). CP<sub>h</sub> proved the infeasibility of the 16 infeasible instances of classes GKDd1(100,20) and GKDd2(100,20) and found solutions in the other 4, whereas none of the other solvers managed to do so in any instance. Of course, the fast proof of infeasibility is not due to the bounding mechanism of CP<sub>h</sub>, as this is not invoked, but most likely due to the lightweight model and mechanics of the custom-written solver.

Regarding the quality of the solutions, as a downside, CP<sub>h</sub> finds the optimal in only 2 out of the 80 instances of smaller size, for which the optimal is known (excluding infeasible ones). However, in these classes, the exact solvers (and especially OR-Tools) can be orders of magnitude slower than CP<sub>h</sub> in discovering solutions of the same quality as CP<sub>h</sub>, which reaches its best solution in less than 0.1 secs in most cases. This is evident in class SOMa41(150,15) where CP<sub>h</sub> found its best solution in less than 0.1 secs on average, while the exact solvers took more than 1,000 secs to match the solution quality of CP<sub>h</sub>, on instances where they managed to do this. However, the best solution discovered by these solvers (including OR-Tools) is typically better than the best solution discovered by CP<sub>h</sub>.

■ **Table 5** Comparing solvers on MDPLIB-generated PDDPs. We denote with bold the mean cost of  $CP_h$  on classes where it was better than the mean cost of all the other solvers.

Class ( $p,  P $ )	$CP_h$ $\sum$ cpu	cpu <sub>b</sub>	cost	Gurobi <sub>k</sub> cpu <sub>h</sub>	Gurobi <sub>s</sub> cpu <sub>h</sub>	OR-Tools cpu <sub>h</sub>
MDG						
a1 (100,10)	0.5 (0)	0	4.35	243 (10)	55 (10)	592 (10)
a1 (100,20)	81 (0)	8	<b>1.69</b>	- (0)	- (1)	- (0)
a2 (100,10)	0.6 (0)	0	4.24	137 (10)	22 (10)	873 (7)
a2 (100,20)	137 (0)	12	<b>1.64</b>	- (0)	- (0)	- (0)
b1 (100,10)	0.4 (0)	0	428.18	10 (10)	345 (10)	187 (10)
b1 (100,20)	54 (0)	4.5	<b>181.20</b>	- (0)	- (0)	- (1)
b2 (100,10)	0.7 (0)	0	428.17	27 (10)	41 (10)	723 (3)
b2 (100,20)	92 (0)	8	<b>159.93</b>	- (0)	- (0)	- (1)
GKD						
d1 (100,10)	1.8 (0)	0.1	33.27	94 (10)	224 (10)	717 (5)
d1 (100,20)	405 (0)	- (9)	-	- (0)	- (0)	- (0)
d1 (250,10)	10 (0)	0.6	34.24	800 (8)	2,074 (6)	- (0)
d1 (250,20)	>18,689 (3)	1,140	<b>16.94<sub>9</sub></b>	- (0)	- (0)	- (0)
d2 (100,10)	1.6 (0)	0	31.29	91 (9)	179 (10)	1,277 (6)
d2 (100,20)	962 (0)	- (7)	-	- (0)	- (0)	- (0)
d2 (250,10)	8 (0)	0.5	35.06	121 (6)	997 (8)	- (0)
d2 (250,20)	>23,767 (4)	1,597	<b>13.28<sub>9</sub></b>	- (0)	- (0)	- (0)
SOM						
a21 (100,10)	0 (0)	0	4	7 (10)	3 (10)	106 (10)
a21 (100,15)	0 (0)	0	2	39 (10)	45 (10)	206 (10)
a21 (100,20)	11 (0)	1	1	41 (6)	49 (6)	522 (9)
a41 (150,15)	2 (0)	0	<b>3</b>	1,170 (6)	1,741 (6)	1,465 (10)
a41 (150,20)	12 (0)	1	<b>1.9</b>	- (0)	- (1)	- (1)
b5 (200,20)	10 (0)	0.3	2	- (4)	1,272 (10)	- (0)

But the power of  $CP_h$  as a heuristic method for PDDP is truly evident on the larger classes with 20 facilities where it discovers solutions of (much) better quality than the exact solvers, and excluding the two hard GKD classes with 250 location points and 20 facilities, it does this very fast. Also,  $CP_h$  finds solutions in all 20 instances of the two hard GKD classes, while Gurobi<sub>k</sub> (resp. Gurobi<sub>s</sub>) in only 4 (resp. 2), and OR-Tools in none.

## 7 Conclusion

We have studied a variant of the p-dispersion problem where distance constraints exist between the facilities to be dispersed. We proposed ILP formulations and a CP model for this problem. We also devised a heuristic CP-based method built around a bounding technique that prunes the search tree by reasoning about the best possible value of the optimization function at each node. Experimental results demonstrated that although the ILP formulations are more efficient than the CP model, they fail to efficiently handle problems with more than 10 facilities, whereas on such problems the heuristic CP method manages to find solutions of better quality than the ILP and CP models in orders of magnitude shorter run times.

## References

- 1 T. Argo and E. Sandstrom. Separation distances in NFPA codes and standards (tech. rep.). Fire Protection Research Foundation. 2014.
- 2 O. Berman and R. Huang. The minimum weighted covering location problem with distance constraints. *Computers and Operations Research*, 35(12):356–372, 2008.
- 3 F. Boussemart, F. Heremy, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 482–486, 2004.
- 4 J. Brimberg and H. Juel. A minisum model with forbidden regions for locating a semi-desirable facility in the plane. *Location Science*, 6(1):109–120, 1998.
- 5 H. Cambazard, D. Mehta, B. O’Sullivan, and L. Quesada. A computational geometry-based local search algorithm for planar location problems. In *Proceedings of the 9th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2012)*, pages 97–112, 2012.
- 6 S. Chaudhry, T. McCormick, and I. D. Moon. Locating independent facilities with maximum weight: Greedy heuristics. *International Journal of Management Science*, 14(5):383–389, 1986.
- 7 R. L. Church and M. E. Meadows. Results of a new approach to solving the p-median problem with maximum distance constraints. *Geographical Analysis*, 9(4):364–378, 1977.
- 8 V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
- 9 W. Comley. The location of ambivalent facilities: Use of a quadratic zero-one programming algorithm. *Applied Mathematical Modeling*, 19(1):26–29, 1995.
- 10 Z. Dai, K. Xu, and M. Ornik. Repulsion-based p-dispersion with distance constraints in non-convex polygons. *Annals of Operations Research*, 307:75–91, 2021.
- 11 Choco development team. An Open-Source java library for constraint programming. <https://choco-solver.org/>.
- 12 OR-Tools development team. OR-Tools, CP-SAT solver. [https://developers.google.com/optimization/cp/cp\\_solver](https://developers.google.com/optimization/cp/cp_solver).
- 13 T. Drezner, Z. Drezner, and A. Schöbel. The weber obnoxious facility location model: A big arc small arc approach. *Computers and Operations Research*, 98:240–250, 2018.
- 14 Z. Drezner, P. Kalczyński, and S. Salhi. The planar multiple obnoxious facilities location problem: A Voronoi based heuristic. *Omega*, 87:105–116, 2019.
- 15 E. Erkut. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990.
- 16 E. Erkut and S. Neuman. Analytical models for locating undesirable facilities. *European Journal of Operational Research*, 40(3):275–291, 1989.
- 17 M. M. Fazel-Zarandi and J. C. Beck. Solving a location-allocation problem with logic-based benders’ decomposition. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP 2009)*, pages 344–351, 2009.
- 18 J. B. Ghosh. Computational aspects of the maximum diversity problem. *Operations Research Letters*, 19(4):175–181, 1996.
- 19 C. Gomes and M. Sellmann. Streamlined constraint reasoning. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP 2004)*, pages 274–289, 2004.
- 20 T. Guns. Increasing modeling language convenience with a universal n-dimensional array, CPython as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation*, 2019.
- 21 LLC Gurobi Optimization. Gurobi optimizer reference manual. , 2023. URL: <https://www.gurobi.com>.
- 22 N. Isoart and J.-C. Régin. A k-Opt Based Constraint for the TSP. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, 2021.

## 30:18 The p-Dispersion Problem with Distance Constraints

- 23 B. M. Khumawala. An efficient algorithm for the p-median problem with maximum distance constraints. *Geographical Analysis*, 5(4):309–321, 1973.
- 24 R. K. Kincaid. Good solutions to discrete noxious location problems via meta-heuristics. *Annals of Operations Research*, 40(1):265–281, 1992.
- 25 M. J. Kubly. Programming models for facility dispersion: The p-dispersion and maximum dispersion problems. *Mathematical and Computer Modelling*, 10(10):792, 1988.
- 26 Mikael Zayenz Lagerkvist and Magnus Rattfeldt. Half-checking propagators. In *Proceedings of the 19th workshop on Constraint Modelling and Reformulation*, 2020.
- 27 A. Maier and H.W. Hamacher. Complexity results on planar multifacility location problems with forbidden regions. *Mathematical Methods of Operations Research*, 89:433–484, 2019.
- 28 R. Marti, A. Martinez-Gavara, S. Perez-Pelo, and J. Sanchez-Oro. A review on discrete diversity and dispersion maximization from an OR perspective. *European Journal of Operational Research*, 299(3):795–813, 2022.
- 29 I. D. Moon and S. Chaudhry. An analysis of network location problems with distance constraints. *Management Science*, 30(3):290–307, 1984.
- 30 I. D. Moon and L. Papayanopoulos. Minimax location of two facilities with minimum separation: Interactive graphical solutions. *Journal of the Operations Research Society*, 42:685–694, 1991.
- 31 C.S. Orloff. A theoretical model of net accessibility in public facility location. *Geographical Analysis*, 9:244–256, 1977.
- 32 M. G. Resende, R. Marti, M. Gallego, and A. Duarte. Grasp and path relinking for the max-min diversity problem. *Computers and Operations Research*, 37(3):498–508, 2010.
- 33 B. Saboonchi, P. Hansen, and S. Perron. MaxMinMin p-dispersion problem: A variable neighborhood search approach. *Computer & Operations Research*, 52:251–259, 2014.
- 34 D. Sayah and S. Irnich. A new compact formulation for the discrete p-dispersion problem. *European Journal of Operational Research*, 256(1):62–67, 2017.
- 35 F. Sayyady and Y. Fathi. An integer programming approach for solving the p-dispersion problem. *European Journal of Operational Research*, 253(1):216–225, 2016.
- 36 D. R. Shier. A min-max theorem for p-center problems on a tree. *Transportation Science*, 11(3):243–252, 1977.
- 37 S.Y.D. Sorkhabi, D.A. Romero, J.C. Beck, and C. Amon. Constrained multi-objective wind farm layout optimization: Novel constraint handling approach based on constraint programming. *Renewable Energy*, 126(C):341–353, 2018.
- 38 B.C. Tansel, R.L. Francis, T.J. Lowe, and M.L. Chen. Duality and distance constraints for the nonlinear p-center problem and covering problem on a tree network. *Operations Research*, 30(4):725–744, 1982.
- 39 S.B. Welch and S. Salhi. The obnoxious p facility network location problem with facility interaction. *European Journal of Operational Research*, 102:302–319, 1997.
- 40 49 C.F.R. §175.701. Separation distance requirements for packages containing class 7 (radioactive) materials in passenger-carrying aircraft. Title 49 Code of Federal Regulations, Part 175. 2021.
- 41 29 C.F.R. §1910.157. Portable fire extinguishers. Title 29 Code of Federal Regulations, Part 157. 2021.

# Partially Preemptive Multi Skill/Mode Resource-Constrained Project Scheduling with Generalized Precedence Relations and Calendars

Guillaume Povéda  

Airbus (AI Research), Toulouse, France

Nahum Alvarez  

Airbus (AI Research), Toulouse, France

Christian Artigues  

LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

---

## Abstract

Multi skill resource-constrained project scheduling Problems (MS-RCPSP) have been object of studies from many years. Also, preemption is an important feature of real-life scheduling models. However, very little research has been investigated concerning MS-RCPSPs including preemption, and even less research moving out from academic benchmarks to real problem solving. In this paper we present a solution to those problems based on a hybrid method derived from large neighborhood search incorporating constraint programming components tailored to deal with complex scheduling constraints. We also present a constraint programming model adapted to preemption. The methods are implemented in a new open source python library allowing to easily reuse existing modeling languages and solvers. We evaluate the methods on an industrial case study from aircraft manufacturing including additional complicating constraints such as generalized precedence relations, resource calendars and partial preemption on which the standard CP Optimizer solver, even with the preemption-specific model, is unable to provide solutions in reasonable times. The large neighborhood search method is also able to find new best solutions on standard multi-skill project scheduling instances, performing better than a reference method from the literature.

**2012 ACM Subject Classification** Computing methodologies → Planning and scheduling; Applied computing → Industry and manufacturing; Theory of computation → Optimization with randomized search heuristics; Theory of computation → Constraint and logic programming

**Keywords and phrases** Large-scale scheduling problem, partial preemption, multi-skill, multi-mode, resource calendars, constraint programming, large neighborhood search

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.31

**Funding** This work received the support of the Support from the French ANR-3IA Artificial and Natural Intelligence Toulouse Institute (ANITI).

*Guillaume Povéda:* ANITI

*Nahum Alvarez:* ANITI

*Christian Artigues:* ANITI

**Acknowledgements** The authors are grateful to the anonymous reviewers for their constructive comments. In particular, we thank the anonymous reviewer that pointed out the issue linked to preemption in the *CP-Base* model and made inspiring suggestions that led to the *CP-SmartPreemption* model.

## 1 Introduction

Multi skill resource-constrained project scheduling problems (from hereon MS-RCPSP) are critical in business applications like automotive, human resources, aerospace or nuclear industry [3]; managing efficiently the workforce assigned to perform required tasks directly



© Guillaume Povéda, Nahum Alvarez, and Christian Artigues;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 31; pp. 31:1–31:21



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

impacts the performance and progress of such businesses. This type of problems have been widely studied in academia from long, as well as the more general yet less compact multi-mode model involving non renewable resources [36]. Also, preemption in scheduling is found in many works due to the flexibility it brings to improve the performance of a practical application. However, the scheduling literature including both multi skill and task preemption, all the more grounded in a real business application, is rather sparse [1, 34].

The presence of multiple skilled resources adds more complexity to the problem, and new types of goals aside the usual makespan and cost optimization, like assignment score and worker performance [35]. Task preemption adds another layer of complexity, and in real scenarios usually is tied to resource availability, with special focus in worker shifts and calendars. Also, it is important to note that whilst a task can be preempted, pausing its progress, maybe only part of the resources can be freed to make them available for other tasks, with others being locked in place. This has been already referred to as partial preemption [25]. Finally, practical problems also contain generalized precedence constraints and resource calendars, which have been well known from long ago. These constraints extend simple tasks precedence to detail concrete temporal conditions between tasks. Calendars on the other hand, define resource unavailability periods, hence both features make the problem harder to solve [17].

In this paper, we present a method oriented towards a real assembly line use case, requiring to include all these five features: multi skilled resources, multiple modes, (partially) preemptive tasks, calendars and generalized precedence constraints. Hence, we call our model the partially preemptive- multi-skill/mode resource-constrained project scheduling problem with generalized precedence relations and resource calendars (PP-MS-MM-RCPSP/max-cal). Our solution is aimed for industrial domains, but can solve generic problems in the scope of the model as well. We propose an extended version of the large neighborhood search (LNS) method [21, 11], softening constraints initially to find a base solution that will be improved on each iteration. We observed this approach to be faster than other more direct constraint programming approaches. We evaluated our method against different benchmarks, including one using real data from aerospace manufacturing plants, real instances from an hazardous material examination facility [25] and standard multi-skill scheduling instances [38].

The rest of the paper is structured as follows: in Section 2, literature about multi skill scheduling and preemption in scheduling is discussed, along with previous scheduling works oriented towards industrial applications. Section 3 describes the domain of our problem and contains the formulation of our model, including a new CP formulation tailored to preemption. In Section 4 we detail our approach explaining the algorithm we developed to solve PP-MS-RCPSP/max-cal. Following this, Section 5 contains the experiments and benchmarks conducted to evaluate the performance of our solution. Lastly, our conclusions can be seen in Section 6.

## **2 Multi-skill/mode, preemption, calendars and generalized precedence in the literature**

The multi-mode RCPSP (MM-RCPSP) allows reaching a compromise between resource usage and task duration as it frequently occurs in practice. It also includes non renewable resources to model budget constraints that may prevent from using the fastest modes for all tasks. Many solution methods have been proposed over the years from local search, to sat-based methods using hyperheuristics. [6, 9, 15, 37]. CP approaches are currently highly popular and efficient tools to solve RCPSP variants [18] and in particular the MM-RCPSP [36].

MS-RCPSPs are a particular MM-RCPSP based on the concept of skills enabling the compact representation of a possibly large set of modes corresponding to combination of resources managing a set of skills required by a task. There is a variety in the methods and models to solve the problem: tree search [29], genetic algorithms [20], mixed-integer linear programming [3, 10, 19, 31]. On the standard MS-RCPSP, we will compare our method with two state-of-the-art methods: the CP approach using no-good learning proposed in [38] and the greedy randomized adaptive search procedure (GRASP) presented in [25].

Preemption can be defined as the capability of stopping the execution of a task in the generated schedule, releasing the resources it was using in order to allow for the execution of a different one, and being able to be continued later. In some cases preemption makes the problem easier to solve when an NP-hard non preemptive scheduling problem has a polynomial preemptive counterpart [5] but in the case of job-shop or resource-constrained project scheduling the problem may become harder to solve due to a much larger search space [7, 22]. Only a handful of studies have combined multi-skilled resources and preemption in the same work. We can cite [12, 24, 25, 26], where mixed-integer linear programming, constraint programming and metaheuristics methods were proposed. Prominent preemption can cause trouble to CP approaches: in [25] an experiment revealed that MILP obtained better results on a set of highly preemptive MS-RCPSP than the default search of IBM CP Optimizer. For preemptive MS-RCPSP, we will compare our method with the hybrid GRASP-LNS heuristics proposed in [26]. For their application, the authors also considered partially preemptive tasks, where only part of the resources are released during preemption. Due to industrial relevance, we also include partial preemption in our model.

Resource calendars are often unavoidable characteristics in industrial contexts. They can be linked to a basic form of preemption in the sense that an on-going task has to be preempted when one of its resource becomes unavailable due to an off-time in the calendar. In [13], calendars were used in a MS-RCPSP context without including task preemption. Our model considers both situations and tasks can be preempted or not by calendars as frequently observed [17].

Generalized precedence constraints, also considered in our model specify more complex temporal relations between tasks than standard precedence constraints. They make the problem more complex in all RCPSP variants as even finding a feasible solution becomes NP-hard in the presence of generalized precedence constraints. Many works have considered such constraints in the standard RCPSP [4, 32] and also for multi-mode preemptive RCPSP [27]. The RCPSP/max-cal problem involving both generalized precedence and calendar constraints was again successfully solved by CP in [17].

A difference between the performance of generic solvers on academic RCPSP models and on their adaptation to domain problems with very specific requirements is observed in other works and we also could experience it in our benchmark experiments: usually the more special characteristics the problem has, the worse is the performance of generic solvers. Our model falls into this category as it includes all the above-mentioned complicating characteristics. The LNS approach appears as a technique of choice to find the right compromise between genericity and performance.

### **3 PP-MS-MM-RCPSP/max-cal: definition and formulation**

The PP-MS-MM-RCPSP/max-cal generalizes the partially preemptive MS-RCPSP introduced in [26]. An additional feature is the possibility to execute the task in different modes, in the same way it is done in multi-mode RCPSP [15]. Generalized precedence constraints and



calendars are also added. Formally a PP-MS-MM-RCPSP/max-cal is defined by:

1.  $\mathcal{A}$  a set of activities to schedule,  $\mathcal{A} = \{1, \dots, n\}$ , two special activities are created by convention: the source task 1 (predecessor of all other tasks) and sink task  $n$  (successor of all other tasks);
2. Each activity  $i \in \mathcal{A}$  can be performed in  $m_i$  different modes,  $m_i \in M_i = \{1, \dots, |M_i|\}$ ; (as defined in [28])
3.  $\mathcal{R}^\rho$  a set of renewable resource types,  $\mathcal{R}^\rho = \{R_1^\rho, \dots, R_{m^\rho}^\rho\}$ .  $\forall k \in \mathcal{R}^\rho, t \in \mathbb{N}, B_{kt}$  is the discrete amount of available resource  $k$  at discrete time  $t$  (available between time  $t$  and  $t + 1$ );
4.  $\mathcal{R}^\nu$  a set of non-renewable resource types,  $\mathcal{R}^\nu = \{R_1^\nu, \dots, R_{m^\nu}^\nu\}$ ;  $\forall k \in \mathcal{R}^\nu, B_k$  is the total capacity of the non-renewable resource;
5.  $\mathcal{O}$  the set of disjunctive resources representing individual skilled workers (from now on, we will refer to them as Operators);
6.  $\mathcal{L} = \{1, \dots, L\}$  the set of skills;
7.  $\forall o \in \mathcal{O}, t \in \mathbb{N}, A_{ot} \in \{0, 1\}$  indicates if the Operator  $o$  is present or not at time  $t$  (hence we treat the temporal availability of Operators as fixed from the problem definition);
8.  $\forall o \in \mathcal{O}, l \in \mathcal{L}, y_{ol} \in \{0, 1\}$  indicates if Operator  $o$  masters skill  $l$ ;
9.  $\forall i \in \mathcal{A}, r_i$  is the release time of task  $i$ ;
10.  $\forall i \in \mathcal{A}, d_i$  is the deadline time of task  $i$ ;
11.  $\forall i \in \mathcal{A}, p_{i,m_i}$  is the processing time under mode  $m_i \in M_i$ ;
12.  $\forall i \in \mathcal{A}, k \in \mathcal{R}^\rho \cup \mathcal{R}^\nu, b_{i,m_i,k}$  is a natural number representing the resource demand of activity  $i$  for resource  $k$ , under mode  $m_i$ ;
13.  $\forall i \in \mathcal{A}, l \in \mathcal{L}, s_{i,m_i,l}$  is a natural number representing the skill requirement of activity  $i$  under mode  $m_i$ ;
14.  $P$  is the set of precedence constraints  $\mathcal{A} \times \mathcal{A}$  specifying which activity should precede another one;
15.  $P_{sync-start}$  is the set of activity pairs that must start at the same time;
16.  $P_{startlag}$  is the set of ordered pairs  $(i, j)$  specifying  $\Delta s_{(i,j)}$ , the minimum time lag between start of  $i$  and start of  $j$ ;
17.  $P_{sync-end}$  is the set of activity pairs that must end at the same time;
18.  $P_{endlag}$  is the set of ordered pairs  $(i, j)$  specifying  $\Delta e_{(i,j)}$ , the minimum time lag between end of  $i$  and start of  $j$ ;
19. The set  $\mathcal{A}$  is split in three different subsets :
  - $\mathcal{A} = \mathcal{A}^P \cup \mathcal{A}^{NP} \cup \mathcal{A}^{PP}$ ;
  - $\mathcal{A}^P$  is the set of fully preemptive activities (activities can be preempted and all the resources are released);
  - $\mathcal{A}^{NP}$  is the set of non-preemptive activities;
  - $\mathcal{A}^{PP}$  is the set of partially preemptive activities (where at least one resource is not releasable);
20.  $\forall i \in \mathcal{A}, m_i \in M_i, k \in \mathcal{R}^\rho, \rho_{i,m_i,k} \in \{0, 1\}$  indicates if resource  $k$  is releasable for activity  $i$  under mode  $m_i$ .

### 3.1 Known variants in the literature

Our generic formulation encompasses more classical scheduling problems, that can be therefore solved using the proposed solution, such as:

- the classical RCPSP ( $\mathcal{L} = \emptyset, \mathcal{R}^\nu = \emptyset, \forall k \in \mathcal{R}^\rho, \forall t, t' \in \mathbb{N}, B_{kt} = B_{kt'}, \mathcal{O} = \emptyset, \mathcal{A}^P = \emptyset, \mathcal{A}^{PP} = \emptyset, \forall i \in \mathcal{A}, M_i = \{1\}$ );

- the classical multi-mode RCPSP ( $\mathcal{L} = \emptyset, \forall k \in \mathcal{R}^p, \forall t, t' \in \mathbb{N}, B_{kt} = B_{kt'}, \mathcal{O} = \emptyset, \mathcal{A}^P = \emptyset, \mathcal{A}^{PP} = \emptyset, \forall i \in \mathcal{A}, M_i = \{1, \dots, |M_i|\}$ );
- the classical multi-skill RCPSP ( $\mathcal{L} = \{1, \dots, L\}, \mathcal{O} \neq \emptyset, \mathcal{A}^P = \emptyset, \mathcal{A}^{PP} = \emptyset, \forall i \in \mathcal{A}, M_i = \{1, \dots, |M_i|\}$ ).

## 3.2 Constraint Programming formulation

We developed a combinatorial optimisation library containing the CP model for PP-MS-MM-RCPSP/max-cal. It can be found as part of an open source framework to solve discrete optimization problems <sup>1</sup>.

A big focus of the library is on the RCPSP's class of problems described in this paper. All solver approaches, from Local search, rule based heuristics, CP, LP and LNS methods are either coded using the discrete-optimisation library or wrapped into it, allowing the user to easily benchmark methods and hybridize different approaches.

The following sections detail the decision variables and constraints of our formulation for PP-MS-MM-RCPSP/max-cal.

### 3.2.1 Decision variables

We assume that each task can be preempted at regular discretized time points, resulting in a sequence of small non-preemptive chunks for each activity. A subpart of an activity is thus defined as a subset of adjacent chunks of this activity. Let's define  $max_{preemption}$  as an arbitrary input of our problem which represents an upper bound on the number of preemption breaking times allowed for all our activities. We will note  $\mathcal{J} = \{1, \dots, max_{preemption}\}$  the set of preemption breaking time indexes and  $\mathcal{J}^- = \{2, \dots, max_{preemption}\}$

1. Starting time decision :  $starts$  is a  $|\mathcal{A}| \times max_{preemption}$  matrix, which contains the starting time of subparts of all the tasks in increasing order.
2. Duration decision :  $durations$  is a  $|\mathcal{A}| \times max_{preemption}$  matrix, which contains the duration of subparts of all the tasks.
3. Mode decision :  $modes$  is a  $|\mathcal{A}|$  vector specifying in which mode a task is executed.  $\forall i \in \mathcal{A}, modes[i] \in M_i$ .
4. Resource allocation :  $allocation$  is a  $|\mathcal{O}| \times |\mathcal{A}| \times max_{preemption}$  3D binary matrix which indicates which worker  $o$  is allocated to each subpart of an activity.

### 3.2.2 Constraints

We will use the notation  $[\cdot]$  in matrix indexing to ease the readability of constraints. For e.g  $starts[i, \cdot]$  is the vector of starting times for the task  $i$  and  $allocations[o, \cdot, \cdot]$  is the  $|\mathcal{A}| \times max_{preemption}$  matrix allocation of resource  $o$  to all activity subparts.

1. Resource consumption constraint :

$$\forall k \in \mathcal{R}^p, cumulative(starts, durations, b_{modes, k}, B_k)$$

where  $b_{modes, k}$  is an array of dimension  $|\mathcal{A}| \times max_{preemption}$  defined by:

$\forall i \in \mathcal{A}, j \in \mathcal{J}, b_{modes, k}[i, j] = b_{i, modes[i], k}$ , being  $b_{modes, k}[i, j]$  the resource demand of activity  $i$  on the  $j$ -th subpart of its execution, which doesn't depend on  $j$  in our problem.

We use the global cumulative constraint implemented in most CP language to model the cumulative resource consumption in RCPSPs [30].

<sup>1</sup> <https://github.com/airbus/discrete-optimization>

To take into account variable resource availability, we use a discrete stepped function  $max_t(B_{kt}) - B_{kt_{step}}$  at each  $t_{step}$  that includes an artificial fixed task consuming it. This way, whenever availability changes, that task removes the resources from the pool during a period defined by the step function. In our domain, variable resource availability translates into worker's shifts and calendars, also indirectly defining points where a task needs to be preempted. For simplicity we didn't include it in the description of the constraint.

2. Non-renewable resources:

$\forall k \in \mathcal{R}^\nu, \sum_{i \in \mathcal{A}} b_{i, modes[i], k} \leq B_k$ . This non-renewable resource constraint is only meaningful when there are several possible modes. In multi-mode settings, some *modes* value wouldn't satisfy the constraint.

3. Skills requirement:

$$\forall i \in \mathcal{A}, j \in \mathcal{J}, l \in \mathcal{L},$$

$$durations[i, j] > 0 \rightarrow \sum_{o \in \mathcal{O}} allocation[o, i, j] \cdot y_{ol} \geq s_{i, modes[i], l}$$

Contrary to other multi-skill variants, we consider that Operators assigned to a task contribute to the task skill requirements with all of their skills, similar to what it is seen in [26].

4. Operator availability:

$$\forall o \in \mathcal{O}, cumulative(starts, durations, allocation[o, :, :], A_o)$$

Variable availability of an Operator is dealt with in the same way as Constraint 1 above.

5. Precedence relation:  $\forall (i, j) \in P$ ,

$$starts[j, 1] \geq starts[i, maxpreemption] + durations[i, maxpreemption]$$

6.  $P_{sync-start}$  relations:

$$\forall (i, j) \in P, starts[i, 1] = starts[j, 1]$$

7.  $P_{startlag}$  relations:

$$\forall (i, j) \in P_{startlag}, starts[j, 1] \geq starts[i, 1] + \Delta s_{(i,j)}$$

8.  $P_{sync-end}$  relations:  $\forall (i, j) \in P_{sync-end}$ ,

$$starts[j, 1] = starts[i, maxpreemption] + durations[i, maxpreemption]$$

9.  $P_{endlag}$  relations:

$$\forall (i, j) \in P_{endlag},$$

$$starts[j, 1] \geq starts[i, maxpreemption] + durations[i, maxpreemption] + \Delta e_{(i,j)}$$

10. Tasks duration :

a.  $\forall i \in \mathcal{A}^{NP}, durations[i, 1] = p_{i, modes[i]} \wedge (\forall j \in [2, maxpreemption], durations[i, j] = 0)$

b.  $\forall i \in \mathcal{A} \setminus \mathcal{A}^{NP}, \sum_{j \in \mathcal{J}} durations[i, j] = p_{i, modes[i]}$

11. Release time:

$$\forall i \in \mathcal{A}, starts[i, 1] \geq r_i$$

12. Deadline time:

$$\forall i \in \mathcal{A}, starts[i, maxpreemption] + durations[i, maxpreemption] \leq d_i$$

13. Conventions constraints for *starts* and *durations*:

The following constraints are modeling choices aiming to help the solver to explore the search space.

- a.  $\forall i \in \mathcal{A}, j \in \mathcal{J}^-, starts[i, j] \geq starts[i, j-1] + durations[i, j]$  : precedence constraints between each subparts of the task.

- b.  $\forall i \in \mathcal{A}, j \in [1, maxpreemption - 1], durations[i, j] = 0 \rightarrow durations[i, j+1] = 0$ : As soon as there is a 0 duration subtask, all the following ones are 0 too.

- c.  $\forall i \in \mathcal{A}, j \in [1, maxpreemption], durations[i, j] = 0 \rightarrow starts[i, j] = starts[i, j-1] + durations[i, j-1]$ : Whenever there is a 0 duration, the remaining *starts* values are uniquely determined by previous values, pruning redundant solutions.

## 14. Partially preemptive and non-releasable resources :

We introduce variable  $blockedduration[i, j, k]$ ,  $\forall i \in \mathcal{A}$ ,  $j \in \mathcal{J}$ ,  $k \in \mathcal{R}^p$  as the usage duration of resource  $k$  for task  $i$  and activity subpart  $j$ . It is regulated by the following constraint:  $\forall j \in \mathcal{J}$ ,

$$\rho_{i, modes[i], k} = 1 \rightarrow blockedduration[i, j, k] = durations[i, j]$$

This means if the resource is releasable then the duration of the usage is the same as the duration of the subtask.

On the other hand, to describe the consumption of a resource over the total span of the activity for not releasable resources, we have:

$$\rho_{i, modes[i], k} = 0 \rightarrow (blockedduration[i, 1, k] = starts[i, maxpreemption] - starts[i, 1]) \wedge (blockedduration[i, j, k] = 0, \forall j \in \mathcal{J}^-)$$

This will set  $blockedduration[i, 1, k]$  to the total span of the activity, including preempted time, and ignore the other subparts.

Then, when indicating the *cumulative* constraint, instead of the normal duration, this resource uses *blockedduration*:

$$\forall k \in \mathcal{R}^p, cumulative(starts, blockedduration, b_{modes, k}, B_k)$$

We note that Constraint (13) implementing preemption won't always lead to a feasible solution without backtracking even if there are no time windows or maximum time lags when using CP-Optimizer solver.

Let us take a simple example of a single preemptive activity  $A_1$  of duration 5 task decomposed in 3 subtasks  $A_{1,1}$ ,  $A_{1,2}$  and  $A_{1,3}$ . Suppose the solver sets the start time of  $A_{1,1}$  to 0 and the end time of  $A_{1,1}$  to 3 (and consequently its duration to 3). Suppose now that the solver sets the start time and the end time of  $A_{1,2}$  to 3 (duration 0). Then according to constraint (13b) the duration of  $A_{1,3}$  is set to 0 and a failure occurs due to the impossibility to satisfy constraint (10b).

To deal with this issue, we took advantage of the expressiveness of CP-Optimizer to add an alternative variant in our model: using the concept of interval instead of using variables *starts* and *durations*. An interval is defined as follows:  $\forall i \in \mathcal{A}$ ,  $interval[i]$  is a  $|maxpreemption|$  array of optional intervals variable. Each interval has the attributes *present* (boolean), *duration*, *start*, *end* (all integers). We also create one unique interval for each task, called spantask. New constraints are applied to this *interval* variable :

15.  $\forall i \in \mathcal{A}$ , *s.t*  $p_{i, modes[i]} \geq 1, \forall j \in \mathcal{J}, interval[i, j].duration \geq 1$  : We don't consider 0 duration for task intervals, which avoids the above-presented issue. This could also help the solver to remove unnecessary solutions in its search space since otherwise there could be multiple equivalent solutions.
16.  $\forall i \in \mathcal{A}, j \in \mathcal{J}^-, interval[i, j].start \geq interval[i, j - 1].end$ : precedence constraints between sub-intervals of the same task.
17.  $\forall i \in \mathcal{A}, j \in [1, maxpreemption - 1], interval[i, j].present = False \rightarrow interval[i, j + 1].present = False$ : When one sub-interval is not present, the remaining ones are not present either. This is equivalent to 13b constraint.
18.  $\forall i \in \mathcal{A}, \sum_{j \in \mathcal{J}} (interval[i, j].duration * interval[i, j].present) = p_{i, modes[i]}$  The sum of duration of present intervals should sum to the duration of the task.
19.  $\forall i \in \mathcal{A}, span(spantask[i], interval[i, :])$ : We use the native constraint *span* of CPOptimizer so that the spantask[i] spans over all present intervals in  $interval[i]$ , ignoring the non-present.

The precedence constraints are written using the spantask interval. We call this new model specific to CPOptimizer, Model *CP-SmartPreemption* while Model (1–14) is called *CP-Base*.

### 3.2.3 Objective function

We will focus exclusively on the makespan, which in terms of our formulation is equal to the ending time of the sink task  $n$ . Hence the goal is to minimize

$$makespan = starts[n, max_{preemption}] + durations[n, max_{preemption}]$$

However as detailed in section 4.3, due to the difficulty to obtain feasible solution, in our solution approach we opt in a first phase for a relaxation of a set of constraints and a penalization of their violation in the objective function. Thus the actual objective in this phase is a lexicographic optimization of the violation penalty and the makespan (see section 4.3).

### 3.3 A small PP-MS-MM-RCPSP/max-cal instance and its optimal solution

We provide an example PP-MS-MM-RCPSP/max-cal instance to illustrate its output from a simple problem definition contained in Table 1. It contains 6 activities to schedule (including source and sink tasks); it also includes multi-mode tasks, variable operator availability, release and deadlines, complete and partial preemption and finally one synchronisation constraint. This instance can be found in the toy model folder of our open source model's repository <sup>2</sup>. The optimal solution is depicted in Figure 1. We can observe that the tasks  $A_1$  and  $A_3$  are partially preemptive activities : the resource  $R_1$  is therefore still used even though the task is paused. The calendar constraint of Operator 1 is visible in the corresponding Gantt chart where we see no operations assigned to it during its break time. The mode allocation is the following :  $mode_{A_0} = 1, mode_{A_1} = 2, mode_{A_2} = 1, mode_{A_3} = 2, mode_{A_4} = 2, mode_{A_5} = 1$ .

■ **Table 1** An instance of PP-MS-MM-RCPSP/max-cal.

Activity	Mode	Duration	Skills	Resource	Deadline	Release	Preemption	Successors
$A_0$	1	0	-	-	-	-	-	$A_1, A_2, A_3, A_4, A_5$
$A_1$	1	5	$l_1$	$R_1$	-	-	$\mathcal{A}^{PP}$	$A_5$
	2	3	$l_1$	$R_1$	5	-	$\mathcal{A}^{PP}$	
$A_2$	1	1	$l_3, l_4$	$(R_1, 1)$	-	2	$\mathcal{A}^{NP}$	$A_3, A_5$
	2	2	$l_3$	$(R_1, 1)$	-	2	$\mathcal{A}^{NP}$	
$A_3$	1	3	$l_2$	$(R_1, 1)$	-	-	$\mathcal{A}^{PP}$	$A_5$
	2	2	$l_3, l_2$	$(R_1, 1)$	-	-	$\mathcal{A}^{PP}$	
$A_4$	1	2	$l_3$	-	-	5	$\mathcal{A}^P$	$A_5$
	2	3	-	$(R_1, 1)$	-	5	$\mathcal{A}^P$	
$A_5$	1	0	-	-	-	-	-	-

Operator	Skills	Calendar
$O_1$	$l_1, l_3$	[1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0]
$O_2$	$l_1, l_2, l_4$	[1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1]

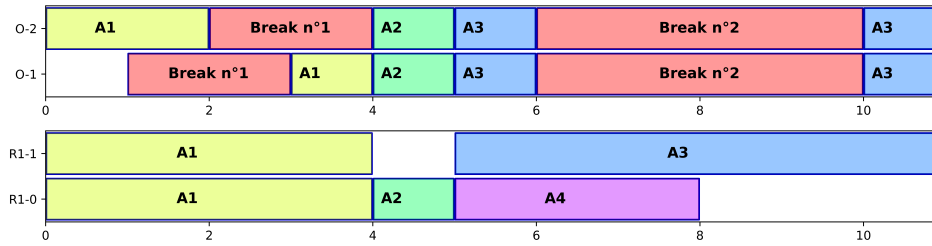
  

Resource	Capacity
$R_1$	2

$P_{sync-start}$	$[(A_3, A_4)]$
------------------	----------------

<sup>2</sup> [https://github.com/g-poveda/do\\_experiments](https://github.com/g-poveda/do_experiments)



■ **Figure 1** Operator and resource oriented Gantt chart for the example instance.

## 4 Algorithms

### 4.1 Generic large neighborhood search algorithm

Our approach to solve the PP-MS-MM-RCPSP/max-cal is to use a generalisation of Large Neighborhood Search (LNS) for scheduling problems [21]. In LNS, We iteratively improve the solution quality of a Master Problem  $\mathcal{MP}$  by solving at each step a Reduced Master Problem  $\mathcal{RMP}$ , based on the  $\mathcal{MP}$  and previously found solutions. The way  $\mathcal{RMP}$  are built are the core of LNS methods. The generic LNS algorithm is described in Algorithm 1, where  $X^{iter}$  denotes the solution at iteration  $iter$  and  $Y^{iter}$  its objective value.

■ **Algorithm 1** Generic Large Neighborhood Search Algorithm.

---

#### Begin

- 1:  $Y^* = \infty, X^* = None$
  - 2:  $(X^0, Y^0) = (X^*, Y^*) = initial_{solution}(P)$
  - 3:  $iter = 0$
  - 4: **repeat**
  - 5:  $\mathcal{RMP} = buildsubproblem(\mathcal{MP}, X^{iter})$
  - 6:  $X^{iter+1}, Y^{iter+1} = solve(\mathcal{RMP})$
  - 7: **if**  $Y^{iter+1} \leq Y^*$  **then**
  - 8:  $X^* \leftarrow X^{iter+1}$
  - 9:  $Y^* \leftarrow Y^{iter+1}$
  - 10:  $iter \leftarrow iter + 1$
  - 11: **until** stop criterion is met
  - 12: **return**  $X^*, Y^*$
- 

### 4.2 Application to the PP-MS-MM-RCPSP/max-cal

#### 4.2.1 Initial solution provider

We rely on a generalisation of the serial schedule generation scheme (SGS) procedure [16] to produce an initial solution for PP-MS-MM-RCPSP/max-cal (Algorithm 1, line 2). A similar generalisation was implemented in [26] being the closest one we found in the literature since it includes multi-skill, preemption and partially releasable resources.

Our *SGS* implementation takes as input a priority ordering of activities  $order_{act}$ , given as a permutation of  $\mathcal{A}$  and, for each activity  $i \in \mathcal{A}$ , another priority ordering  $order_{res_i} \in S_{|\mathcal{O}|}$  of the workers  $o \in \mathcal{O}$  to be assigned to the activity and the mode id  $modes_i$  chosen to run the activity, as defined for the CP model.

Activities are scheduled incrementally based on their priority as in classic serial SGS, at their earliest possible start date considering resource availability. Worker allocation is also done greedily using the *order\_res* array. The preemptivity feature of our problem allows us to schedule subpart of some activities. Our open source library includes an implementation module handling all the features of PP-MS-MM-RCPSP/max-cal except for the  $P_{sync-start/end}$  constraints, release and deadline constraints. The greedy procedure doesn't ensure that the constraints are feasible for these hard constraints.

Using the *SGS* procedure makes possible to run simple local search algorithms to get one initial solution of the problem. Such solver explores the vectored state space *order\_act*, *order\_res*, *modes*, and evaluate its fitness using *SGS*. The output evaluation of *SGS* is the same than the LNS or CP ones : makespan + potential penalty when deadlines and other constraints are not fulfilled (see details in Section 4.3). The state space being a permutation one, we are using moves chosen randomly from a portfolio of potential moves : partial and total shuffling, swap 2 random activities, 2-opt moves. Only one specific mutation has been also implemented to correct potential deadlines constraints (that put first the late task). In this paper we are using simulated annealing (SA) [14] as a local search solver. SA was parameterized with an initial temperature of  $T = 3$  and exponential cooling schedule of factor  $\alpha = 0.999$ . A restart strategy also ensures to roll back to the current best solution when we haven't improved the quality after  $N = 300$  iterations.

#### 4.2.2 Subproblem building

To build the reduced master problem, we rely on the constraint programming formulation and include additional constraints, which makes it simpler to solve. We decompose the subproblem building procedure (Algorithm 1, line 5) in two main methods. The first one, *activityselector* will split the set of activities  $\mathcal{A}$  into 2 disjunctive subsets  $\mathcal{A}_{sub}$  and  $\mathcal{A}_{\overline{sub}}$ . Different methods have been implemented and tested to select  $\mathcal{A}_{sub}$  and  $\mathcal{A}_{\overline{sub}}$  as described below:

1. Random selection : given  $f \in \mathbb{R}, 0 \leq f \leq 1$ ,  
 $\mathcal{A}_{sub} = \text{sample}(\mathcal{A}, \lfloor f \cdot |\mathcal{A}| \rfloor)$ , and  $\mathcal{A}_{\overline{sub}} = \mathcal{A} \setminus \mathcal{A}_{sub}$
2. Random selection and neighbors : consist in the previous selection, then add the predecessors of each task in  $\mathcal{A}_{sub}$ . Formally, given  $f \in \mathbb{R}, 0 \leq f \leq 1$ :  
 $\mathcal{A}_{sub,1} = \text{sample}(\mathcal{A}, \lfloor f \cdot |\mathcal{A}| \rfloor)$ , and  
 $\mathcal{A}_{sub} = \mathcal{A}_{sub,1} \cup \{j \in \mathcal{A} \text{ s.t. } \exists (x, y) \in P, x = j \wedge y \in \mathcal{A}_{sub,1}\}$
3. Cut in equal parts : given  $c \in \mathbb{N}$ , we sort the activities by increasing order of starting times in the current best solution  $X^*$ , then cut this ordered list in  $c$  equal pieces. When called, the function returns one of these  $c$  parts. A good strategy we observed consists in returning them in increasing order.
4. Generalized precedence adapted selection : when considering the constraints of deadlines, release dates, and the generalized precedence constraints ( $P_{sync-start}$ ,  $P_{startlag}$ ,  $P_{sync-end}$ ,  $P_{endlag}$ ), we can reach a solution  $X^*$  violating some of the constraints. Therefore we add in priority the tasks responsible for constraints violation in  $\mathcal{A}_{sub}$  and their predecessors in the precedence graph.
5. Mixing methods : given a pool containing the previous four methods, it will return  $\mathcal{A}_{sub}$  and  $\mathcal{A}_{\overline{sub}}$  of one of the pool members (either randomly at each iteration, or based greedily on the effectiveness of the selected method based on previous iterations).

Once we have the *activityselector* function, its output ( $\mathcal{A}_{sub}$  and  $\mathcal{A}_{\overline{sub}}$ ) can be used in the *constraintbuilder* function to add constraints to the *CP* model as described in Algorithm 2. Instead of totally locking the variables corresponding to  $\mathcal{A}_{\overline{sub}}$  and keeping free for optimisation

the variables of  $\mathcal{A}_{sub}$ , we consider different  $\epsilon$  values to constrain start and duration variables. We typically assign high values to parameters  $\epsilon_{sub,*}$  and small values to  $\overline{\epsilon}_{sub,*}$ .

■ **Algorithm 2** Constraint builder procedure.

---

```

1: constraintbuilder( $\mathcal{A}_{sub}, \overline{\mathcal{A}}_{sub}, X, model_{cp}, params_{constraints}$ ) :
2:  $\epsilon_{sub,+}, \epsilon_{sub,-}, \overline{\epsilon}_{sub,+}, \overline{\epsilon}_{sub,-},$ 
    $\overline{edur}_{sub,+}, \overline{edur}_{sub,-}, \overline{edur}_{sub,+}, \overline{edur}_{sub,-} = params_{constraints}$ 
3: for all  $i \in \mathcal{A}$  do
4:   for all  $j \in [1, max_{preemption}]$  do
5:     if  $X.durations[i, j] > 0$  then
6:       if  $i \in \mathcal{A}_{sub}$  then
7:          $(\epsilon_-, \epsilon_+) \leftarrow (\epsilon_{sub,-}, \epsilon_{sub,+})$ 
8:          $(\overline{edur}_-, \overline{edur}_+) \leftarrow (\overline{edur}_{sub,-}, \overline{edur}_{sub,+})$ 
9:       else if  $i \in \overline{\mathcal{A}}_{sub}$  then
10:         $(\epsilon_-, \epsilon_+) \leftarrow (\overline{\epsilon}_{sub,-}, \overline{\epsilon}_{sub,+})$ 
11:         $(\overline{edur}_-, \overline{edur}_+) \leftarrow (\overline{\overline{edur}}_{sub,-}, \overline{\overline{edur}}_{sub,+})$ 
12:         $model_{cp} \leftarrow X.starts[i, j] - \epsilon_- \leq model_{cp}.starts[i, j] \leq X.starts[i, j] + \epsilon_+$ 
13:         $model_{cp} \leftarrow \max(0, X.durations[i, j] - \overline{edur}_-) \leq model_{cp}.durations[i, j] \leq$ 
    $X.durations[i, j] + \overline{edur}_+$ 

```

---

### 4.2.3 Subproblem solving

To solve the subproblem we use the open source lazy clause generation solver Chuffed<sup>3</sup> or alternatively IBM's CP-Optimizer<sup>4</sup> backend. Both rely on a CP formulation done in Minizinc language, and worked well in our experiments. At each iteration of the solver, we set an upper time computation  $h$ . In case we want to optimize The objective function can be the final activity  $n$ 's end or the end of the subset of activities  $\mathcal{A}_{sub}$  we consider the most important in the  $\mathcal{RMP}$ . Both strategies impact slightly the convergence of the overall algorithm but we didn't assessed the details in our current experiments.

After retrieving the solution of the  $\mathcal{RMP}$  from the solver, it is post-processed by a left shifting procedure that compresses the full schedule. It is particularly useful in the case where the constraints introduced in *constraintbuilder* create idle times in the resulting schedule.

### 4.3 Relaxing constraints

The current *SGS* implementation does not return a feasible solution when certain constraints of PP-MS-MM-RCPSP/max-cal are present. Namely, those are the deadline and generalized precedence constraints  $P_{sync-start}, P_{sync-end}, P_{startlag}, P_{endlag}$ . By relaxing constraints (6), (7), (8), (9), (12) from Subsection 3.2.2 we include a violation penalty into the objective function. This violation penalty for  $P_{sync-end}$  is illustrated in Figure 2. The objective function that LNS and underlying CP solver minimizes is  $makespan + M \cdot penalty$  where  $M$  is a large number.

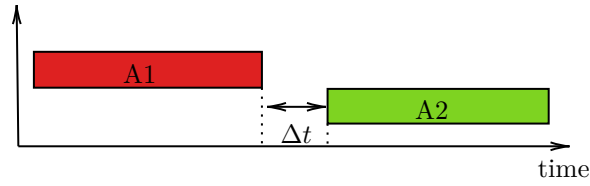
This means that in practice, there will be 2 phases of optimisation :

- Feasibility phase : In this phase the violation penalty decreases, converging to 0.
- Optimisation phase : the algorithm optimises the original objective function (*makespan*)

<sup>3</sup> <https://github.com/chuffed/chuffed>

<sup>4</sup> <https://www.ibm.com/analytics/cplex-cp-optimizer>





■ **Figure 2**  $\Delta t$  is added to the objective function whenever  $(A1, A2) \in P_{sync-end}$ .

## 5 Experimental results

### 5.1 Manufacturing domain instances

We ran a series of experiments in order to evaluate our LNS method for PP-MS-MM-RCPSp/max-cal. All experiments have been done in a MacBook Pro with 2,7 GHz Quad-Core Intel Core i7. In the first experiment, our goal is to assess the performance of our method in a practical situation, so we extracted data samples from a real manufacturing use case. The experiment relies on two scheduling problem base instances (A and B) obtained from one of the assembly plants at Airbus, and are described in Table 2. Using these 2 base instances, we built 32 different instances, using all the possible combinations of the following features :

- Calendar: we can use our definition for resource consumption (Constraint 1 in section 3.2) or consider complete resource availability at any moment, hence making vectors  $B_k : \forall k \in \mathcal{R}^p$  always constant and  $A_o : \forall o \in \mathcal{O}$  always 1.
- Temporal Constraints: we include or not the deadline times, release times (described in Section 3.2 as the constraints number 11 and 12) and generalized precedence constraints (constraints number 6 to 9)
- Preemptive : tasks can be preempted if needed or preemption is forbidden; in the non preemptive use case,  $\mathcal{A}^{NP} = \mathcal{A}$
- Multiskill : skilled workers are present or not; in the non multiskill use case, workers belong to  $\mathcal{R}$  and  $\mathcal{O} = \emptyset$

■ **Table 2** Base instance description of the manufacturing use case.

Instance	$ \mathcal{A} $	$ \mathcal{R} $	$ \mathcal{L} $	$ \mathcal{O} $	$ P $	$ P_{ss} $	$ P_{sc} $	$ P_{st} $	$ P_{el} $
A	291	8	23	27	842	3	70	0	4
B	199	3	6	6	676	1	34	6	4

We ran our LNS algorithm with a time limit of 1800 seconds for all of the instances. As a comparison, we also ran the simulated annealing algorithm (SA) and a direct CP solving of Model *CP-Base* using Minizinc and the CP-Optimizer backend, the Chuffed backend and the *CP-SmartPreemption* model for the same amount of time.

The results for each instance are detailed in Table 3. Four of those instances (number 8 to 11) are infeasible by design, because they include the calendar constraints but not the preemption ones, so we left out those and will focus on the remaining 28 instances for our experiments. For the sake of clarity, we only show *CP-SmartPreemption* results (CP-S in the table) as it is consistently better than the *CP-Base* model whether the latter is solved by Chuffed or CP-Optimizer. A comparison between *CP-SmartPreemption* and *CP-Base* can be found in Appendix A.1.

■ **Table 3** Makespan results obtained for the 28 testing instances (splitted in two tables) using the three solvers: LNS, *CP-SmartPreemption* (abbreviated as CP-S) and SA. For SA penalty values are included when the solution is not satisfying all the constraints (solutions for LNS and *CP-SmartPreemption* got no penalty). The table also includes each instance base problem (A or B) and its features: **M** for Multi skill, **G** for Generalized precedence constraints, **P** for (partial) Preemption and **C** for resource Calendars. In **bold** best results among the three algorithms.

ID	Features	LNS	CP-S	SA	p-SA	ID	Features	LNS	CP-S	SA	p-SA
0	A	<b>1080</b>	<b>1080</b>	<b>1080</b>	0	18	B G	<b>3053</b>	<b>3053</b>	3275	2171
1	A M	<b>1080</b>	<b>1080</b>	<b>1080</b>	0	19	B G M	<b>3109</b>	<b>3109</b>	3530	2590
2	A G	<b>6534</b>	<b>6534</b>	6817	128	20	B P	2340	2340	<b>2339</b>	0
3	A G M	6553	<b>6534</b>	6543	104	21	B M P	<b>2340</b>	2390	<b>2340</b>	0
4	A P	<b>1080</b>	<b>1080</b>	<b>1080</b>	0	22	B G P	3053	<b>3044</b>	3148	750
5	A M P	<b>1080</b>	2495	<b>1080</b>	0	23	B G M P	<b>3117</b>	3180	3456	714
6	A G P	<b>6534</b>	<b>6534</b>	<b>6534</b>	0	24	B C	3202	<b>3196</b>	3250	0
7	A G M P	6553	6612	<b>6537</b>	0	25	B C M	3289	<b>3208</b>	3263	0
12	A C P	<b>7154</b>	-	<b>7154</b>	0	26	B C G	<b>4949</b>	<b>4949</b>	5096	4727
13	A C M	<b>7154</b>	-	<b>7154</b>	0	27	B C G M	5096	<b>4949</b>	5096	4359
14	A C G P	7866	-	<b>7777</b>	0	28	B C P	3205	3195	<b>3192</b>	0
15	A C G M P	<b>7777</b>	-	7805	0	29	B C M P	3193	3208	<b>3192</b>	0
16	B	2340	<b>2339</b>	2373	0	30	B C G P	4580	<b>4383</b>	4436	1167
17	B M	2350	<b>2346</b>	2375	0	31	B C G M P	<b>5010</b>	-	4840	1132

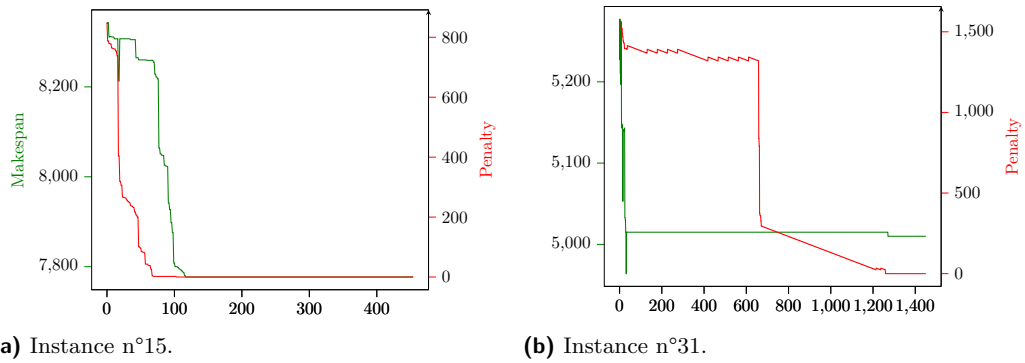
As we can see, LNS reaches the best results on 15 out of the 28 instances, *CP-SmartPreemption* on 16 and SA on 13. By comparing the different instances where each method excels, we observe that instances with high complexity in the type of constraints, i.e a combination of having generalized precedence constraints, calendar, multiskill or preemptive tasks are the ones for which we would rely the most on LNS: instance 15 and 31 are indeed the ones including all the available features. For those instances, *CP-SmartPreemption* is not returning any result in the allowed time whereas SA fails to find feasible solutions.

On the other hand, SA is generally competitive as long as there is no generalized precedence constraint involved. But if those are present it finds difficulty in yielding a solution, which was expected since the *SGS* does not fulfill all the generalized precedence constraints. However, in some of the instances SA got better results with some of those features present, thus we theorize that the solution space heavily impacts in this method, helping to overcome the presence of complex constraints and obtaining nonetheless an optimal solution: if we do not take into account instances where no feasible solution was found, SA achieves best results in 13 of 20 instances.

Figures 3a and 3b show makespan and violation penalty from the solutions found using our method for the most complex two variants (containing all the possible features). We can confirm that our strategy of relaxing certain constraints at the cost of adding a penalty is useful to get an initial solution that will be improved from that point, as described in Subsection 4.3.

## 5.2 Experiments on instances with multiskill and partial preemption

We conducted a second experiment using instances of partial preemptive multi-skill problems based on the schedule of operations of a nuclear facility [26]. The benchmark is divided in 4 different sets, each having different distribution combinations of  $\mathcal{A}^P$ ,  $\mathcal{A}^{NP}$  and  $\mathcal{A}^{PP}$ .



■ **Figure 3** Evolution of solutions found for LNS algorithm in two of the instances. Makespan shows in green at the left Y axis and penalty in red at the right Y axis for both of the instances.

■ **Table 4** Average gap to lower bound for different algorithms and by set of instances. Best results for each set are bold and second best are underlined.

Algorithm	Set A	Set B	Set C	Set D
GRASP + LNS [26]	<b>1.56%</b>	<b>1.79%</b>	6.68%	<b>2.06%</b>
SA (300 seconds)	<u>1.59%</u>	<u>1.82%</u>	<b>5.65%</b>	2.33%
SA (240 s)	1.82%	1.83%	6.57%	2.17%
SA (240 s)+LNS (60 s)	1.82%	1.83%	<u>6.49%</u>	<u>2.12%</u>

The multi-skill variant of this benchmark can be directly mapped into our formalisation, except for the minimum number of operators required by activity constraints. To adapt our model we introduced in the problem instances a dummy skill  $s_{all}$  mastered by all operators. We then set skill requirement of skill  $s_{all}$  for each activity, to the minimum number of operators required.

The comparison results, detailed in Table 4, are based on the average gap to lower bound, like it was done in the original paper. We compared three of our solving methods to the one tested previously. The first one is obtained by running simulated annealing for 300 seconds, working on the permutation of tasks and the SGS algorithm. In the second we ran simulated annealing alone for 240 seconds as second baseline, and finally we ran *LNS* during 60 seconds. The SGS method combined with SA is competitive with the best dedicated algorithm GRASP+LNS [26]. LNS only improves slightly SA results as it can be seen when comparing SA (240 s) and SA (240 s) + LNS (60 s), which may be explained by the performance of SA itself, which is close or even better than the baseline.

### 5.3 Experimental results of LNS on standard multi-skill instances from the literature

Additionally, we did another benchmark comparison using the instances contained in [38]. The goal of running this second experiment is to have again comparative results to the work of [26], which our method can be related with, since it also aims at handling multi-skill and partial preemption in scheduling problems. The mentioned instances are divided in five sets, but we decided only to tests in one of them, set 1B, since it is the set with the lowest solution rates (only 12.5% solved) using the reference solvers.

Set 1B was proposed by [2] and it contains instances with 42 activities, 4 skills and between 20 to 60 resources. For these experiments, we relied on the CP model used in [38]<sup>5</sup>, which models classical multi-skill where operator perform at most 1 skill on a task. We used this CP model in our generic Large Neighborhood Search algorithm to evaluate how well the approach generalizes to different scheduling problem variants. Baseline solutions we compare with are the ones found by running Chuffed solver with the best search strategy. They can also be found in the repository.

The current implementation of *SGS* prevents us to test simulated annealing as a solver or initial solution provider, because of the constraint stating that an allocated worker can at most perform 1 skill on a task. Therefore, the initial solution used in *LNS* is computed by using the Chuffed solver directly on the minizinc model from [38] for 30 seconds. Then, if the solution is not optimal, the large neighborhood search is used with the mixing neighborhood strategy for a maximum time of 500 s. Worker allocation is fixed only in 10% of the tasks, which empirically showed good performance in terms of improvement rate during the *LNS* algorithm. We are improving 151 out of 216 of the Set 1B instances (69.9% of the total) whereas Young *et al.* results [38] are still better on 29 instances (13.4%), and equal on 36 instances (17.7%). On the 151 improved instances, the average improvement of our solution over the baseline is 4.61%. On the 29 that Young *et al.* CP method [38] was better, the makespan was worsen in average by 3.46%. In total average, our method gave a 2.76% improvement on the makespan.

Finally, we conducted some preliminary experiments on the new MSLIB [34] benchmark. The authors provided us some baseline best known solutions found by a genetic algorithm (GA) approach [33] on the hardest set of instances MSLIB4. Precisely we ran experiments on the first 404 instances of MSLIB4 using CP-Chuffed 30 seconds as initial solution followed by *LNS* for 100s. Our computation time is an order of magnitude higher than the GA that has a CPU time of around 15s.

The results indicate a strong impact of the *SS* (skill strength) parameter of the instances. This ratio roughly gives the scarceness of the skills in the sense that  $SS = 0$  means that the number of resources that master a skill is equal to the maximum demand in operators mastering this skills over the activities, i.e. the minimal value that ensure feasibility. *LNS* systematically outperforms GA when  $SS > 0$  (90 instances) with an average improvement of 2.9% but when  $SS = 0$  (164 instances), *LNS* is largely outperformed by GA with an average 19.2% gap. Additional material on these results are presented in Appendix A.3.

## 5.4 Implementation

We provide an open repository containing scripts to rerun the benchmarks presented in Sections 5.2 and 5.3 to ease replication for further research<sup>6</sup>. Every multiskill variant presented in the paper is using the same python object placeholder with different attribute values. About the algorithm most of it is reused by all variants :

1. The *SGS* procedure is the same whatever for all benchmark and therefore the metaheuristics methods that only rely on the *SGS* procedure are reused identically
2. *LNS* algorithm 1 is the same for all benchmarks
3. Activity selectors mixing method described in 4.2 are used on all benchmarks. Experiments that justify the choice of the mixing method are presented in Appendix A.2.

<sup>5</sup> <https://github.com/youngkd/MSPSP-InstLib/blob/master/models/mspsp.mzn>

<sup>6</sup> [https://github.com/g-poveda/do\\_experiments](https://github.com/g-poveda/do_experiments)

4. Specific : We are using 2 different minizinc modeling in our benchmark, one containing the most property of PP-MS-MM-RCPSP/max-cal that is used in benchmark 5.1 and 5.2 and one another model for classical MSRCPS for benchmark 5.3.

## 6 Conclusions

In this paper we have presented a large neighborhood search (LNS) method to solve partially preemptive multi-skill/mode resource-constrained project scheduling problem with generalized precedence relations and resource calendars (PP-MS-MM-RCPSP/ max-cal). Solvers including all of those features are scarce, but are usually needed to approach real manufacturing or assembly environments.

In order to validate our method, we performed three experiments to benchmark it against constraint programming and simulated annealing. In the first experiment we used data from an Airbus use case, containing a real world scenario with 32 different variations. To make a fair comparison, we designed the *CP-SmartPreemption* model that handles preemption in a clever way significantly outperforming the our baseline CP model *CP-Base*. Despite its good performances, *CP-SmartPreemption* is unable to find a feasible solution in 6 industrial instances, while the LNS method finds a solution on all instances. From the observations of the samples, we conjecture that our method works better when (almost) all complex features are present in the problem definition (i.e.: calendar, generalized precedence constraints, multiskill and preemption). Then, we tested our method using a benchmark from research work on partially preemptive multi-skill scheduling [26] to evaluate its performance in another real scenario. In this experiment, the simulated annealing methods appears competitive, slightly improved by LNS. Our method was the second best among the tested ones given the benchmark conditions, still being competitive enough to be close to the best one of that benchmark. In the last experiment on standard MS-RCPSP instances, we used our method as a mean of improvement for the solutions given by the CP based method. On the instances from [38] we improved the best known solutions in 69,9% of the instances. On the instances from [34] the performance of our approach is highly correlated with skill scarceness, which open a path for future research.

In conclusion, we found our LNS based method, available in a new discrete optimization open-source library, appropriate to solve scheduling problems containing combinations of complex features like the ones found in industrial instances, and is still reliable to be used for more academic problems.

An interesting perspective is to be able to reuse the subproblem solving information from one iteration the other to save computational time. Current Minizinc implementation has a limitation w.r.t. incrementality. Nevertheless, it would be worth to investigate how solution-based phase saving approaches [8] use nogoods and see if it is applicable in our LNS framework for future work. Adapting propagation guided LNS[23] to our framework is also a promising research direction.

---

## References

- 1 Behrouz Afshar-Nadjafi. Multi-skilling in scheduling problems: A review on models, methods and applications. *Computers & Industrial Engineering*, 151:107004, 2021.
- 2 Bernardo F Almeida, Isabel Correia, and Francisco Saldanha-da Gama. Priority-based heuristics for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications*, 57:91–103, 2016.

- 3 Bernardo F Almeida, Isabel Correia, and Francisco Saldanha-da Gama. Modeling frameworks for the multi-skill resource-constrained project scheduling problem: a theoretical and empirical comparison. *International Transactions in Operational Research*, 26(3):946–967, 2019.
- 4 Lucio Bianco and Massimiliano Caramia. An exact algorithm to minimize the makespan in project scheduling with scarce resources and generalized precedence relations. *European Journal of Operational Research*, 219(1):73–85, 2012.
- 5 P. Brucker. *Complex Scheduling*. Springer, 1999.
- 6 José Coelho and Mario Vanhoucke. Multi-mode resource-constrained project scheduling using rcpsp and sat solvers. *European Journal of Operational Research*, 213(1):73–82, 2011.
- 7 Erik L Demeulemeester and Willy S Herroelen. An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. *European Journal of Operational Research*, 90(2):334–348, 1996.
- 8 Emir Demirović, Geoffrey Chu, and Peter J Stuckey. Solution-based phase saving for cp: A value-selection heuristic to simulate local search behavior in complete solvers. In *Principles and Practice of Constraint Programming: 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings 24*, pages 99–108. Springer, 2018.
- 9 Andreas Drexl and Juergen Gruenewald. Nonpreemptive multi-mode resource-constrained project scheduling. *IIE transactions*, 25(5):74–81, 1993.
- 10 Davide Giglio, Massimo Paolucci, Abdolreza Roshani, and Flavio Tonelli. Multi-manned assembly line balancing problem with skilled workers: a new mathematical formulation. *IFAC-PapersOnLine*, 50(1):1211–1216, 2017.
- 11 Daniel Godard, Philippe Laborie, and Wim Nuijten. Randomized large neighborhood search for cumulative scheduling. In *ICAPS*, volume 5, pages 81–89, 2005.
- 12 Shima Javanmard, Behrouz Afshar-Nadjafi, and Seyed Taghi Akhavan Niaki. Preemptive multi-skilled resource investment project scheduling problem: Mathematical modelling and solution approaches. *Computers & Chemical Engineering*, 96:55–68, 2017.
- 13 Ahmed Karam, El-Awady Attia, and Philippe Duquenne. A milp model for an integrated project scheduling and multi-skilled workforce allocation with flexible working hours. *IFAC-PapersOnLine*, 50(1):13964–13969, 2017.
- 14 S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi:10.1126/science.220.4598.671.
- 15 RAINER KOLISCH and ANDREAS DREXL. Local search for nonpreemptive multi-mode resource-constrained project scheduling. *IIE Transactions*, 29(11):987–999, 1997. doi:10.1080/07408179708966417.
- 16 Rainer Kolisch and Sönke Hartmann. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In *Project scheduling*, pages 147–178. Springer, 1999.
- 17 Stefan Kreter, Andreas Schutt, and Peter J Stuckey. Using constraint programming for solving rcpsp/max-cal. *Constraints*, 22(3):432–462, 2017.
- 18 Philippe Laborie. An update on the comparison of mip, cp and hybrid approaches for mixed resource allocation and scheduling. In Willem-Jan van Hoeve, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 403–411, Cham, 2018. Springer International Publishing.
- 19 Haitao Li and Keith Womer. Scheduling projects with multi-skilled personnel by a hybrid milp/cp benders decomposition algorithm. *Journal of Scheduling*, 12(3):281–298, 2009.
- 20 Paweł B. Myszkowski, Maciej Laszczyk, Ivan Nikulin, and Marek Skowronski. imopse: a library for bicriteria optimization in multi-skill resource-constrained project scheduling problem. *Soft Computing*, 23:3397–3410, 2019.
- 21 Mireille Palpant, Christian Artigues, and Philippe Michelon. Lssper: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1):237–257, 2004.

- 22 Claude Le Pape and Philippe Baptiste. Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem. *Journal of Heuristics*, 5(3):305–325, 1999.
- 23 Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In *International Conference on Principles and Practice of Constraint Programming*, pages 468–481. Springer, 2004.
- 24 Oliver Polo-Mejía, Marie-Christine Anselmet, Christian Artigues, and Pierre Lopez. Mixed-integer and constraint programming formulations for a multi-skill project scheduling problem with partial preemption. In *12th International Conference on Modelling, Optimization and Simulation (MOSIM 2018)*, pages 367–374, 2018.
- 25 Oliver Polo-Mejía, Christian Artigues, Pierre Lopez, and Virginie Basini. Mixed-integer/linear and constraint programming approaches for activity scheduling in a nuclear research facility. *International Journal of Production Research*, 58(23):7149–7166, 2020.
- 26 Oliver Polo-Mejía, Christian Artigues, Pierre Lopez, Lars Mönch, and Virginie Basini. Heuristic and metaheuristic methods for the multi-skill project scheduling problem with partial preemption. *International Transactions in Operational Research*, 2021.
- 27 Sacramento Quintanilla, Ángeles Pérez, Pilar Lino, and Vicente Valls. Time and work generalised precedence relationships in project scheduling with pre-emption: An application to the management of service centres. *European Journal of Operational Research*, 219(1):59–72, 2012.
- 28 Jerzy Rosłon. The multi-mode, resource-constrained project scheduling problem in construction: state of art review and research challenges problem. *Technical Transactions*, 2017. doi:10.4467/2353737XCT.17.070.6427.
- 29 Mónica A Santos and Anabela P Tereso. On the multi-mode, multi-skill resource constrained project scheduling problem—a software application. In *Soft computing in industrial applications*, pages 239–248. Springer, 2011.
- 30 Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, 2011.
- 31 Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Solving rcpsp/max by lazy clause generation. *J. Sched.*, 16(3):273–289, 2013. doi:10.1007/s10951-012-0285-x.
- 32 Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Solving rcpsp/max by lazy clause generation. *Journal of scheduling*, 16(3):273–289, 2013.
- 33 Jakob Snauwaert and Mario Vanhoucke. A new algorithm for resource-constrained project scheduling with breadth and depth of skills. *European Journal of Operational Research*, 292(1):43–59, 2021. doi:10.1016/j.ejor.2020.10.032.
- 34 Jakob Snauwaert and Mario Vanhoucke. A classification and new benchmark instances for the multi-skilled resource-constrained project scheduling problem. *European Journal of Operational Research*, 2022. doi:10.1016/j.ejor.2022.05.049.
- 35 Tianshu Song, Ke Xu, Jiangneng Li, Yiming Li, and Yongxin Tong. Multi-skill aware task assignment in real-time spatial crowdsourcing. *GeoInformatica*, 24(1):153–173, 2020.
- 36 Ria Szeredi and Andreas Schutt. Modelling and solving multi-mode resource-constrained project scheduling. In *International Conference on Principles and Practice of Constraint Programming*, pages 483–492. Springer, 2016.
- 37 Tony Wauters, Joris Kinable, Pieter Smet, Wim Vancroonenburg, Greet Vanden Berghe, and Jannes Verstichel. The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling*, 19(3):271–283, 2016.
- 38 Kenneth Young, Thibaut Feydy, and Andreas Schutt. Constraint programming applied to the multi-skill project scheduling problem. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, pages 308–317, August 2017. doi:10.1007/978-3-319-66158-2\_20.

## A Appendix

### A.1 CP modeling comparisons

We ran new experiments in comparing the two different CP modeling we propose : the default one presented in section 3.2.2 and its upgrade that we called *CP-SmartPreemption*. Computation time are still 1800 s each like in experiments of section 5.1. Results are shown in 4. We interestingly note that *CP-SmartPreemption* is both faster and more efficient than

■ **Table 5** Best solution and time to best solution for CP-Base and *CP-SmartPreemption*.

ID	Features	CP-B	CP-B (s)	CPSmart	CPSmart (s)	ID	Features	CP-B	CP-B (s)	CPSmart	CPSmart (s)
0	A	1080	1.09	1080	0.78	18	B G	3053	0.59	3053	0.20
1	A M	-	-	1080	8.20	19	B G M	3109	81	3109	3.20
2	A G	6534	0.99	6534	0.23	20	B P	2342	133	2340	178
3	A G M	-	-	6534	3.90	21	B M P	-	-	2390	196
4	A P	1104	2.21	1080	64.9	22	B G P	3063	106	3044	30.0
5	A M P	-	-	2495	24.4	23	B G M P	-	-	3180	198
6	A G P	6534	2.40	6534	62.3	24	B C	3196	154	3196	170
7	A G M P	-	-	6612	141	25	B C M	3272	127	3208	88
12	A C P	-	-	-	-	26	B C G	4949	0.52	4949	0.17
13	A C M	-	-	-	-	27	B C G M	4949	2.2	4949	0.80
14	A C G P	-	-	-	-	28	B C P	3252	1.32	3195	217
15	A C G M P	-	-	-	-	29	B C M P	-	-	3208	203
16	B	2339	106	2339	22.9	30	B C G P	4949	1.35	4383	59.0
17	B M	2359	193	2346	158	31	B C G M P	-	-	-	-

the default version. Notably, it manages to find feasible solution to 7 additional instances compared to basic CP formulation. It also succeeds in solving instance ID 30 way more efficiently than CP-Base or even the LNS that was presented in Table 3. As of today, the current modeling of *CP-SmartPreemption* still fails at the most complicated instances on the instance A that have the most operators/workers and number of task.

### A.2 Subproblem methods benchmark

We give more hindsight on the general choice done in the paper to use the mixing methods described in section 4.2.2. Naturally we expect that such a portfolio approach that picks a method from a pool of  $N$  neighborhood methods can be better in average than using only one method. To confirm this intuition, we run LNS methods using different parameters:

1. “Random selection” methods with parameter  $f$  taking values in  $[0.1, 0.2, 0.3, 0.4]$
2. “Cut in equal parts” methods with parameter  $c$  taking values in  $[2, 3, 4, 5, 6]$
3. “Mixing method” : portfolio of the previous 9 methods, chosen randomly at each iteration of LNS.

As a starting experiment, we are testing this methods on 5 classical RCPSP instances with 120 tasks, and run the solver 10 times per neighborhood methods and instances to get a better statistical confidence. We limited to 100 the number of iteration of LNS. Detailed results are in Table 6. It shows empirical confidence in using a mixing method, that achieved best performance. We also ran the same experiment on the instances ID=16,17 in Table 7 and got similar behaviour. Deeper hyper-parameters optimisation could be done for the more complex problems as PP-MS-MM-RCPSP/max-cal in a further research.



■ **Table 6** Neighborhood methods performance on a few instances.

Method	j1201_1	j1201_2	j1201_3	j1201_4	j1201_5
RS(0.1)	116.6	131.3	138.6	106.5	131.3
RS(0.2)	112.2	129.8	136.8	106.0	128.3
RS(0.3)	110.4	128.4	134.3	105.4	120.4
RS(0.4)	108.9	118.0	132.6	103.3	116.7
Cut(2)	<b>105</b>	<u>118.0</u>	<u>130</u>	<u>102.0</u>	<b>113.0</b>
Cut(3)	107	118.0	135	<b>101.0</b>	118.0
Cut(4)	108	126.0	131	106.0	132.0
Cut(5)	111	128.0	136	106.0	130.0
Cut(6)	111	129.0	137	106.0	130.0
Mixing	<u>106.5</u>	<b>115.0</b>	<b>128.3</b>	<u>102.0</u>	<b>113.0</b>

■ **Table 7** Neighborhood methods performance on an industrial use case (ID=16).

Method	ID 16 (B)	ID 17 (BM)
RS(0.1)	2698.0	2745.0
RS(0.2)	2659.0	2675.0
RS(0.3)	2534.0	2635.0
RS(0.4)	2489.0	2662.0
Cut(2)	2344.0	2628.0
Cut(3)	<u>2343.0</u>	<b>2436.0</b>
Cut(4)	2346.0	2480.0
Cut(5)	2346.0	2486.0
Cut(6)	2349.0	2776.0
Mixing	<b>2342</b>	<u>2460.0</u>

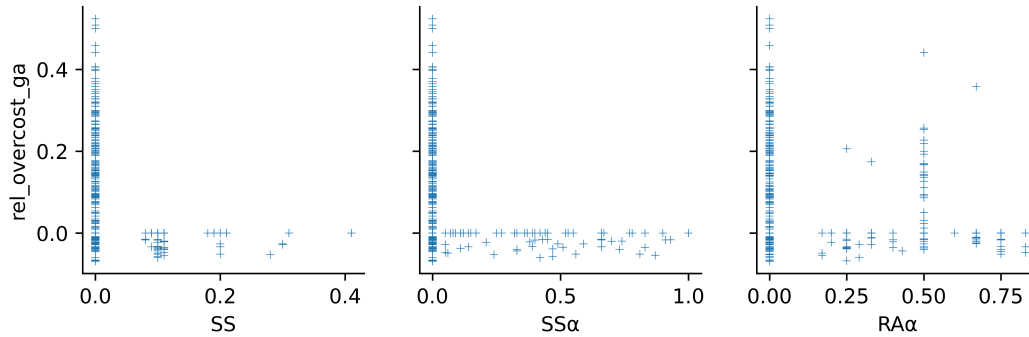
### A.3 MSLIB Experiments

To better understand the results we plot the overcost of the *LNS* method as a function of different parameters used to generate MSLIB4 instances in Figure 4. We theorize that increasing  $\#R$  (number of resources) has a negative impact on *LNS* performance. It also highlights that *LNS* is underperforming only on instances where  $SS = 0$  and  $SS\alpha = 0$ , defined as skill strength and skill strength variability. As future work we want to continue testing on MSLIB benchmark to possibly improve the *LNS* method. Detailed results comparing the GA and LNS approach over the tested instances are depicted in Tables 8 and 9.

■ **Table 8** Best results over subset of MSLIB4 instances (404 instance).

Algorithms	LNS (ours)	GA 2021	Equal
#Best Results	90	164	150
Mean Improvement when better solution than GA	2.9%	-	-
Mean Degradation when worse solution than GA	19.2%	-	-

■ **Figure 4** Relative overcost to GA of LNS method, function on different features of MSLIB4 instances.



■ **Table 9** Detailed results comparing GA (from [33]) and our methods, function on instance parameters. The GA, Equal, LNS columns counts the number of instance of some given parameters where each of the algorithms got the best results. (LNS-GA)/GA column store the average overcost of LNS method over the subset of instances described by SP, SS, RA.

SP	SS	RA	NbRun	GA	Equal	LNS	Mean (LNS-GA)/GA (%)
0.1	0.0	0.0	79	41	34	4	12.4
-	-	0.2	40	5	17	18	1.89
-	-	0.3	33	33	0	0	18.6
-	-	0.4	25	9	9	7	5.59
-	-	0.5	66	44	11	11	12.4
-	-	0.6	15	5	7	3	2.86
-	-	0.7	18	11	5	2	9.21
-	-	0.8	22	13	7	2	8.04
-	-	0.9	3	3	0	0	6.78
-	-	1.0	28	0	18	10	-1.01
-	0.1	0.0	24	0	14	10	-1.06
-	-	0.2	6	0	2	4	-2.71
-	-	0.3	3	0	1	2	-3.60
-	-	0.4	1	0	0	1	-4.88
-	-	0.5	1	0	0	1	-1.67
-	-	0.6	1	0	1	0	0.0
-	-	0.7	2	0	1	1	-0.76
-	0.2	0.0	18	0	12	6	-1.00
-	-	0.2	6	0	2	4	-2.67
-	-	0.3	1	0	1	0	0.0
-	-	0.4	2	0	1	1	-2.38
-	0.3	0.0	5	0	3	2	-1.61
-	-	0.2	1	0	0	1	-2.63
-	0.4	0.0	2	0	2	0	0.0
-	0.5	-	2	0	2	0	0.0



# Assembly Line Preliminary Design Optimization for an Aircraft

Stéphanie Roussel<sup>1</sup> ✉ 

ONERA, ONERA DTIS, Toulouse, Université de Toulouse, France

Thomas Polacsek ✉ 

ONERA, ONERA DTIS, Toulouse, Université de Toulouse, France

Anouck Chan ✉ 

ONERA, ONERA DTIS, Toulouse, Université de Toulouse, France

---

## Abstract

In the aeronautics industry, each aircraft family has a dedicated manufacturing system. This system is classically designed once the aircraft design is completely finished, which might lead to poor performance. To mitigate this issue, a strategy is to take into account the production system as early as possible in the aircraft design process. In this work, we define the Assembly Line Preliminary Design Problem, which consists in defining, for a given aircraft design, the best assembly line layout and the type and number of machines equipping each workstation. We propose a Constraint Programming encoding for that problem, along with an algorithm based on epsilon constraint for exploring the set of Pareto solutions. We present experiments run on a set of real industrial data. The results show that the approach is promising and offers support to experts in order to compare aircraft designs with each other.

**2012 ACM Subject Classification** Applied computing → Computer-aided manufacturing

**Keywords and phrases** Assembly line design, Constraint Programming, Multi-objective, Industry 4.0

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.32

**Supplementary Material** *Dataset (Assembly Line Problems)*: <https://github.com/stephroussel/assemblyLine>

## 1 Introduction

The aeronautics industry is highly specialized. It requires significant investments in research to design and manufacture a new aircraft. Each new aircraft family, consisting of several models within a given size range, requires the development of a dedicated industrial system. The latter includes the manufacturing and assembly processes, supply chain, tooling and facilities necessary to produce an aircraft at scale. The design of such a system typically involves several steps of high-level objectives refinements, which can take several years and requires collaboration between various stakeholders (aircraft manufacturers, suppliers, and regulatory agencies, etc.). Once established, the system can be used for the entire aircraft family lifecycle, which can last several decades. While aircraft manufacturers can generally leverage existing resources such as buildings and infrastructure, they have to develop new manufacturing and assembly lines to meet the specific requirements of each aircraft family.

An industrial system has its own set of requirements and is designed so as to optimize several criteria. For instance, assembly lines must comply to organizational constraints and ergonomics recommendations while minimizing the overall investment cost. As the manufacturing system depends highly on the aircraft choices (*e.g.* the aircraft material such as carbon fibers, metal, etc.), the manufacturing system design is generally thought after the

---

<sup>1</sup> Corresponding author



aircraft design. This might result in a low performance of the latter or even the impossibility to meet some requirements. For instance, choosing carbon fibers on the aircraft side requires buying autoclaves on the industrial system side, with an associated cost that exceeds a given investment budget.

Therefore, a nowadays trend in aeronautics is to take into account the industrial system design as early as possible in the aircraft design process ([18, 22]). A way to do so, and the approach we follow in this paper, is to design the best preliminary industrial system for an aircraft design that is not validated yet and assess its performance. This allows to estimate whether expected industrial objectives can be met for a given aircraft. Moreover, if several aircraft designs are candidates, this also allows comparison between them with respect to their industrial suitability. Finally, such an approach offers support for making trade-offs between the aircraft and the manufacturing. More precisely, a modification of the aircraft that downgrades its performance might allow to decrease significantly the cost associated to the production system. For instance, a change in the aircraft shape might increase a little bit its fuel consumption but reduce significantly the time required to build it because the new shape allows to use robots for making junctions.

This work presents contributions produced in the context of an industrial project with an aircraft manufacturer for supporting trade-offs between an aircraft and its industrial system. The objective of this project was to focus on one element of the industrial system, namely the *assembly line*. An aircraft assembly line is responsible for bringing together all of the components of the aircraft, including the fuselage, wings, engines and interiors, into a complete and fully functional aircraft. The assembly lines we consider are pulsed and composed of several workstations that are equipped with specific machines. Assembly lines must be designed to be flexible enough to accommodate changes in the manufacturing process as the aircraft family evolves over time. Moreover, because of the international context, production rates can vary a lot within the aircraft lifecycle. It is therefore essential to choose an aircraft design and an assembly line design that are compatible with such evolution. Other elements of the industrial system, *e.g.* the supply chain, depend also highly from the aircraft design but they were out of scope of our project. Nevertheless, the assembly line by itself represents a quite expensive part of the industrial system. In fact, when an aircraft design allows to have one workstation (or one machine) less than another one, it can correspond to costs of several million euros.

In the preliminary stages of designing an assembly line, aircraft manufacturers can use the PERT (Program Evaluation and Review Technique). PERT is a network diagram that shows the sequence of tasks and their dependencies. It allows to identify critical paths and potential bottlenecks. However, PERT is a very optimistic view of the building process. In fact, because it does not take resources into account, it assumes that all tasks that are not dependent can be performed in parallel. In order to have a more realistic assessment on the assembly line, industrial architects design workstations and equip them, assign tasks to workstations and then compute a fine-grain line balancing that takes into account workers and their profiles. However, such an iterative process can be quite heavy to put in place for each aircraft design candidate.

In this paper, we present an aid-decision tool that we have developed in the context of the previously mentioned project. For a given aircraft, the tool returns the best preliminary assembly line designs with respect to several criteria. These criteria deal with the assembly line layout (workstations and associated equipment), its cost and also with the production duration. We call this problem the Assembly Line Preliminary Design Problem, which can be seen as a Resource-Constrained Project Scheduling Problem (RCPSP) with additional

constraints ([9]). The tool relies on a Constraint Programming encoding of the problem. In fact, Constraint Programming is particularly suited for handling resource and temporal constraints inherent in the assembly line design problem ([16]). In order to explore the Pareto solutions set, we define an epsilon constraint based algorithm ([4]). It basically consists in computing bounds for each criteria, choosing an exploration order and a number of points in the front, and next running a mono-criterion algorithm to generate Pareto solutions.

Our contribution can be summarized as follows:

- we present a new problem, namely the Assembly Line Preliminary Design Problem, and formalize associated constraints and criteria;
- we define a Constraint Programming encoding for this problem;
- we develop an epsilon constraint based algorithmic approach for exploring the Pareto front;
- we present experiments on a set of real data coming from an aircraft manufacturer;
- we make those data available for the community.

The paper is structured as follows. In Section 2, we describe related works. Section 3 is dedicated to an informal description of the problem and some prior work for eliciting constraints and criteria in the case of a preliminary design. In Section 4, we formally model the Assembly Line Preliminary Design Problem and propose an associated Constraint Programming encoding. We detail the algorithmic approach for exploring the Pareto solutions set in Section 5. In Section 6, we describe the real world benchmarks we have used and discuss the results obtained with the paper's approach. Finally, we conclude on future works in Section 7.

## 2 Related Works

A vast body of literature exists on the production and line balancing problems research topic ([5, 8]). The *Simple Assembly Line Balancing Problem* (SALBP) consists in assigning tasks to workstations on a single straight line with respect to some precedence constraint and to a fixed workstation capacity. Such a problem can be seen as a bin packing problem with additional constraints due to precedence. Depending on the objective function, there are two classical extensions: *SALBP-1* in which the number of workstations is minimized for a given production rate and *SALBP-2* in which the production rate is maximized for a given number of stations.

**Assembly Line Design and Constraint Programming.** When considering the design of assembly lines, additional features are generally taken into account, in particular resources required for the task execution ([24]). Resources are generally used to model machines or equipment that have to be positioned on the assembly line but also to model workers performing assembly operations. Constraint Programming (CP) models have been developed for solving various assembly line problems, and specifically problems in which resources are involved. In [11], the authors present encodings for solving SALBP-1, SALBP-2 and the Task Assignment and Equipment Selection Problem (TAESP). In the latter, one type of equipment has to be assigned to each station and the objective is to minimize the total associated cost. In [2], the authors use CP for solving the Resource-Constrained Assembly Line Balancing Problem (RCALBP) in which there can be several resources candidates for performing each task. The objective is the cycle time minimization, *i.e.* the time spent in each workstation of the line. This work has been extended in [1] into the Generalized RCALBP, in which several

modes with different resources consumption are considered for each task. These modes are expressed through conjunction normal form formulas that are handled directly by the CP solver through the use of *AND* and *OR* operators.

**Multi-criteria Line Design.** As described in [24], the assembly design problem often comes to the optimization of several criteria. In [29], the authors consider the minimization of the number of workstations, the minimization of cycle time and maximization of workload smoothness and of work relatedness. Each criterion is studied independently, except for the last two. The authors of [13] consider the problem of equipping workstations with respect to some exclusion and compatibility constraints between pieces of equipment. They study several criteria: minimization of the investment cost, maximization of the line throughput rate, minimize the occupied space and minimize the workstations complexity. In [19], the assembly line considered is mixed-model, meaning that similar models of a product are produced on the same line. Their objective is to minimize the idle time of each model on the line and minimize the total equipment cost. More recently, in [20], the authors optimize the idle time and the unit product costs. All those works mainly use evolutionary algorithms for computing a set of non-dominated solutions. An extensive comparison between 34 algorithms is provided in [20]. In [15], the authors address the two-sided assembly line balancing problem with multi-operator workstations with respect to several criteria: minimization of the number of mated workstation, minimization of the number of workstation and minimization of the number of workers. Those criteria are optimized sequentially. A CP and an ILP (Integer Linear Programming) approaches, that both consider the time at which tasks start and end, are proposed.

**Line Balancing in the Aeronautical Industry.** In the context of assembly line balancing for the aeronautical industry, literature is less extensive. The authors of [14] consider the efficiency of labor utilization, specifically for tasks that consist in preparing the aircraft for assembly, for instance when it arrives on a workstation. In a generic context of producing in low-volume, which applies for the aircraft industry, the article [6] assigns workers to station and start times to each task so as to minimize the costs of total labor and inventory costs. The authors of [10] consider the local order scheduling for mixed-model assembly lines. In [7] and more recently in [26], exact (Mixed Integer Linear Programming and CP) and heuristic methods are studied for solving a mixed-model assembly line problem in which tasks have been assigned to workstations and the objective is to minimize the overall number of operators. In [3], the authors compare CP and ILP approaches for assigning workers to tasks so as to minimize the overall makespan and while ensuring that some ergonomics constraints are satisfied. Both approaches consider not only the assignment of worker to station but also the precise scheduling of all activities. In [23], a scheduling tool is developed for bridging the gap between aircraft design and aircraft manufacturing. Such a tool relies on a local search approach and is compared with a CP encoding. The problem consists in minimizing the makespan. As done in the present paper, it considers aircraft zones that limit the number of workers that can work simultaneously. However, it does not take into account machines on the assembly line or zone neutralization and does not allow to optimize the takt-time.

Finally, [12] is a previous work in which we present results we have obtained on the same industrial project. In that work, we focus on the elicitation part and present a Goal-Oriented Requirements Engineering methodology that allowed us to obtain constraints and criteria associated with an assembly line performance. We indicate that an operations research approach was used to explore the set of assembly line solutions but this approach is not formally described and only returns a few assembly line designs solutions.

**RCPSP and Multi-Objective Optimization.** As explained in [26], the assembly line balancing problems in the aeronautical industry can be modeled into RCPSPs with some additional constraints, such as temporal constraints or incompatibility constraints. Constraint Programming provides a expressive language for such constraints, along with efficient solvers [16].

Some works of the literature address the Multi-Objective Constrained Optimization Problems (MO-COP). In [25], the authors propose a multi-objective lower bound set that allows to detect inconsistency of a multi-objective problem. The authors of [21] define an interactive algorithm for MO-COP, that gradually refines a set of Pareto solutions by exploring regions chosen by the user in which a Pareto front might exist. The specific case of Multi-Objective in RCPSP has also received attention, as shown in [4].

### 3 Preliminary Assembly Line Design Description

In this section, we describe the assembly lines we have considered, namely pulse assembly lines. Then, we briefly present the work that we have done prior to optimization for eliciting constraints and criteria relevant for a preliminary assembly line design stage.

#### 3.1 Pulse Assembly Line

The assembly line we consider in this work is a pulse line composed of several workstations. In aeronautics, pulse assembly lines are common solutions when it comes to high production rates.

The layout of pulse assembly lines we consider is a basic straight line. Each workstation in the line is responsible for performing a specific set of tasks in the assembly process, and the product being assembled moves from one workstation to the next until it is complete. When an aircraft enters the line, it goes to the first workstation. After a duration called *takt-time*<sup>2</sup>, the aircraft goes to the second workstation and a new aircraft enters the first workstation. The aircraft that was on the last workstation is done and leaves the assembly line. Tasks cannot be performed when the aircraft is moving. This implies that all tasks must be assigned to one unique workstation.

The total time that the aircraft remains on the assembly line is called *leadtime*. The leadtime is equal to the multiplication of the takt-time by the number of workstations. Note that the range of values for the takt-time of a given assembly line can be quite large. In fact, when a new line is opened, the production can start with a few aircraft per month, which corresponds to large takt-times. Depending on business considerations, the production rate can reach several dozen aircraft per month (takt-time equal to a few hours).

The set of operations performed on each workstation is always the same and they are performed in the same order for all aircraft instances. Moreover, each workstation is equipped with a set of jigs and tools, or more generally machines, that are dedicated to it. Such machines can be heavy installations, such as scaffolds for supporting the aircraft, dedicated robots or some smaller tools, such as dedicated drilling devices.

Assembly lines we consider in this paper are single-model lines as there is one product to manufacture. In practice, there can be small variants between all the aircraft present on an assembly line but such differences are not taken into account in the preliminary stage design.

---

<sup>2</sup> Takt-time is generally called cycle time in the literature. However, cycle time sometimes refer to another duration in the aircraft industry.



### 3.2 Constraints and Criteria Elicitation

Prior to the tool implementation, a major part of the project we are involved in has been dedicated to the elicitation of preliminary assembly line performance constraints and criteria. To do so, we have followed a goal-oriented requirements engineering based that is described in details in [12].

As expected, one of the criteria for the assembly line design is to minimize the associated investment cost. The costs of designing an aircraft assembly line can vary significantly depending on a variety of factors, including the size and complexity of the aircraft family, the level of automation and technology required, and the specific requirements of the production process. In the case of the preliminary assembly line design, we focus on two types of cost.

**Minimize the number of workstations.** We first consider the workstations cost. In fact, each workstation occupies a floor area in factories and has therefore an associated cost. In reality, the latter depends on the size of machines equipping the workstation and on the volume of the aircraft parts manipulated. In the assembly lines we consider in this work, it is reasonable to consider that the parts manipulated are comparable and therefore that the workstations cost is uniform. Therefore, we try to minimize the number of workstations in the assembly line.

**Minimize the number of machines.** The second type of cost is the machines and tools cost. In fact, to build an assembly line, specialized tooling and equipment must be designed and installed. This may include robots, fixtures and specialized machinery, which can be very costly. As we are in a preliminary design stage, we only consider the sizing machines and tools of the line. Cost range of such machines can be quite large as it depends on the machine features. In this paper, we focus only on highly expensive machines. As each machine is really specific, there is almost one criterion for each type of machine. In this project, we did not have any cost information nor ranking between the machines. In fact, costs can be quite complex to compute (because it implies not only buying the machine but also maintaining it). We have therefore decided to simplify the machines cost and consider only the total number of machines minimization. However, the number of machines of each type should be carefully examined when looking into details solutions in the front.

**Machines assignment and incompatibility.** Machines and tools we consider can be assigned to exactly one workstation. As they can be voluminous, there exists incompatibility between some machines types. Therefore, we have an exclusion relationship between machine.

**Minimize the leadtime.** Leadtime reduction is a critical goal for any manufacturing process. In fact, manufacturers can reduce the amount of cash that is tied up in work-in-progress inventory, which can improve cash flow and financial performance.

**Minimize the takt-time.** As expected, the higher the production rate, the more efficient the assembly line. This corresponds to the takt-time minimization. On that point, we can note that a target takt-time is generally fixed when designing a new aircraft. Nevertheless, it is essential to assess the best takt-time that is compatible with a given aircraft design, in order to prepare future production rate increase.

**Aircraft zones capacity.** The space available for workers inside the aircraft is a major constraint. Workers may need to operate in tight spaces or confined areas to access parts of the aircraft being assembled, which can increase the errors or accidents risk. This is particularly the case for tasks that require precision and accuracy, such as drilling, fastening or installing electrical components. To model that, we consider that the aircraft is divided into zones, and that each zone has a capacity representing the maximum number of workers that can perform assembly tasks simultaneously.

**Zone neutralization.** Because of safety constraints or some assembly task nature, some assembly task can neutralize aircraft zones, meaning that it prevents any other task to be performed in those neutralized zones simultaneously. For instance, if a task requires the removal of a temporary floor during its execution, it prevents access to the associated zone.

**Task features.**

- There is one alternative for each task. Each alternative is characterized by types of machine used, zones occupied and neutralized.
- Duration of tasks is fixed.
- It is not possible to interrupt tasks during their execution (no preemption). However, as we are in preliminary design, tasks we consider are in fact macro ones. Therefore, it is possible for tasks that do not consume any machine to overlap on several workstations.

## 4 Assembly Line Preliminary Design Problem Definition

In this section, we formally define the Assembly Line Preliminary Design Problem (ALPDP) and present a Constraint Programming encoding for it.

### 4.1 Problem Definition

**Assembly Line Preliminary Design Problem.** An Assembly Line Preliminary Design Problem (ALPDP) is formally defined by a tuple  $\langle \mathcal{Z}, \mathcal{S}, \mathcal{E}, \mathcal{T}, \mathcal{P} \rangle$  where:

- $\mathcal{Z}$  is the set of zones of the aircraft in which workers perform assembly activities. Each zone  $z \in \mathcal{Z}$  has a capacity, denoted  $capa_z$ , that represents the number of workers that can work simultaneously in zone  $z$ ;
- $\mathcal{S}$  is the set of machine skills that are required by assembly activities. Each skill  $s \in \mathcal{S}$  represents a type of jig and tool (*i.e.* a type of machine) used for the aircraft assembly;
- $\mathcal{E}$  is a symmetric relation included in  $\mathcal{S}^2$  that represents skills of machines that are incompatible and cannot belong to the same workstation. Note that for a given skill  $s \in \mathcal{S}$ , it is possible to have  $(s, s) \in \mathcal{E}$ : this indicates that two machines with the same skill  $s$  cannot be assigned to the same workstation;
- $\mathcal{T}$  is the set of assembly activities that must be performed.  $\mathcal{T}$  can be partitioned into two subsets: a set of atomic activities denoted  $\mathcal{A}$  and a set of composite activities denoted  $\mathcal{C}$ . Intuitively, composite activities are a group of atomic activities that allow to write precedence between activities in a compact way. They do not have a fixed duration and do not consume any resource (zone or machine);
- for each composite activity  $c \in \mathcal{C}$ ,  $\mathcal{A}_c$  is a subset of  $\mathcal{A}$  that represent all the atomic tasks that are part of composite task  $c$ .

*Assumption.* An atomic task is the child of at most one composite activity. Formally, for  $c_1$  and  $c_2$  in  $\mathcal{C}^2$  such that  $c_1 \neq c_2$ , we have  $\mathcal{A}_{c_1} \cap \mathcal{A}_{c_2} = \emptyset$ ;

- for each atomic activity  $a \in \mathcal{A}$ , we consider the following features:

- $dur_a \in \mathbb{N}^+$  is the duration of  $a$ ;
- $\mathcal{S}_a \subseteq \mathcal{S}$  is the set of skills required by  $a$ . Note that, for a given skill  $s$ , an activity requires at most one machine with that skill;
- for each zone  $z \in \mathcal{Z}$ ,  $occ_{a,z} \in \mathbb{N}^+$  is the number of places in  $z$  required by activity  $a$ ;
- $\mathcal{Z}_a^{neutr} \subseteq \mathcal{Z}$  is the set of zones that are neutralized by  $a$ . If zone  $z'$  belongs to  $\mathcal{Z}_a^{neutr}$ , then for all activity  $a'$  that occupies  $z'$  (i.e. such that  $occ_{a',z'} > 0$ ),  $a$  and  $a'$  cannot temporally overlap.

*Assumption.* For a given activity, the set of zones it occupies and the set of zones it neutralizes have an empty intersection;

- $\mathcal{P}$  is a relation in  $\mathcal{T}^2$  that represents precedence requirements between activities. More precisely,  $(t, t') \in \mathcal{P}$  if  $t$  must be finished before  $t'$  can start.

*Assumption 1.* There does not exist precedence between composite activities and their children, i.e. for all  $c \in \mathcal{C}$  and  $a \in \mathcal{A}_c$ ,  $(a, c) \notin \mathcal{P}$  and  $(c, a) \notin \mathcal{P}$ .

*Assumption 2.* The directed graph induced by relation  $\mathcal{P}$  is acyclic. To formally build such a graph, the first step is to create a node  $n_a$  for each atomic activity  $a$  and two nodes  $s_c$  and  $e_c$  for each composite activity  $c \in \mathcal{C}$  that respectively represent the start and end of  $c$ . Then, arcs are added as follows. For each precedence  $(a_1, a_2) \in \mathcal{A}^2$ , an arc is added between  $n_{a_1}$  and  $n_{a_2}$ . For each precedence  $(c_1, c_2) \in \mathcal{C}^2$ , an arc is added between  $e_{c_1}$  and  $s_{c_2}$ . For each precedence  $(a, c) \in \mathcal{A} \times \mathcal{C}$  (resp.  $(c, a) \in \mathcal{C} \times \mathcal{A}$ ), an arc is added between  $n_a$  and  $s_c$  (resp. between  $e_c$  and  $n_a$ ). Note that this graph construction is similar to the one proposed in [23].

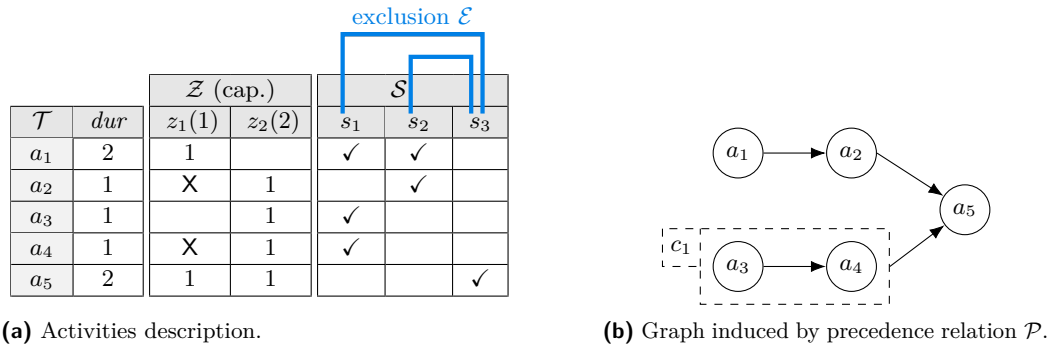
**Alternative.** An alternative for a ALPDP  $\langle \mathcal{Z}, \mathcal{S}, \mathcal{E}, \mathcal{T}, \mathcal{P} \rangle$  is described through the following elements:

- a set of workstations  $\mathcal{W}$  and for each workstation  $w \in \mathcal{W}$ , a start date and an end date;
- a takt-time, denoted  $takt$ , that represents the duration the aircraft stays on each workstation;
- for each skill  $s$  and for each workstation  $w$ , a number of machines with skill  $s$  on workstation  $w$ ;
- for each activity  $t \in \mathcal{T}$ , a start date and an end date.

The leadtime of a solution, denoted  $T_{\max}$ , is the total time the aircraft spends on workstations. Formally,  $T_{\max} = |\mathcal{W}| \cdot takt$ .

**Solution.** A solution for an ALPDP  $\langle \mathcal{Z}, \mathcal{S}, \mathcal{E}, \mathcal{T}, \mathcal{P} \rangle$  is an alternative in which the following constraints are satisfied. Note that we only give an informal definition of constraints here, as the formal definition is detailed through the Constraint Programming encoding later in the paper.

- Start dates and end dates of workstations must represent a consistent assembly line (no temporal hole and no overlap between successive workstations) and consistent with the  $takt$  value (duration of a workstation is equal to  $takt$ );
- start dates and end dates of activities are consistent with duration (for atomic activities), with children start and end dates (for composite activities) and with precedence relationship;
- capacity of zones is never exceeded and zone neutralization is respected;
- the number of machines of each skill in each workstation allows the associated activities to be performed;
- skills exclusion is respected;
- atomic activities that consume at least one machine cannot overlap on successive workstations.



■ **Figure 1** Toy example illustration.

**Criteria.** According to the criteria elicitation phase, we consider the following criteria: minimize the takt-time, minimize the leadtime value, minimize the total number of machines on the assembly line and minimize the number of workstations. As described earlier, we do not have any insight on how to aggregate this criteria together.

► **Example 1.** Figure 1 illustrates an ALPDP toy example. In this example, the set of zones  $\mathcal{Z}$  is composed of two zones  $z_1$  and  $z_2$  that respectively have a capacity equal to 1 and 2. The set of skills  $\mathcal{S}$  contains three skills,  $s_1$ ,  $s_2$  and  $s_3$ , and we consider that  $s_1$  and  $s_3$  exclude each other, and so do skills  $s_2$  and  $s_3$  (i.e.  $\mathcal{E} = \{(s_1, s_3), (s_3, s_1), (s_2, s_3), (s_3, s_2)\}$ ). The set of activities  $\mathcal{T}$  contains one composite activity  $c_1$  and five atomic activities  $a_i$  with  $i \in [1..5]$ . Composite activity  $c_1$  encompasses activities  $a_3$  and  $a_4$ .

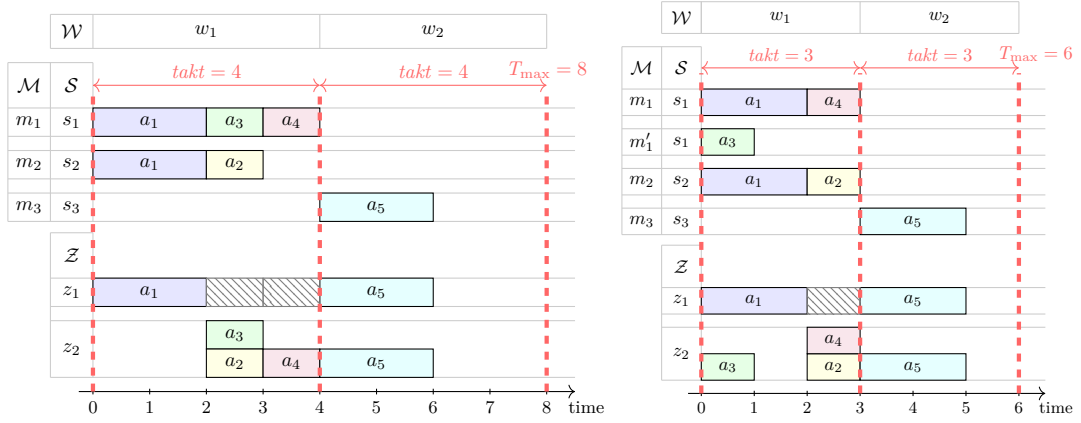
Table 1a describes atomic activities features. The X symbol indicates that an activity neutralizes a zone. The ✓ symbol indicates that an activity requires a skill. For instance, the second line of the table indicates that activity  $a_2$ : has a duration equal to 1, neutralizes zone  $z_1$ , occupies one place in zone  $z_2$ , and requires skill  $s_2$ .

Precedence relation is  $\mathcal{P} = \{(a_1, a_2), (a_2, a_5), (c_1, a_5), (a_3, a_4)\}$ , as illustrated on Figure 1b<sup>3</sup>. Figure 2a illustrates a solution, denoted  $sol_{3machines}$  for this ALPDP. In this solution, there are two workstations,  $w_1$  and  $w_2$ . The first workstation contains machine  $m_1$  with skill  $s_1$  and machine  $m_2$  with skill  $s_2$ . Workstation  $w_2$  contains machine  $m_3$  with skill  $s_3$ . Start dates of activities are respectively 0, 2, 2, 3, 4 for  $a_1, \dots, a_5$ . Composite activity  $c_1$  starts with its earliest child (here  $a_3$  at time 2) and ends with its latest (here  $a_4$  at time 4). In this solution, the takt is equal to 4 and the leadtime equal to 8.

A second solution,  $sol_{4machines}$ , is illustrated on Figure 2b. In this solution, there are two machines with skill  $s_1$  in the first workstation. Note that for this solution, composite activity  $c_1$  is not continuous in the sense that there is a temporal hole between its children activities. This solution allows to reach a takt equal to 3 and a leadtime equal to 6.

We can notice that  $sol_{3machines}$  is better than  $sol_{4machines}$  with respect to the total number of machines criteria. Both solutions are equivalent for the number of stations. However,  $sol_{4machines}$  is better than  $sol_{3machines}$  with respect to the takt and to the leadtime. Thus, no solution dominates the other on all criteria.

<sup>3</sup> Note that we do not represent explicitly start and end activities associated with  $c_1$ .



(a) Solution  $sol_{3machines}$ , with one machine for each skill, a takt-time equal to 4 and a leadtime equal to 8. (b) Solution  $sol_{4machines}$ , with two machines for skill  $s_1$ , a takt-time equal to 3 and a leadtime equal to 8.

■ **Figure 2** Two solution examples that each has two workstations.

## 4.2 Constraint Programming Encoding

We present here a Constraint Programming encoding for the ALPDP problem. For this encoding, we suppose that we are given two additional input parameters: an upper bound for the number of workstations in the assembly line, denoted  $nW$ , and an upper bound  $H$  for the leadtime.

We base our CP encoding on data structures and functions that are available in the Optimization Programming Language (OPL - [28]). In particular, we use interval variables and state functions. An interval variable encompasses a start date variable and a duration variable. It can be optional, *i.e.* it might not be present in the schedule, and it can be temporally bounded. State functions allow to express values that a function should take over given temporal intervals.

We consider the following decision variables:

- for each activity  $t \in \mathcal{T}$ ,  $itv_t$  is an interval variable in  $[0, H]$  that represented the execution of activity  $a$ . For atomic activities  $a \in \mathcal{A}$ , the duration of the interval is fixed to  $dur_a$ ;
- for each atomic activity  $a \in \mathcal{A}$ , for each workstation  $w \in [1..nW]$ ,  $itv_{a,w}$  is an optional interval variable in  $[0, H]$  with a duration fixed to  $dur_a$ . The interval  $itv_{a,w}$  is present if and only if  $a$  starts being performed in workstation  $w$ ;
- for each workstation  $w \in [1..nW]$ ,  $itv_w$  is an optional interval variable in  $[0, H]$ .  $itv_w$  is present if and only if  $w$  is a used workstation, *i.e.* a workstation in which some activities are performed;
- $takt$  is an integer variable in  $[1, H]$ ;
- for each skill  $s \in \mathcal{S}$ , for each workstation  $w \in [1..nW]$ ,  $skUsed_{s,w}$  is a boolean variable that indicates whether at least one machine with skill  $s$  is assigned to workstation  $w$  and  $nMachines_{s,w}$  is a integer variable that represents the number of machines with skill  $s$  that are assigned to  $w$ . An upper bound for that variable is the number of atomic activities.
- for each zone  $z \in \mathcal{Z}$ ,  $stateOcc_z$  is a state function that represents the occupation state of zone  $z$ . Such a state is 1 when  $z$  is occupied by an activity being performed and is equal to 0 if  $z$  is neutralized by an activity.

With those decision variables, the ALPDP constraints can be encoded as follows.

**Assembly Line Consistency Constraints.**

$$\forall w \in [1..nW], \quad \text{sizeOf}(itv_w, 0) \leq \text{takt} \quad (1)$$

$$\forall w \in [1..nW], \quad \text{sizeOf}(itv_w, H) \geq \text{takt} \quad (2)$$

$$\text{startOf}(itv_1, 0) = 0 \quad (3)$$

$$\forall w \in [2..nW], \quad \text{endAtStart}(itv_{w-1}, itv_w) \quad (4)$$

$$\forall w \in [2..nW], \quad \text{presenceOf}(itv_{w-1}) \geq \text{presenceOf}(itv_w) \quad (5)$$

Constraints (1) and (2) together ensure that the duration of a used workstation is equal to the takt-time. The second parameter in the *sizeOf* function defines the value of an interval variable that is not present in the final schedule. Through Constraint (3), the first workstation starts at time 0. Constraint (4) prevents temporal holes between successive workstations. Finally, Constraint (5) ensures that no unused workstation is placed in between used workstations.

**Activities and Workstation Constraints.**

$$\forall c \in \mathcal{C}, \quad \text{span}(itv_c, \{itv_a \mid a \in \mathcal{A}_c\}) \quad (6)$$

$$\forall (t, t') \in \mathcal{P}, \quad \text{endBeforeStart}(itv_t, itv_{t'}) \quad (7)$$

$$\forall a \in \mathcal{T}, \quad \text{alternative}(itv_a, \{itv_{a,w} \mid w \in [1..nW]\}) \quad (8)$$

$$\forall a \in \mathcal{A}, \forall w \in [1..nW], \quad \text{startBeforeStart}(itv_w, itv_{a,w}) \quad (9)$$

$$\forall a \in \mathcal{A} \text{ s.t. } \mathcal{S}_a \neq \emptyset, \forall w \in [1..nW], \quad \text{endBeforeEnd}(itv_{a,w}, itv_w) \quad (10)$$

$$\forall a \in \mathcal{T}, \forall w \in [1..nW], \quad \text{presenceOf}(itv_{a,w}) \leq \text{presenceOf}(itv_w) \quad (11)$$

$$\forall w \in [1..nW], \quad \text{presenceOf}(itv_w) \leq \sum_{a \in \mathcal{A}} \text{presenceOf}(itv_{a,w}) \quad (12)$$

$$\forall a \in \mathcal{A}, \quad \text{endOf}(itv_a) \leq \max_{w \in [1..nW]} \text{endOf}(itv_w, 0) \quad (13)$$

Constraint (6) makes interval variables associated to composite activities span over interval variables of their children. Constraint (7) enforces precedence constraints associated with precedence relation  $\mathcal{P}$ . Constraint (8) uses the *alternative* constraint in CP Optimizer and guarantees that there exists exactly one workstation  $w$  such that  $itv_{a,w}$  coincides with interval  $itv_a$  (only for atomic activities). Constraint (9) ensures the consistency between start dates of workstation  $w$  and of activity  $a$  if the latter starts in workstation  $w$ . Constraint (10) ensures a similar consistency for end dates but only for activities that require machines. Constraints (11) and (12) guarantees the consistency between presence of activities on workstations intervals and presence of workstation intervals. Constraint (13) ensures that workstations on which atomic activities finish are used.

**Machines Constraints.**

$$\forall s \in \mathcal{S}, \forall w \in [1..nW], \quad \sum_{a \in \mathcal{A}, s \in \mathcal{S}_a} pulse(itv_{a,w}, 1) \leq nMachines_{s,w} \quad (14)$$

$$\forall s \in \mathcal{S}, \forall w \in [1..nW], \quad nMachines_{s,w} \geq skUsed_{s,w} \quad (15)$$

$$\forall s \in \mathcal{S}, \forall w \in [1..nW], \quad nMachines_{s,w} \leq |\mathcal{A}| \cdot skUsed_{s,w} \quad (16)$$

$$\forall (s, s') \in \mathcal{E} \text{ s.t. } s \neq s', \forall w \in [1..nW], \quad skUsed_{s,w} + skUsed_{s',w} \leq 1 \quad (17)$$

$$\forall s \in \mathcal{S} \text{ s.t. } (s, s) \in \mathcal{E}, \forall w \in [1..nW], \quad nMachines_{s,w} \leq 1 \quad (18)$$

$$\forall s \in \mathcal{S} \text{ s.t. } \exists a \in \mathcal{A} \text{ with } s \in \mathcal{S}_a, \quad \sum_{w=1}^{nW} skUsed_{s,w} \geq 1 \quad (19)$$

$$\sum_{w=1}^{nW} \sum_{s \in \mathcal{S}} nMachines_{s,w} \geq M_{UB} \quad (20)$$

$$\text{with } M_{UB} = |\{s \in \mathcal{S} | \exists a \in \mathcal{A}, s \in \mathcal{S}_a\}|$$

Through Constraint (14), we build the consumption profile of machines with skill  $s$  by using the *pulse* primitive and we ensure that this consumption does not exceed the number of machines with skill  $s$  in the workstation. Constraints (15) and (16) express the relationship between *nMachines* and *skUsed* variables: *skUsed* <sub>$s,w$</sub>  is equal to 0 if and only if the number of machines is equal to 0. Constraint (17) forbids to use excluded skills in the same workstation. Constraint (18) handles the specific case in which exclusion targets the same skill. Constraints (19) and (20) give an upper bound for the total number of machines.

**Zones Constraints.**

$$\forall z \in \mathcal{Z} \text{ s.t. } capa_z = 1, \quad noOverlap(\{itv_a | occ_{a,z} > 0\}) \quad (21)$$

$$\forall z \in \mathcal{Z} \text{ s.t. } capa_z > 1, \quad \sum_{\substack{a \in \mathcal{A} \\ occ_{a,z} > 0}} pulse(itv_a, occ_{a,z}) \leq capa_z \quad (22)$$

$$\forall a \in \mathcal{A}, \forall z \in \mathcal{Z} \text{ s.t. } occ_{a,z} > 0, \quad alwaysEqual(stateOcc_z, itv_a, 1) \quad (23)$$

$$\forall a \in \mathcal{A}, \forall z \in \mathcal{Z}_a^{neutr}, \quad alwaysEqual(stateOcc_z, itv_a, 0) \quad (24)$$

Constraints (21) and (22) express that the capacity of zones is never exceeded (constraints (21) considers the specific case when zone have a capacity equal to 1). Finally, Constraints (23) and (24) enforce the state functions to be equal to 0 or 1 depending of occupation and neutralization of activities. As state function have one value for each instant, this prevents activities occupying a zone and activities neutralizing it to overlap.

We consider the following four criteria:

$$\text{minimize} \quad takt \quad (25)$$

$$\text{minimize} \quad \max_{w \in [1..nW]} endOf(itv_w, 0) \quad (26)$$

$$\text{minimize} \quad \sum_{w \in [1..nW]} presenceOf(itv_w) \quad (27)$$

$$\text{minimize} \quad \sum_{w \in [1..nW]} \sum_{s \in \mathcal{S}} nMachines_{s,w} \quad (28)$$

Criterion (25), (26), (27) and (28) respectively encode the minimization of the rate, the leadtime, the number of workstations in the assembly line and the total number of machines used. Note that, because of the equation  $takt \cdot |\mathcal{W}| = T_{\max}$ , if the takt-time and the leadtime are fixed, so is the number of workstations. This means that one of the three first criteria is redundant. Therefore, in the following, we only consider the takt-time, the leadtime and the number of machines minimization.

## 5 Algorithmic approach

In this section, we describe how we have used the previously defined Constraint Programming encoding to explore the Pareto solutions set.

The algorithm follows an epsilon-constraint based strategy ([4]), which requires bounds for each criteria. In our problem, there are three criteria and we therefore have to find in which order we explore them, how to fix the bounds and the number of steps on each part of the front. As the resulting procedure is not trivial, we present it in this section. We believe that this exploration algorithm could be a starting point for other problems with three or more criteria. In fact, to the best of our knowledge, the literature mostly contains epsilon-constraint based strategies for problems with two criteria.

In order to compute bounds for the criteria values, we use a lexicographic objective function. Then, for a given number of machines in the assembly line, we compute a takt-time step that allows to explore solutions. From the set of all computed solutions, we extract the Pareto ones.

The algorithm is formally described in Listing 1. In the pseudo-code, a run of the Constraint Programming encoding is represented through the function `cpSolve` that requires two parameters. The first one is the set of constraints to satisfy and the second one is the objective function to optimize. Such an objective function can be simple (one criterion) or a lexicographic order over several criteria (denoted  $lex(\dots)$ ). In the pseudo-code, we consider that we have a data structure for storing solution. If  $sol$  is a solution, then  $sol.machines$ ,  $sol.takt$  and  $sol.T_{max}$  respectively return the number of machines, the takt-time and the leadtime values associated with  $sol$ .

The algorithm's inputs are all constraints ((1) to (24)) of an ALPDP instance (denoted  $P$ ) and a maximum number of points in the Pareto front for each number of machines. We start by computing three solutions  $sol^M$ ,  $sol^T$  and  $sol^L$  that respectively minimize the number of machines, the takt and the leadtime values (lines 2-4). Solution  $sol^M$  provides a lower bound for the number of machines. The three solutions are added to the set of solutions through the function `updateFront` that maintains the set of Pareto solutions (line 5). While minimizing the takt and the leadtime, we also minimize the number of machines as the last criterion of a lexicographic objective function, which allows to compute an upper bound of the number of machines (line 6).

Next, we go through each value of the total number of machines, starting from the upper-bound. For each value, we consider the associated constraint  $c_{MACHINES}$  that limits the number of machines in the assembly line (line 8). Then, we compute a lower bound and an upper bound for the takt-time, respectively through  $sol_1$  and  $sol_2$  (lines 9-10), that are stored in variables  $takt_1$  and  $takt_2$ . If there is a Pareto front to explore (difference between both takt-times strictly greater than 1), we compute a step, denoted  $\delta$ , for decreasing the considered Stakt-time according to the number of points in input (line 14). Starting from the takt upper-bound, we consider a constraint  $c_{TAKT}$  that fixes the takt-time and look for a solution satisfying that constraint (line 18). If it exists, the takt-value is decreased from  $\delta$  (line 22) and this step is iterated until reaching the lower bound. Otherwise, there is no solution with a smaller takt value for that number of machines and the loop is stopped. The algorithm finally returns the Pareto front.

Note that the number of workstations criteria is never used in the algorithm. In fact, if two of the three criteria  $takt$ ,  $nWorkstations$  and  $T_{max}$  are fixed, so is the third. As presented in the Experiments Section, results tend to show that `cpSolve` is less efficient when it comes to minimize the number of workstations. Therefore, we use the two other criteria.



■ **Algorithm 1** Epsilon approach for ALPDP.

---

```

1: function epsilonParetoALPDP( $P, nMaxPointsPerMachine$ )
2:    $sol^M \leftarrow \mathbf{cpSolve}(P, MACHINES)$ 
3:    $sol^T \leftarrow \mathbf{cpSolve}(P, \mathit{lex}(TAKT, LEADTIME, MACHINES))$ 
4:    $sol^L \leftarrow \mathbf{cpSolve}(P, \mathit{lex}(LEADTIME, TAKT, MACHINES))$ 
5:    $front \leftarrow \mathit{updateFront}(\{\}, \{sol^M, sol^R, sol^T\})$ 
6:    $m \leftarrow \max(sol^L.machines, sol^T.machines)$ 
7:   while  $m \geq sol^M.machines$  do
8:      $c_{MACHINES} \leftarrow \mathbf{CONSTRAINT}(\sum_{w \in [1..nW]} \sum_{s \in \mathcal{S}} nMachines_{s,w} \leq m)$ 
9:      $sol_1 \leftarrow \mathbf{cpSolve}(P \cup \{c_{MACHINES}\}, \mathit{lex}(TAKT, LEADTIME))$ 
10:     $sol_2 \leftarrow \mathbf{cpSolve}(P \cup \{c_{MACHINES}\}, \mathit{lex}(LEADTIME, TAKT))$ 
11:     $front \leftarrow \mathit{updateFront}(front, \{sol_1, sol_2\})$ 
12:     $takt_1 \leftarrow sol_1.takt; takt_2 \leftarrow sol_2.takt$ 
13:    if  $takt_2 - takt_1 > 1$  then
14:       $\delta = \lfloor \frac{takt_2 - takt_1}{nMaxPointsPerMachine} \rfloor$ 
15:       $t \leftarrow takt_2 - \delta$ 
16:       $sol \leftarrow sol_1$ 
17:      while  $t > takt_1 \wedge sol \neq nil$  do
18:         $c_{TAKT} \leftarrow \mathbf{CONSTRAINT}(takt = t)$ 
19:         $sol \leftarrow \mathbf{cpSolve}(P \cup \{c_{MACHINES}, c_{TAKT}\}, LEADTIME)$ 
20:        if  $sol \neq nil$  then
21:           $front \leftarrow \mathit{updateFront}(front, \{sol\})$ 
22:           $t \leftarrow t - \delta$ 
23:       $m \leftarrow m - 1$ 
24:    return  $front$ 

```

---

Functions *updateFront* and *dominates* are quite straightforward and therefore not detailed in the paper. Note that the Pareto front does not take into account the number of workstations.

## 6 Experiments

This section is dedicated to the experiments presentation. We first describe the benchmarks that we have used. We present two sets of experiments. First, we have solved the CP encoding with each criteria alone. This allows us to customize the **cpSolve** procedure in terms of heuristics and time out. Then, we present and analyze results associated with the Pareto solutions computation.

### 6.1 Benchmark Description

We have worked on a real industrial use-case dealing with the airframe assembly line of an aircraft. The airframe is the physical structure that supports all the other components of the aircraft, including avionics, passenger and cargo compartments. The materials used in its construction can include aluminium alloys, composites, titanium and other high-strength materials. The materials are cut and shaped into the various components using tools such as saws and drills. The assembly process for the airframe components typically involves careful alignment and attachment using specialised tools and techniques.

We were given three aircraft designs candidates, denoted *Design 1*, *Design 2* and *Design 3*. Features of instances are detailed in Table 1. Designs 1 and 2 have the same granularity level in terms of macro-tasks and have to be compared with each other. Design 3 is a detailed version of Design 1 with more fine-grain tasks that allows to study the scalability of the approach. These benchmarks are available at <https://github.com/stephroussel/assemblyLine>.

■ **Table 1** Instances features: number of zones, skills, incompatible skills, tasks and atomic tasks, precedence, maximum number of workstations and time horizon.

Instance	$ Z $	$ S $	$ E $	$ T $	$ A $	$ P $	$nW$	$H$
Design 1	48	5	6	176	153	186	20	40000
Design 2	48	5	6	187	187	279	20	40000
Design 3	48	5	6	628	461	417	20	40000

In this use case, the number of skills is the same in each instance. In fact, all the designs require the same types of machines but these machines are not used the same way.

## 6.2 Single Objective Experiments

**Experiments setup.** We use IBM CP Optimizer 22.1.1 through the Java API with a timeout equal to 5 minutes. Experiments were all run on a 20-core Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz, 62GB RAM.

In order to test the CP encoding, we have first experimented it on the instances by considering one unique criterion for every run. The objective was to improve the solver setup and specifically the search strategy.

In CP Optimizer, the default search strategy is generally efficient. However, in our problem, we noticed a real performance improvement when asking the solver to instantiate first the *takt* variable. In fact, it allows to fix the duration of workstations, which is a key element for the remaining variables. Once this variable is fixed, we have tried other strategies such as deciding the presence of workstations or the number of machines in each workstation but it globally downgraded the performance compared to the solver default strategy. Therefore, we only present in this paper results associated with the *takt* variable instantiation strategy.

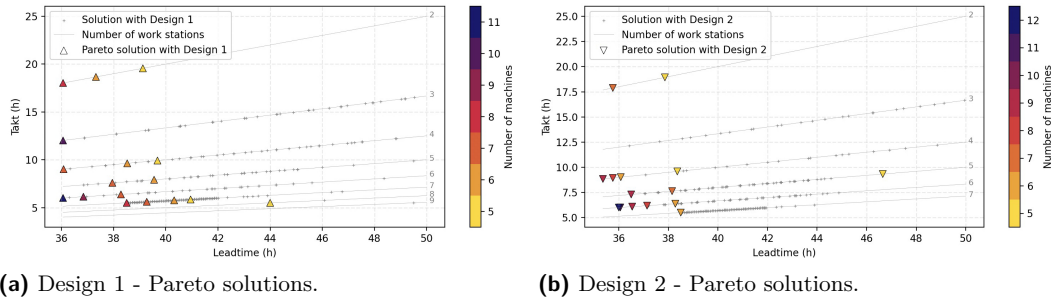
Results for the mono-objective solving are presented in Table 2 for each design and each criteria. The solver finds the optimal solution in several cases, even for the largest instance. The optimal solution is either found in less than a minute or reached the time out. Note that the solver was unable to establish the optimality of the solution with respect to the number of workstations. On that point, it would be possible to compute a lower bound using the skill exclusion relationships. We could, for instance, find the size of the largest clique in the exclusion skills graph.

■ **Table 2** Mono-criteria CP solving with search phase customization on the *takt* variable. For each criteria, value and time (in seconds) are given. The symbol \* denotes that the solution found is optimal.

Instance \ Criterion	<i>takt</i>		<i>nMachines</i>		$T_{\max}$		<i>nWorkstations</i>	
	Value	Time	Value	Time	Value	Time	Value	Time
Design 1	550*	25	5*	69	3606*	59	2	300
Design 2	550*	32	5*	28	3536	300	2	300
Design 3	340*	35	8	300	3456	300	2	300

### 6.3 Multi Objective Results

**Experiments setup.** We did not give a global timeout to the whole algorithm but instead a 1 minute timeout to each call to **cpSolve**. Note that we have let CP Optimizer handle directly the lexicographic criteria. In fact, preliminary experiments showed that there was no real performance improvement when decomposing the resolution into optimizing the first criterion and fix it, then optimizing the next one and so on.

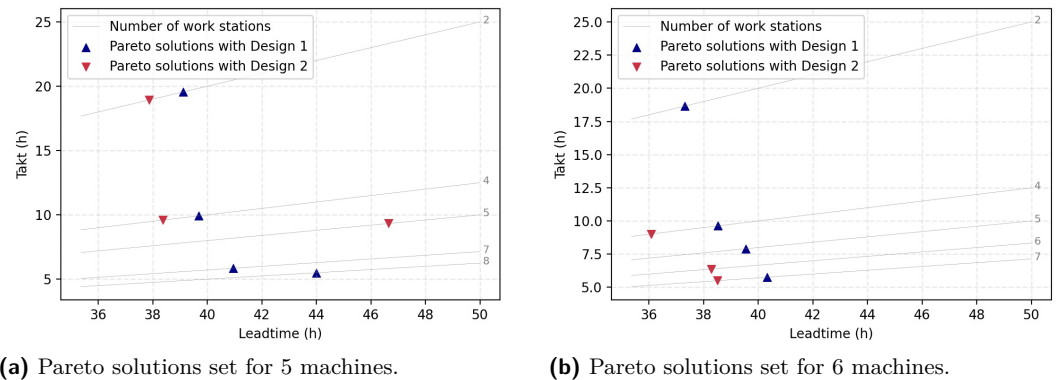


■ **Figure 3** Pareto solutions obtained for Designs 1 and 2.

Figures 3a and 3b show the Pareto front obtained respectively for Designs 1 and 2. The leadtime is on the horizontal axis, the takt-time on the vertical axis, the number of machines is represented through colors and the number of stations are the lines on each figure. Each small cross corresponds a to dominated solution. Note that for Design 2, we have not represented a Pareto solution that had 18 stations and a smaller takt-time.

As expected, for each design, the more machines on the assembly line, the better takt-time and leadtime values it is possible to get. Each figure shows the possible trade-offs that have to be made for each design. For instance, in Design 1, with 2 workstations, the best takt-time is equal to 18 hours. When adding 1 more workstation, such a value falls to 12 hours but requires many more machines in order to maintain the leadtime value. With one additional workstation and the same leadtime, the takt-time is less than 10 hours. We also observe that it is not worth considering more than 8 workstations for that design. Similar remarks can be made for Design 2.

Figure 4a presents Pareto solutions for Designs 1 and 2 when the assembly line is equipped with 5 machines. It shows that Design 1 (in blue on the Figure) allows to reach lower takt-time values. If one more machine is available (Figure 4b), then solutions associated with Design 2 dominate solutions of Design 1.



■ **Figure 4** Comparison of Design 1 and 2 with 5 and 6 machines.

These results have been presented to industrial partners. They have appreciated the possibility to visualize the Pareto front, which allows them to foresee the trade offs that could be made not only within the assembly line (number of workstations, number of machines, etc) but also between the aircraft design and the associated assembly line. Following results presented on Figure 4, they were surprised that Design 2 seems more promising than Design 1 only in the presence of an additional machine. Indeed, before this study, they had the false intuition that Design 2 would dominate Design 1 even with 5 machines. Such a positive feedback from end-users experts shows the added value of computing and exploring the Pareto front in this project.

## 7 Conclusion

In this paper, we have proposed a Constraint Programming based approach for supporting manufacturer in the early assembly design phase. We have formally defined the associated problem and have developed an algorithm for exploring the Pareto front. Each solution in this front is a trade off between the takt-time, the leadtime, the number of machines equipping the assembly line and the number of workstations. Such tools allow the aircraft manufacturer not only to assess the performance of aircraft candidates with respect to the assembly line performances but also to compare candidates with each other.

There are several directions for future works. First, the CP encoding could be improved by computing some lower and upper bounds for the targeted criteria. In fact, the addition of redundant constraints for boosting the solving might change the way the Pareto exploration should be performed and should be studied more deeply. Then, the Pareto front exploration could benefit from some recent works on computing representative Pareto solutions, such as [27]. It could also be possible to compare the CP approach and the Pareto exploration with evolutionary algorithms in terms of quality of results. We would also like to port the model to other CP solvers in order to test them. While most of the constraints could easily be written in a more classical language such as PyCSP3 ([17]), the neutralization constraints modeled by a state function in OPL might be a little trickier to encode in order to stay compact.

The instances we have used here come from a real assembly line. In order to test the approach more broadly, it would be possible to modify these instances by adding random resources or changing the precedence between tasks. Similarly, we could also adapt existing instances of the literature.

Finally, a last perspective is to consider uncertainty in the tasks duration. To do so, it might be worth considering coupling CP solvers with learning approaches that are particularly suited for managing uncertainty.

---

## References

- 1 Hacı Mehmet Alakaş. General resource-constrained assembly line balancing problem: conjunction normal form based constraint programming models. *Soft Computing*, 25(8):6101–6111, 2021.
- 2 Hacı Mehmet Alakaş, Mehmet Pınarbaşı, and Mustafa Yüzükırmızı. Constraint programming model for resource-constrained assembly line balancing problem. *Soft Computing*, 24:5367–5375, 2020.
- 3 Dmitry Arkhipov, Olga Battaïa, Julien Cegarra, and Alexander Lazarev. Operator assignment problem in aircraft assembly lines: a new planning approach taking into account economic and ergonomic constraints. *Procedia CIRP*, 76:63–66, 2018.

- 4 Francisco Ballestín and Rosa Blanco. Theoretical and practical fundamentals for multi-objective optimisation in resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):51–62, 2011. Project Management and Scheduling. doi:10.1016/j.cor.2010.02.004.
- 5 Olga Battaia and Alexandre Dolgui. A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142(2):259–277, 2013. Anticipation of risks impacts and industrial performance evaluation in distributed organizations life cycles. doi:10.1016/j.ijpe.2012.10.020.
- 6 Alexander Biele and Lars Mönch. Hybrid approaches to optimize mixed-model assembly lines in low-volume manufacturing. *Journal of Heuristics*, 24(1):49–81, 2018.
- 7 Tamara Borreguero, Alvaro García, and Miguel Ortega. Scheduling in the aeronautical industry using a mixed integer linear problem formulation. *Procedia engineering*, 132:982–989, 2015.
- 8 Nils Boysen, Philipp Schulze, and Armin Scholl. Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research*, 301(3):797–814, 2022. doi:10.1016/j.ejor.2021.11.043.
- 9 Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1):3–41, 1999.
- 10 Jens Buergin, Sina Helming, Jan Andreas, Philippe Blaettchen, Yannick Schweizer, Frank Bitte, Benjamin Haefner, and Gisela Lanza. Local order scheduling for mixed-model assembly lines in the aircraft manufacturing industry. *Production Engineering*, 12:759–767, 2018.
- 11 Yossi Bukchin and Tal Raviv. Constraint programming for solving various assembly line balancing problems. *Omega*, 78:57–68, 2018. doi:10.1016/j.omega.2017.06.008.
- 12 Anouck Chan, Anthony Fernandes Pires, Thomas Polacsek, and Stéphanie Roussel. The aircraft and its manufacturing system: From early requirements to global design. In Xavier Franch, Geert Poels, Frederik Gailly, and Monique Snoeck, editors, *Advanced Information Systems Engineering - 34th International Conference, CAiSE 2022, Leuven, Belgium, June 6-10, 2022, Proceedings*, volume 13295 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2022. doi:10.1007/978-3-031-07472-1\_10.
- 13 Hicham Chehade, Alexandre Dolgui, Frédéric Dugardin, Lina Makdessian, and Farouk Yalaoui. Multi-objective approach for production line equipment selection. *Management and Production Engineering Review*, 3(1):4–17, 2012.
- 14 G. Heike, M. Ramulu, E. Sorenson, P Shanahan, and Kamran Moinzadeh. Mixed model assembly alternatives for low-volume manufacturing: the case of the aerospace industry. *International Journal of Production Economics*, 72(2):103–120, 2001.
- 15 Damla Kizilay and Zeynel Abidin Çil. Constraint programming approach for multi-objective two-sided assembly line balancing problem with multi-operator stations. *Engineering Optimization*, 53(8):1315–1330, 2021. doi:10.1080/0305215X.2020.1786081.
- 16 Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 23:210–250, 2018.
- 17 Christophe Lecoutre and Nicolas Szczepanski. PyCSP3: Modeling Combinatorial Constrained Problems in Python. Technical report, arXiv, December 2021. URL: <https://hal-univ-artois.archives-ouvertes.fr/hal-03701203>.
- 18 Fernando Mas, José Luis Menéndez, Manuel Oliva, Javier Servan, Rebeca Arista, and Carmelo Del Valle. Design within complex environments: Collaborative engineering in the aerospace industry. In *Information System Development: Improving Enterprise Communication*, pages 197–205. Springer, 2014.
- 19 Jonathan Oesterle and Lionel Amodeo. Efficient multi-objective optimization method for the mixed-model-line assembly line design problem. *Procedia CIRP*, 17:82–87, 2014. Variety Management in Manufacturing. doi:10.1016/j.procir.2014.01.038.

- 20 Jonathan Oesterle, Lionel Amodeo, and Farouk Yalaoui. A comparative study of multi-objective algorithms for the assembly line balancing and equipment selection problem under consideration of product design alternatives. *Journal of intelligent Manufacturing*, 30:1021–1046, 2019.
- 21 Tenda Okimoto, Yongjoon Joe, Atsushi Iwasaki, Toshihiro Matsui, Katsutoshi Hirayama, and Makoto Yokoo. Interactive algorithm for multi-objective constraint optimization. In Michela Milano, editor, *Principles and Practice of Constraint Programming*, pages 561–576, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 22 Thomas Polacsek, Stéphanie Roussel, Cédric Pralet, and Claude Cuiller. Design for efficient production, A model-based approach. In Manuel Kolp, Jean Vanderdonckt, Monique Snoeck, and Yves Wautelet, editors, *13th International Conference on Research Challenges in Information Science, RCIS 2019, Brussels, Belgium, May 29-31, 2019*, pages 1–6. IEEE, 2019. doi:10.1109/RCIS.2019.8877088.
- 23 Cédric Pralet, Stéphanie Roussel, Thomas Polacsek, François Bouissière, Claude Cuiller, Pierre-Eric Dereux, Stéphane Kersuzan, and Marc Lelay. A scheduling tool for bridging the gap between aircraft design and aircraft manufacturing. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28(1):347–355, June 2018. doi:10.1609/icaps.v28i1.13910.
- 24 Brahim Rekiek, Alain Delchambre, Alexandre Dolgui, and Antoneta Bratcu. Assembly line design: A survey. *IFAC Proceedings Volumes*, 35(1):155–166, 2002. 15th IFAC World Congress. doi:10.3182/20020721-6-ES-1901.01647.
- 25 Emma Rollon and Javier Larrosa. Multi-objective propagation in constraint programming. *Frontiers in Artificial Intelligence and Applications*, 141:128, 2006.
- 26 Tamara Borreguero Sanchidrián, Tom Portoleau, Christian Artigues, Alvaro García Sánchez, Miguel Ortega Mier, and Pierre Lopez. Exact and heuristic methods for an aeronautical assembly line time-constrained scheduling problem with multiple modes and a resource leveling objective. *hal-03344445*, 2021.
- 27 Nicolas Schwind and Demirović Emir. Representative solutions for bi-objective optimisation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1436–1443, 2020.
- 28 Pascal Van Hentenryck, Laurent Michel, Laurent Perron, and Jean-Charles Régim. Constraint programming in opl. In *PPDP*, volume 99, pages 98–116. Springer, 1999.
- 29 Yeo Keun Kim, Yong Ju Kim, and Yeongho Kim. Genetic algorithms for assembly line balancing with various objectives. *Computers & Industrial Engineering*, 30(3):397–409, 1996. IE in Korea. doi:10.1016/0360-8352(96)00009-5.



# SAT-Based Learning of Compact Binary Decision Diagrams for Classification

Pouya Shati ✉

Department of Computer Science, University of Toronto, Canada  
Vector Institute, Toronto, Canada

Eldan Cohen ✉

Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Sheila McIlraith ✉

Department of Computer Science, University of Toronto, Canada  
Vector Institute, Toronto, Canada

---

## Abstract

Decision trees are a popular classification model in machine learning due to their interpretability and performance. However, the number of splits in decision trees grow exponentially with their depth which can incur a higher computational cost, increase data fragmentation, hinder interpretability, and restrict their applicability to memory-constrained hardware. In contrast, binary decision diagrams (BDD) utilize the same split across each level, leading to a linear number of splits in total. Recent work has considered optimal binary decision diagrams (BDD) as compact and accurate classification models, but has only focused on binary datasets and has not explicitly optimized the compactness of the resulting diagrams. In this work, we present a SAT-based encoding for a multi-terminal variant of BDDs (MTBDDs) that incorporates a state-of-the-art direct encoding of numerical features. We then develop and evaluate different approaches to explicitly optimize the compactness of the diagrams. In one family of approaches, we learn a tree BDD first and model the size of the diagram the tree will be reduced to as a secondary objective, in a one-stage or two-stage optimization scheme. Alternatively, we directly learn diagrams that support multi-dimensional splits for improved expressiveness. Our experiments show that direct encoding of numerical features leads to better performance. Furthermore, we show that exact optimization of size leads to more compact solutions while maintaining higher accuracy. Finally, our experiments show that multi-dimensional splits are a viable approach to achieving higher expressiveness with a lower computational cost.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial optimization; Computing methodologies → Machine learning

**Keywords and phrases** Binary Decision Diagram, Classification, Compactness, Numeric Data, MaxSAT

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.33

**Supplementary Material** *Software (Source code)*: <https://github.com/PouyaShati/BDD>

**Funding** We gratefully acknowledge funding from Natural Sciences and Engineering Research Council of Canada (NSERC) and the CIFAR AI Chairs program (Vector Institute).

## 1 Introduction

Classifiers are complete functions for assigning labels to datapoints, learned from a limited set of supervised training data. A classifier is trained to focus on informative aspects of the input and extract patterns from the data to make decisions. In complex black-box classifiers such as deep and convoluted neural networks, it is often challenging to understand the influential features of the data and the inner-workings of the decision-making [36, 38]. In contrast, interpretable classifiers such as decision trees [1, 26, 33, 34] and diagrams [16, 24] are concise and easy to understand [10]. The simplicity in interpretable solutions makes them the



© Pouya Shati, Eldan Cohen, and Sheila McIlraith;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 33; pp. 33:1–33:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



perfect candidates for when we need to formally analyze or explain the classifier’s behavior [27, 19, 20]. Surprisingly, the benefit of interpretability does not come with significant cost to accuracy in many applications [31, 25].

Binary decision diagrams (BDD) are graph representations of functions with binary inputs and are widely used in logical synthesis and formal verification methods [8, 2, 28, 23]. While equivalent to truth tables in purpose, BDDs are more compact since redundant sub-tables can be eliminated and merged [23]. A multi-terminal BDD (MTBDD) is a BDD variant that supports multiple outputs [11].

Decision trees are the most common form of interpretable classifiers (e.g., [7, 29, 30, 3, 37, 18, 5, 33, 34]). However, the number of splits double at each level of a decision tree, making it challenging to interpret the solution as depth increases [15]. The large number of splits in decision trees can hinder the learning performance given that the search space grows exponentially with each level. Further, deep splits that only affect a small number of datapoints can cause data fragmentation and overfitting [35, 21, 14]. Lastly, the exponential number of splits in decision trees restricts their applicability to memory-constrained hardware [35]. The sequential nature of decision-making in BDDs resembles that of decision trees, but unlike decision trees, the same split is used across a level, leading to a linear number of splits.

In this paper, we choose BDDs as our interpretable classifiers in order to emphasize compactness to an even greater degree compared to popular interpretable approaches, e.g. ones based on decision trees. Alongside the primary objective of accuracy, the size of a BDD classifier is also encoded and optimized. Our encoding for learning BDDs is inspired by the encoding of numerical branchings in Shati et al. [33, 34], which allows us to directly learn splits over numeric features without explicitly binarizing them. We then expand the notion of learning splits to multiple dimensions in order to learn solutions from a limited but distinctively interpretable family of diagrams.

The main contributions of this paper are as follows:

1. We present a novel SAT encoding of maximum accuracy BDD classifiers for numerical datasets. Our model represents a non-reduced BDD which can be reduced according to its sequence of terminals.
2. We extend our encoding by modelling the size of the final reduced BDD. The size encoding can be used as a secondary weighted objective, or it can be optimized in a second stage.
3. We present a variant of our encoding which supports direct learning of diagrams through multi-dimensional splits. Multi-dimensional BDDs enable learning more efficiently as they provide more expressive solutions within the same number of splits.
4. We run extensive experiments which demonstrate that our base encoding outperforms the state of the art in runtime and accuracy. Additionally, we show that explicitly optimizing size leads to significantly more compact solutions while maintaining high accuracy. Finally, we show that multi-dimensional BDDs scale better than BDDs due to their expressiveness and the size of their encoding.

## 2 Related Work

Decision tree classifiers are traditionally constructed via local search and heuristics [7, 29, 30]. Recent advances in tools and techniques for exact optimization have enabled approaches for finding optimal decision trees via branch-and-bound search [1], SAT-based encodings [33, 18, 3], Constraint Programming [37], or Mixed Integer Programming [5]. Exact optimization produces solutions that are optimal in accuracy, size, or both [33]. The size of a decision tree is often measured by its depth or number of nodes, which does not take into account the amount

of redundancy between nodes. Decision diagrams address redundancy directly by merging equivalent nodes. Approaches to learn optimal decision diagrams via exact optimization usually require a pre-determined skeleton as input. For example, Florio et al. [14] requires user-provided width for each level, limiting solutions but guaranteeing compactness and improving performance. Oblivious decision diagrams encourage compactness even further by requiring all splits of the same level to be the same, leading to only a linear number of splits instead of exponential as in ordinary decision trees.

Binary Decision Diagrams (BDD) are oblivious decision diagrams that can be reduced and ordered, making them ideal candidates to represent boolean functions. While BDDs are commonly used for hardware synthesis [8, 2, 28, 23], there has been a recent focus on utilizing BDD classifiers as interpretable solutions [16, 9]. For example, Hu et al. [16] uses MaxSAT to learn a maximum accuracy tree BDD which will then be reduced into a diagram. On the other hand, Cabodi et al. [9] considers a SAT-based approach for learning BDDs of minimum size that correctly classify all the training data.

## 3 Background

### 3.1 Binary Decision Diagrams

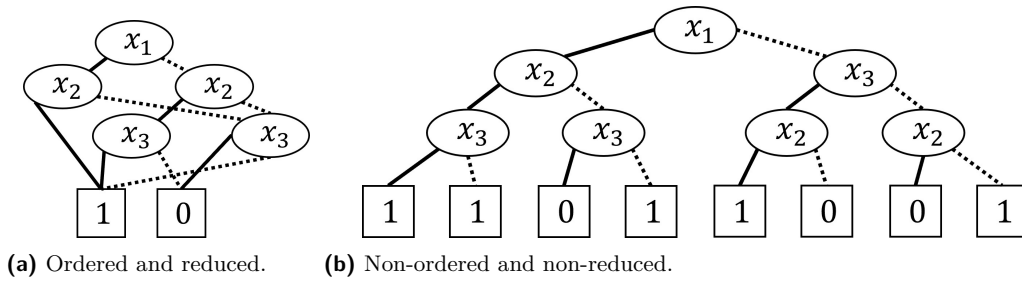
In this section, we first define standard Binary Decision Diagrams which can be used for binary classification. To support multi-class classification, we extend the BDD definition to its multi-terminal variant.

► **Definition 1** (Binary Decision Diagram). *Given a boolean function operating on boolean input, a Binary Decision Diagram (BDD) is a graph representation of the function. A BDD is a rooted, directed, and acyclic graph with sink nodes as terminals ( $N_T$ ) and the rest as decision nodes ( $N_D$ ). Each decision node is labelled with a split and has two outgoing edges, corresponding to the two (1/0) possible values of the boolean input. The terminal nodes are assigned output values 0 or 1. The size of a BDD is the number of its decision nodes.*

► **Definition 2** (Multi-Terminal Binary Decision Diagram). *Given a function operating on binary features with constant range  $[1..k]$ , a Multi-Terminal Binary Decision Diagram (MTBDD) is a graph representation of the function. An MTBDD is a BDD with its terminals supporting  $k$  output values instead of 2.*

Throughout the paper, we will simply refer to MTBDDs as BDDs to be more concise. We adopt standard BDD terminology with minor exceptions. As a concise way of addressing the nodes, we borrow from the decision tree literature to view the outgoing edges as parent-child relationships. Specifically, we address the node connected via the 1 (0) outgoing edge as the left (right) child. Moreover, we refer to the boolean inputs assigned to decision nodes as splits rather than features or variables, to avoid confusion with the features of our data and the variables of our encoding. Finally, considering that the features in our data are numerical rather than binary, we overload the definition of splits to represent pairs of numerical features and a valid threshold.

► **Definition 3** (Split). *Given a finite set of numerical features  $F$ , a split is a binarization of a numerical feature through pairing with a threshold value  $(f, \alpha) \in F \times \mathbb{R}$ . A split assigns a value of 1 to each point  $x$  that comes before the threshold ( $x[f] \leq \alpha$ ). Similarly, it assigns a value of 0 to each point  $x$  that comes after the threshold ( $x[f] > \alpha$ ).*



■ **Figure 1** Two BDDs representing the same boolean function. Solid (dotted) lines correspond to value 1 (0).

In order to calculate the value of a function for a given input using its BDD representation, we start from the root, move to the left (right) child if the input has value 1 (0) for the current split, and assign an output when a terminal is reached. A node is said to *contain* an input if the node is on the input's path from the root to a terminal.

The same function with binary splits can be represented with multiple BDDs. In order to uniquely represent a function, we define the BDD properties of being *ordered* and *reduced*. The ordered property enforces the same sequence of splits along each path, making BDDs practically equivalent to Oblivious Read-Once Decision Graphs (OODGs). The reduced property requires all equivalent parts of the BDD to be merged and all of the redundant parts to be removed. Given a function and an ordering of splits, there only exists one ordered and reduced BDD representing the function.

► **Definition 4** (Ordered BDD and Split Sequence). *A BDD is said to be ordered if the splits observed along every path from the root to a terminal respect a singular total ordering, called its split sequence.*

► **Definition 5** (Node Equivalency and Redundancy). *Given a binary decision diagram  $\mathcal{T}$  and two of its nodes  $t_1, t_2 \in \mathcal{N}$ ,  $t_1$  and  $t_2$  are equivalent if they are both decision nodes with the same split and the sub-graph of  $t_1$  and its descendants is isomorphic to the sub-graph of  $t_2$  and its descendants, or if they are both terminal nodes with the same label. Furthermore, a decision node  $t \in \mathcal{N}_D$  is redundant if its left and right children are equivalent.*

► **Definition 6** (Reduced BDD). *A BDD is said to be reduced if all of its equivalent nodes are merged and all of its redundant nodes are replaced with their children.*

We assume a BDD to be ordered and reduced unless noted otherwise. Figure 1 depicts two BDDs representing the same boolean function, one ordered and reduced and the other one not. It has been shown in the literature that independent of the merging and elimination process, the final result of reducing a BDD is always the same. We formally state the uniqueness of reduced BDDs in Proposition 7 and refer the reader to the BDD literature for more details [23].<sup>1</sup>

► **Proposition 7.** *Every function operating on splits and a given split sequence is represented by exactly one ordered and reduced BDD up to renaming.*

<sup>1</sup> Note that when employing standard BDD terminology, redundant and equivalent nodes are defined through binary sequences called *beads*. Beads correspond to nodes of the uniquely reduced BDD described in Proposition 7.

### 3.2 Weighted Partial MaxSAT

In this section, we describe the MaxSAT paradigm which we use to learn binary decision diagrams. A SAT formula is a conjunction of clauses, where each clause is a disjunction of literals, and each literal is either a Boolean variable or its negation. In a weighted partial MaxSAT instance, clauses are categorized into **hard** and **weighted soft** clauses. The goal is then to find a truth assignment to variables which satisfies all of the hard clauses and maximizes the total weight of the satisfied soft clauses.

## 4 MaxSAT Encoding of BDD Classifiers

In this section, we propose a MaxSAT encoding to find a BDD classifier with maximum accuracy. The inputs to our problem are: a training dataset  $X$  over a set of numerical features  $F$  and labels  $K$ , a ground-truth label  $y_i$  for each point  $x_i \in X$ , and a maximum number of splits  $s_{max}$ . The outputs are  $s_{max}$  splits  $\theta \in \{F \times \mathbb{R}\}^{s_{max}}$  with terminal labelling  $\gamma : \{0, 1\}^{s_{max}} \mapsto K$ . Similar to previous work of Hu et al. [16], we learn an ordered yet non-reduced (tree) BDD first. We then focus on merging and replacing nodes towards a reduced BDD. Inspired by the numerical branching in Shati et al. [33, 34], we encode splits (Definition 3) without the need for prior binarization of data which was shown to lead to significant performance degradation in decision trees. We instead directly encode how each split directs each point, while making sure the order of values is respected given a selected feature.

### 4.1 Variables

The following set of binary variables represents different aspects of modeling splits, labels, and accuracy in our BDD encoding.

- $a_{s,j}$ : The feature chosen at split  $s$  is or comes before  $j$ .
- $d_{s,i}$ : Point  $x_i$  is directed to the left child at split  $s$ .
- $c_{t,l}$ : Output label  $l$  is assigned to terminal node  $t$ .
- $o_i$ : Point  $x_i$  is classified correctly.

### 4.2 Clauses

We propose the following sets of Conjunctive Normal Form (CNF) clauses for modelling a non-reduced ordered BDD. The clauses in Eqs. (1-2) guarantee that one feature is selected at each split by enforcing the ordered encoding of  $a_{s,j}$  variables. The clauses in Eqs. (3-5) guarantee that for each split and chosen feature, the points directed to the left (resp. right) have a lesser (resp. greater) value compared to a threshold. The clauses in Eqs. (6-7) guarantee that one output label is chosen for each terminal node. The clauses in Eq. (8) guarantee that a point can only be considered classified if it ends up in a terminal node with the same label as its ground truth. We use  $O_j(X)$  to denote the set of all consecutive pairs when the members of  $X$  are sorted based on their  $j$  values,  $O_j^=(X)$  to denote the subset of pairs with equal  $j$  values, i.e.,  $O_j(X) \cap \{(i_1, i_2) \mid x_{i_1}[j] = x_{i_2}[j]\}$ , and  $\#_j^1$  to denote the index of the point with the smallest  $j$  value. Furthermore, we set  $a_{s,|F|}$  to the false constant and define  $A_R(t)$  ( $A_L(t)$ ) as the set of right (left) ancestors of terminal  $t$ .

$$(a_{s,j}, \neg a_{s,j+1}) \quad s < s_{max}, j \in F \quad (1)$$

$$(a_{s,0}) \quad s < s_{max} \quad (2)$$

$$(\neg a_{s,j}, a_{s,j+1}, d_{s,i_1}, \neg d_{s,i_2}) \quad s < s_{max}, j \in F, (i_1, i_2) \in O_j(X) \quad (3)$$

$$(\neg a_{s,j}, a_{s,j+1}, \neg d_{s,i_1}, d_{s,i_2}) \quad s < s_{max}, j \in F, (i_1, i_2) \in O_j^-(X) \quad (4)$$

$$(\neg a_{s,j}, a_{s,j+1}, d_{s,\#_j^1}) \quad s < s_{max}, j \in F \quad (5)$$

$$(\neg c_{t,l_1}, \neg c_{t,l_2}) \quad t \in \mathcal{N}_T, l_1, l_2 \in K \quad (6)$$

$$\left( \bigvee_{l \in L} c_{t,l} \right) \quad t \in \mathcal{N}_T \quad (7)$$

$$\left( \bigvee_{s \in A_L(t)} \neg d_{s,i}, \bigvee_{s \in A_R(t)} d_{s,i}, c_{t,y_i}, \neg o_i \right) \quad t \in \mathcal{N}_T, x_i \in X \quad (8)$$

In order to model the objective, we include the soft clauses in Eq. (9) with unit weights, which represent correctly classifying as many training points as possible.

$$(o_i) \quad x_i \in X \quad (9)$$

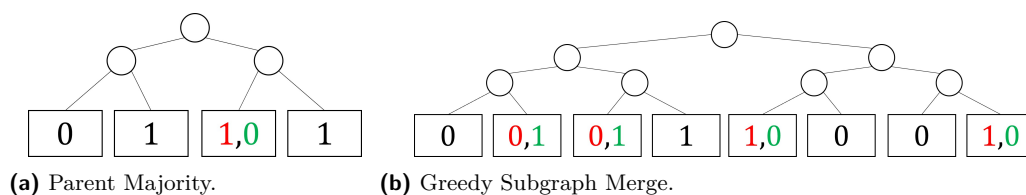
### 4.3 Decoding

An assignment to the variables in Section 4.1 which is satisfying with regard to the hard clauses in Section 4.2 is decoded into a reduced BDD in two steps. First, the assignment is decoded into a non-reduced BDD. Specifically, the labels of terminals are decoded from the selected labels ( $c_{t,l}$ ) and the sequence of splits are decoded from the pairings of selected features ( $a_{s,j}$  values) and thresholds ( $d_{s,i}$  values). Second, the resulting BDD is reduced by merging equivalent nodes and eliminating redundant nodes.

## 5 Size Optimization

In the encoding presented in Section 4, not all terminals are guaranteed to contain training points. Thus, we may end up with empty terminals, which we can relabel without affecting the training accuracy. However, the labels of such terminals can affect the prediction on unseen datapoints as well as the size of the final reduced BDD. In Hu et al. [16], the labels of empty terminals are decided arbitrarily by the solver and they investigate the impact of two post-processing relabelling heuristics on the testing accuracy (i.e., accuracy on unseen data). The first heuristic assigns the majority label of the terminal's first non-empty ancestor. The second heuristic finds and merges equivalent nodes in a greedy top-down search. Hu et al.'s experiments showed that the heuristics do not have significant impact on the testing accuracy. However, as we demonstrate in Figure 2a and Figure 2b, such heuristic approaches can increase the size of the final reduced BDDs.

In this section, we propose to use exact optimization for solving the same problem as in Section 4, with additional compactness considerations via deciding the labels of empty terminals. Specifically, we aim to model the size of the BDD after the merging of its equivalent nodes and the removal of its redundant nodes. The variables and clauses introduced in this section can either be added to the encoding in Section 4 as a secondary objective (i.e. 1-stage approach), or they can be used as a separate post-processing stage alongside variables  $c_{t,l}$  and the clauses in Eqs. (6-7) with the labels of non-empty terminals fixed (i.e. 2-stage approach).



**Figure 2** Examples for heuristics of assigning labels to empty terminals where size is increased. Black labels represent non-empty terminals, red labels are chosen by the heuristics, and green labels are the original labels which lead to a more compact solution.

### 5.1 Variables

The following set of binary variables represents uniqueness and repetition aspects of terminal labels, required for modelling the size of the ultimately reduced BDD.

- $\sigma_{t_1,t_2}$ : Terminals  $t_1$  and  $t_2$  have been assigned different output labels.
- $b_{t,\Delta}$ : The sequence of  $\Delta$  labels starting from terminal node  $t$  (inclusive) cannot be divided into two equal sub-sequences.
- $r_{t_1,t_2,\Delta}$ : The sequence of  $\Delta$  labels starting from terminal node  $t_1$  is equal to the sequence of  $\Delta$  labels starting from terminal node  $t_2$  (both inclusive).

Note that the terminal order which is referred to in the variable definitions, is the order of appearance in the depth-first search of the tree with the left outgoing edges prioritized. We say a sequence of terminals correspond to a decision node, if they are the sequence of terminals in the decision node’s sub-graph. Moreover, we say a single terminal coincides with a decision node if it marks the beginning the node’s sequence of terminals.

Note that variables  $\sigma_{t_1,t_2}$ ,  $b_{t,\Delta}$ , and  $r_{t_1,t_2,\Delta}$  are not defined for all possible index combinations. Specifically,  $\Delta$  always refers to the length of a sequence of terminals corresponding to a decision node (a power of 2), and can uniquely determine the node, if paired with a coinciding terminal. Consequently,  $b_{t,\Delta}$  is only used to represent that the decision node at level  $s_{max} - \log_2(\Delta) + 1$  coinciding with  $t$  is not redundant. Moreover,  $r_{t_1,t_2,\Delta}$  is only used to represent that the two decision nodes at level  $s_{max} - \log_2(\Delta) + 1$  coinciding with  $t_1$  and  $t_2$  are equivalent. Lastly,  $\sigma_{t_1,t_2}$  is only used when it affects the equivalency of two decision nodes.

### 5.2 Clauses

We first define two sets to help with the clause construction. The set  $P(s_{max})$  contains all possible lengths of terminal sequences corresponding to decision nodes except for the root (i.e., all  $2^{p'}$  where  $p' < s_{max}$ ). The set  $G(s_{max})$  contains all triples  $(t_1, t_2, \Delta)$  where comparing the labels of terminals  $t_1$  and  $t_2$  has impact on a decision node at level  $s_{max} - \log_2(\Delta) + 1$  being redundant.

$$G(1) = \{(0, 1, 2)\}$$

$$G(p) = G(p - 1) \cup \{(t_1 + 2^{p-1}, t_2 + 2^{p-1}, \Delta) | (t_1, t_2, \Delta) \in G(p - 1)\}$$

$$\cup \{(t, t + 2^{p-1}, 2^p) | 0 \leq t < 2^{p-1}\}$$

We propose the following sets of CNF clauses for modelling the size of a reduced BDD given its terminal labels. The clauses in Eqs. (10-12) guarantee that  $\sigma_{t_1,t_2}$  variables correctly represent the difference in labels. The clauses in Eq. (13) guarantee that each decision node

can only be redundant if all of the corresponding terminal pairs have the same label. The clauses in Eqs. (14-15) guarantee that a pair of decision nodes can only be equivalent if their corresponding sequence terminals have the same labels.

$$(\neg c_{t_1,l}, \neg c_{t_2,l}, \neg \sigma_{t_1,t_2}) \quad (t_1, t_2, \Delta) \in G(s_{max}), l \in K \quad (10)$$

$$(\neg c_{t_1,l}, c_{t_2,l}, \sigma_{t_1,t_2}) \quad (t_1, t_2, \Delta) \in G(s_{max}), l \in K \quad (11)$$

$$(c_{t_1,l}, \neg c_{t_2,l}, \sigma_{t_1,t_2}) \quad (t_1, t_2, \Delta) \in G(s_{max}), l \in K \quad (12)$$

$$(b_{\Delta[t_1/\Delta],\Delta}, \neg \sigma_{t_1,t_2}) \quad (t_1, t_2, \Delta) \in G(s_{max}) \quad (13)$$

$$(\neg r_{t_1\Delta,t_2\Delta,\Delta}, \neg c_{t_1\Delta+\delta,l}, c_{t_2\Delta+\delta,l}) \quad \Delta \in P(s_{max}), t_1 < t_2 < 2^{s_{max}}/\Delta, \delta < \Delta, l \in K \quad (14)$$

$$(\neg r_{t_1\Delta,t_2\Delta,\Delta}, c_{t_1\Delta+\delta,l}, \neg c_{t_2\Delta+\delta,l}) \quad \Delta \in P(s_{max}), t_1 < t_2 < 2^{s_{max}}/\Delta, \delta < \Delta, l \in K \quad (15)$$

To model our objective, we include the soft clauses in Eq. (16) which represent each decision node to be either redundant or equivalent to a previous one. Maximizing the number of redundant or equivalent decision nodes will consequently minimize the size of our reduced BDD. Note that if size is being considered as a secondary objective to accuracy, the clauses in Eq. (16) can be weighted against the clauses in Eq. (9) proportionately. Otherwise, if size is our second-stage objective, we can use the clauses in Eq. (16) with unit weights.

$$\left( \bigvee_{0 \leq t_2 < t} r_{t_2\Delta,t\Delta,\Delta}, \neg b_{t\Delta,\Delta} \right) \quad \Delta \in P(s_{max}), t < 2^{s_{max}}/\Delta \quad (16)$$

### 5.3 Decoding

We first show the soundness of the size encoding.

► **Proposition 8.** *Given a sequence of terminal labels  $c_{t,l}$ , the encoding in Sections (5.1,5.2) has an optimal objective value equal to the reduced size of an ordered tree BDD with the given terminals.*

**Proof Sketch.** Given the alternative interpretation of variables described in Section 5.2, we can conclude that the objective clauses in Eq. (16) aim to minimize the number of decision nodes that are not redundant and not equivalent to any of the nodes that come before them in the same level. Considering that equivalent decision node pairs can only appear in the same level, we can restate the objective as minimizing the number of decision nodes that remain when all of equivalent ones are merged and redundant ones are eliminated. According to Proposition 7, this objective will lead to a unique reduced BDD in its most compact form, proving that our encoding correctly models and minimizes the size of our solution. ◀

To decode the solution, we simply need to do as we did in Section 4.3, since the additional variables for size encoding are not involved in structuring the BDD. Once we decode and reduce the solution as before, we will end up with one decision node for every unsatisfied soft clause in Eq. (16).

## 6 MaxSAT Encoding of Multi-Dimensional BDDs

In Section 4 and Section 5, we consider learning BDDs by finding trees and reducing them to diagrams. However, the clauses in Eq.(8) are exponential in the number of splits and multiplied by the size of the dataset. Learning diagrams directly can help us consider more expressive solutions without increasing the number of splits, resulting in smaller encodings. In this section, we look at a family of diagrams that contain splits over multiple features at each level, i.e., multi-dimensional splits which themselves are BDDs with two labels.

► **Definition 9** (Directional Inner BDD). *Given a finite set of numerical features  $F$  and a dimension  $D$ , a directional inner BDD  $\mathcal{T}^{\leftrightarrow}$  is a BDD operating on  $D$  splits  $\theta^{\leftrightarrow} \in \{F \times \mathbb{R}\}^D$  with 0 and 1 as labels.*

A multi-dimensional BDD operates on multi-dimensional splits. A multi-dimensional split uses a directional inner BDD to direct points towards left (label 1) and right (label 0), rather than using a single split.

► **Problem 10** (Multi-dimensional BDD Learning Problem). *Given a finite set of numerical features  $F$ , a finite set of labels  $K$ , a set of training points  $x_i \in X$  with corresponding labels  $y_i \in K$ , a sequence of dimensions  $\mathcal{D} = \{D_0, D_1, D_{s_{max}-1}\}$ , and a number of multi-dimensional splits  $s_{max}$ , the goal is to find a sequence of directional inner BDDs  $\theta^M = \{\mathcal{T}_0^{\leftrightarrow}, \mathcal{T}_1^{\leftrightarrow}, \dots, \mathcal{T}_{s_{max}-1}^{\leftrightarrow}\}$  of respective dimensions  $D_0, D_1, \dots, D_{s_{max}-1}$  with terminal labelling  $\gamma^M : \{0, 1\}^{s_{max}} \mapsto K$  to construct a BDD with the directional inner BDDs as splits. The objective for learning Multi-dimensional BDDs is high accuracy.*

A multi-dimensional split is a generalization of an ordinary split, which makes a multi-dimensional BDD more expressive compared to an ordinary BDD with the same number of splits. Previous works have considered other representations that are designed to be more expressive than BDDs, such as sentential decision diagrams [12] or read- $k$ -times branching programs [6, 22]. Next, we formally compare the expressiveness of BDDs against their multi-dimensional variants in the other direction.

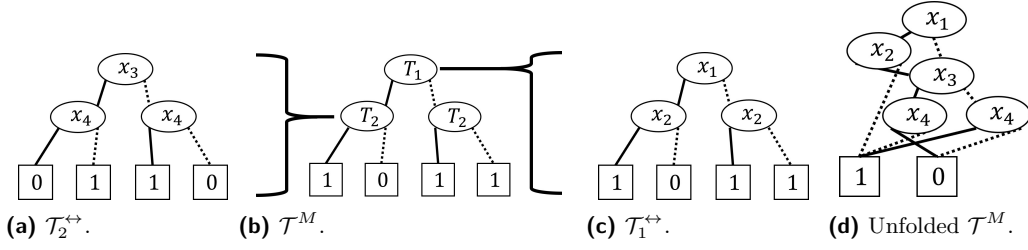
► **Theorem 11.** *Consider a multi-dimensional BDD  $\mathcal{T}^M$  operating on  $s_{max}$  directional inner BDDs  $\mathcal{T}_s^{\leftrightarrow}$  with split sequences  $\theta_s^{\leftrightarrow}$  and terminal labellings  $\gamma_s^{\leftrightarrow}$ , which itself has a terminal labelling  $\gamma^M$ . The binary function that  $\mathcal{T}^M$  represents is the same as the binary function represented by a BDD  $\mathcal{T}$  with split sequence  $\theta(\#(s, h)) = \theta_s^{\leftrightarrow}(h)$  and terminal labelling  $\gamma(t) = \gamma^M(\gamma_0^{\leftrightarrow}(t_0)\gamma_1^{\leftrightarrow}(t_1)\dots\gamma_{s_{max}-1}^{\leftrightarrow}(t_{s_{max}-1}))$  where  $\#(s, h)$  produces a complete ordering of  $S_H$  by concatenating the split sequences of each directional inner BDD, and  $t_0, t_1, \dots, t_{s_{max}-1}$  are sub-sequences of  $t$  divided based on the sequence of dimensions.*

**Proof Sketch.** The BDD using multi-dimensional splits can be transformed into a tree BDD with ordinary splits. In the transformation, every split of dimension  $D$  will correspond to  $D$  levels where nodes are expanded  $2^D$ -fold. With each expansion, we add annotations to the nodes specifying the label which they were assigned to by the corresponding directional inner BDD. Once all multi-dimensional splits are added, the final level of nodes are considered as terminals and are labelled according to how  $\gamma^M$  labels their annotations. Note that the resulting tree can be turned into a diagram by merging all of the nodes that have equal annotations and replacing all of redundant nodes with their children. ◀

We can conclude from Theorem 11, that multi-dimensional BDDs operating on dimensions  $\{D_0, D_1, \dots, D_{s_{max}-1}\}$  with  $D_{total} = \sum_{i < s_{max}} D_i$  are less expressive than BDDs operating on  $D_{total}$  splits. Combining with the aforementioned fact that multi-dimensional BDDs operating on  $\{D_0, D_1, \dots, D_{s_{max}-1}\}$  dimensions are trivially more expressive than BDDs operating on  $s_{max}$  splits, we have shown a lower and upper bound number of BDD splits when discussing the expressiveness of a multi-dimensional BDD.

Figure 3 shows how a multi-dimensional BDD can operate on two directional inner BDDs and how it can be unfolded to operate on ordinary splits.





■ **Figure 3** Multi-dimensional BDD  $\mathcal{T}^M$  operating on directional inner BDDs  $\mathcal{T}_1^{\leftrightarrow}$  and  $\mathcal{T}_2^{\leftrightarrow}$  and its unfolded version.

## 6.1 Variables

In order to encode a multi-dimensional BDD, we use the same set of variables as Section 4.1 but remove  $a_{s,j}$  variables since multi-dimensional splits require more features to be selected at each split. Furthermore, we add the following variables to encode the inner-workings of each directional inner BDD.

- $\hat{a}_{(s,h),j}$ : The feature chosen at split  $h$  of directional inner BDD  $s$  is or comes before  $j$ .
- $\hat{d}_{(s,h),i}$ : Point  $x_i$  is directed to left at split  $h$  of directional inner BDD  $s$ .
- $\hat{c}_{s,t}$ : Terminal  $t$  of directional inner BDD  $s$  is assigned the label 1.

## 6.2 Clauses

We discard clauses Eqs. (1-6) from the original encoding since splits are learned differently than before. We keep the clauses Eqs. (6-9) however, since we still need the labels and the appearance of points at terminals to be modelled correctly. Lastly, we add CNF clauses for modelling multi-dimensional splits, in order to complete our encoding of a multi-dimensional BDD.

The clauses in Eqs. (17-18) guarantee that one feature is selected at each split of each directional inner BDD by enforcing the ordered encoding of  $\hat{a}_{(s,h),j}$  variables. The clauses in Eqs. (19-21) guarantee that each split of each directional inner BDD conforms to a pairing of selected feature and threshold. The clauses in Eqs. (22-23) guarantee that the direction of a point in a multi-dimensional split matches the label of its containing leaf in the corresponding directional inner BDD. The clauses in Eq. (24) guarantee the leftmost leaf in each directional inner BDD to be assigned the 0 label (analogous to Eq. (5), see Shati et al. [33, 34] for further discussion). The set  $S_H$  contains all pairs of directional inner BDDs and their corresponding splits  $S_H = \{(s, h) \mid s < s_{max}, h < D_s\}$ ,  $\mathcal{N}_T^s$  contains the terminals of directional inner BDD  $s$ ,  $A_R(s, t)$  ( $A_L(s, t)$ ) contains the right (left) ancestors of terminal  $t$  within directional inner BDD  $s$ , and  $\hat{a}_{(s,h),|F|}$  is set to the false constant.

$$(\hat{a}_{(s,h),j}, \neg\hat{a}_{(s,h),j+1}) \quad (s, h) \in S_H, j \in F \quad (17)$$

$$(\hat{a}_{(s,h),0}) \quad (s, h) \in S_H \quad (18)$$

$$(\neg\hat{a}_{(s,h),j}, \hat{a}_{(s,h),j+1}, \hat{d}_{(s,h),i_1}, \neg\hat{d}_{(s,h),i_2}) \quad (s, h) \in S_H, j \in F, (i_1, i_2) \in O_j(X) \quad (19)$$

$$(\neg\hat{a}_{(s,h),j}, \hat{a}_{(s,h),j+1}, \neg\hat{d}_{(s,h),i_1}, \hat{d}_{(s,h),i_2}) \quad (s, h) \in S_H, j \in F, (i_1, i_2) \in O_j^-(X) \quad (20)$$

$$(\neg\hat{a}_{(s,h),j}, \hat{a}_{(s,h),j+1}, \hat{d}_{(s,h),\#j^1}) \quad (s, h) \in S_H, j \in F \quad (21)$$

$$\left( \bigvee_{h \in A_R(s,t)} \hat{d}_{(s,h),i}, \bigvee_{h \in A_L(s,t)} \neg\hat{d}_{(s,h),i}, d_{s,i}, \neg\hat{c}_{s,t} \right) \quad s \in S, x_i \in X, t \in \mathcal{N}_T^s \quad (22)$$

$$\left( \bigvee_{h \in A_R(s,t)} \widehat{d}_{(s,h),i}, \bigvee_{h \in A_L(s,t)} \neg \widehat{d}_{(s,h),i}, \neg d_{s,i}, \widehat{c}_{s,t} \right) \quad s \in S, x_i \in X, t \in \mathcal{N}_T^s \quad (23)$$

$$(\widehat{c}_{s,0}) \quad s \in S \quad (24)$$

### 6.3 Decoding

A satisfying assignment to the variables in Section 6.1 with regard to the clauses in Section 6.2, can be decoded into a tree BDD operating on multi-dimensional splits. In order to further transform the results into a BDD operating on 1-dimensional splits, we unfold the multi-dimensional splits and multiply the terminals according to the proof sketch for Theorem 11. Note that the number of splits in the resulting BDD is equal to the sum of dimensions from the multi-dimensional BDD.

The size optimization presented in Section 5 can also be utilized as a second stage after the solution is decoded. Specifically, we consider the multiplied sequence of terminal labels as input and treat empty terminals as before. In order to respect the structure of a multi-dimensional BDD, we also need to have additional clauses guaranteeing that multiplied instances of the same original terminal have the same label. Note that the size encoding cannot be employed in a 1-stage approach since the structure of the BDD is not yet determined at first.

## 7 Experiments

In this section, we perform studies to experimentally analyze the performance of our tree (Section 4), size (Section 5), and multi-dimensional (Section 6) encodings for learning BDDs. We compare the performance of our approach against state-of-the-art BDD learning baseline and investigate the trade-off between size, accuracy, and performance in 1-stage, 2-stage, and multi-dimensional approaches to learning compact diagrams. Throughout the experiments, we seek to find out whether compactness can be achieved without significant compromise, and investigate its impact on testing accuracy. Furthermore, we aim to understand if the added expressiveness of multi-dimensional BDDs allows us to achieve high quality solutions with lower number of splits.

### 7.1 Setup

We use the Java programming language to produce encodings and the Loandra MaxSAT solver [4] to solve each instance. We set the solver timeout limit to 15 minutes and use the best found solution in case of a timeout. Our experiments are run on an AMD EPYC 7502 32-core processor and 256GB of RAM.

### 7.2 Baseline

We compare our approach against the recent work on SAT-based learning of BDDs. Hu et al. [16] aims to find BDD classifiers similar to our approach. However, unlike our approach, they only support binary classification and require pre-processing of each numeric feature into a set of binary features. Furthermore, they do not explicitly model the size of the reduced BDD.

Note that Hu et al. [16, 17] have compared optimal BDDs against optimal decision trees and heuristic decision trees and found that BDDs are competitive in terms of testing accuracy and have smaller size. Our approach aims to improve the performance of Hu et al. and learn more compact BDD classifiers with comparable accuracy. We therefore compare our approach to Hu et al. in terms of runtime, accuracy, and size of BDDs.

### 7.3 Datasets

We run our experiments over a range of datasets from the UCI repository [13] covering different number of labels, both numerical and binary features, and different dataset sizes.

### 7.4 Results

#### Results on Comparing Our Base Encoding Against Hu et al. [16]

In our first set of results, we compare the performance of our approach in terms of runtime and solution quality against Hu et al. for different numbers of splits. Note that while our approach learns multi-terminal BDDs, Hu et al.’s approach learns standard BDDs and is unable to support datasets with more than two labels. Furthermore, since the objective in Hu et al. does not include compactness considerations, we use our base encoding that only optimizes accuracy as well (Section 4). As both approaches explore the same space of (feasible and) optimal BDD solutions, we focus our comparison on optimization performance (i.e., training accuracy and runtime). In contrast, in the next two sets of experiments, we will also evaluate the testing accuracy over 5-fold cross-validation.

Based on the results presented in Table 1, we see that our approach is able to achieve a higher than or equal to Hu et al. [16] accuracy in all but one case. Furthermore, the runtimes of non-timeout cases show that our approach can also prove optimality much faster. As we expected given our direct encoding of non-binary values, the improvement is most noticeable in datasets with highly numerical features, namely Banknote and Ionosphere.

#### Results on 1-Stage and 2-Stage Size Optimization

Next, we evaluate the encoding presented in Section 5 to minimize the size of our learned BDDs. We perform size optimization in 1-stage and 2-stage approaches. In the 1-stage approach, we use different values for the weight of the size objective against the accuracy objective. Given a weight  $\beta$  and  $s_{max}$  splits, the combined size and accuracy objective aims to find a solution  $f$  with maximum  $\beta(acc(f)) - (2^{s_{max}} - 1)|\mathcal{N}_D^f|$ , where  $\mathcal{N}_D^f$  is the set of decision nodes in the reduced version of  $f$ . We use  $\beta = \infty^2$  to denote the 2-stage approach,  $\beta = \infty^1$  to denote that accuracy is completely prioritized over size in the 1-stage approach, and Def. to denote the approach without any size optimization. Note that even in the 1-stage approach, running the second stage can further optimize the size if the first stage have yielded a sub-optimal solution (timeout). However, we opt against mixing the two approaches for the sake of clarity in comparison between the two. We use 5-fold cross validation and report the average value across folds to understand the effects of compactness on generalization and testing accuracy.

The results are presented in Figure 4. As expected, we see increase in training accuracy and size as we shift the priority to accuracy by changing the  $\beta$  value from 1 to  $\infty^2$ . However, the improvement in accuracy stagnates while the size continues to grow, indicating that a large enough  $\beta$  value can act as complete prioritization of accuracy. Interestingly, testing accuracy stagnates sooner and suffers from a larger variability as  $\beta$  increases, demonstrating a limited but positive effect of compactness on testing accuracy and generalization.

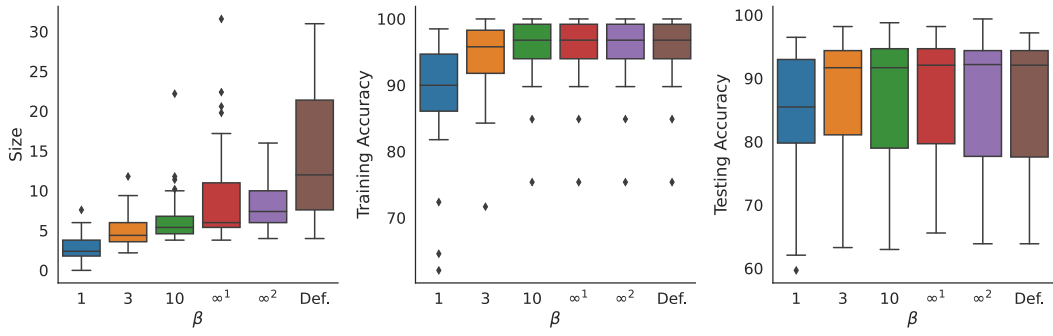
The solutions for lower  $\beta$  values in the 1-stage approaches are significantly smaller than the  $\beta = \infty^2$  case, highlighting the importance of adding size considerations while the solution is being learned. However, optimizing size as a second stage still provides a significant size reduction compared to the case with no size optimization. For a detailed report of the results per dataset, we refer the reader to Table 2 in Appendix A.1.

■ **Table 1** Results for learning max-accuracy BDDs.

Dataset	Splits	Accuracy (%)		Time (s)	
		Ours	Hu et al. [16]	Ours	Hu et al. [16]
Banknote	4	96.8	96.8	843.01	TO
X   F   K	5	97.6	90.4	TO	TO
1372 4 2	6	98.5	91.8	TO	TO
Breast	4	89.7	89.7	TO	TO
X   F   K	5	92.2	91.4	TO	TO
116 9 2	6	94.8	94.8	TO	TO
Cryotherapy	4	97.8	97.8	1.23	2.43
X   F   K	5	98.9	98.9	3.97	9.41
90 6 2	6	100	100	0.44	1.64
Immunotherapy	4	95.6	95.6	8.18	26.65
X   F   K	5	96.7	96.7	74.08	290.99
90 7 2	6	97.8	97.8	433.35	TO
Ionosphere	4	94.9	90.6	TO	TO
X   F   K	5	95.2	85.8	TO	TO
351 34 2	6	96.6	90.6	TO	TO
Iris	4	98.7	-	0.67	-
X   F   K	5	99.3	-	0.62	-
150 4 3	6	100	-	0.43	-
User	4	94.2	-	54.57	-
X   F   K	5	95.7	-	828.34	-
258 5 4	6	97.7	-	TO	-
Vertebral	4	88.1	87.7	TO	TO
X   F   K	5	89.7	90	TO	TO
310 6 2	6	91	90.3	TO	TO
Wine	4	99.4	-	29.29	-
X   F   K	5	100	-	2.07	-
178 13 3	6	100	-	1.14	-
Car	4	92.5	92.5	316.88	TO
X   F   K	5	92.9	92.9	TO	TO
1728 6 2	6	95.4	95.4	TO	TO
Monk2	4	74.6	74.6	85.49	473.43
X   F   K	5	84.6	84.6	185.3	416.37
169 6 2	6	100	100	0.35	1.09

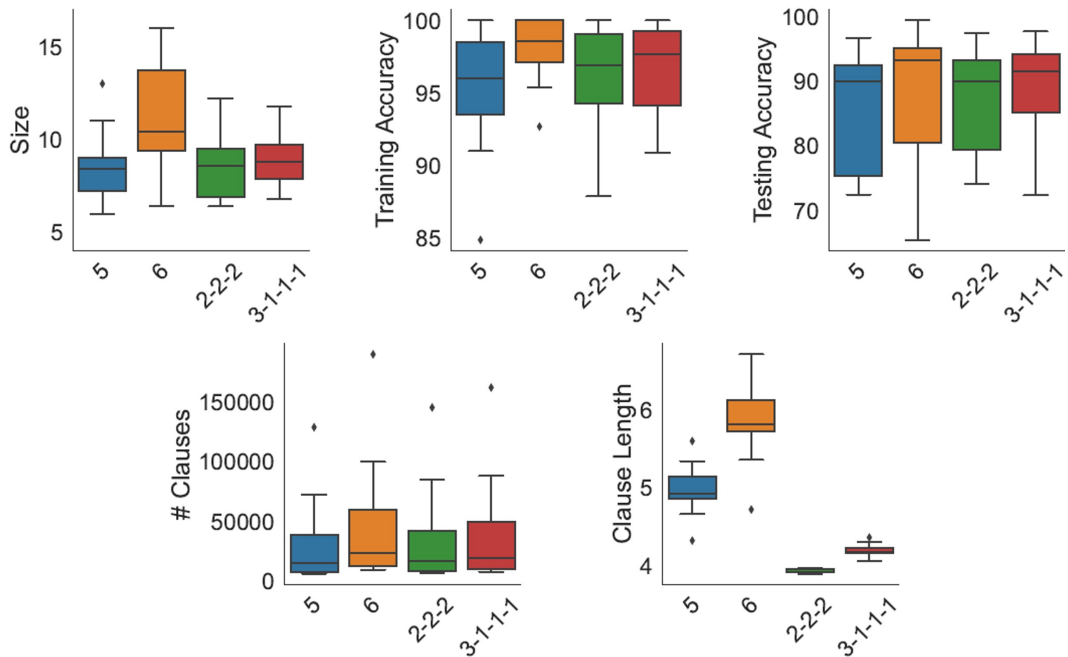
## Results on Learning Multi-Dimensional BDDs

In our next set of experiments, we evaluate our approach for directly learning multi-dimensional BDDs. Our goal is to understand whether the added expressiveness of multi-dimensional BDDs allows us to find high quality solutions with a lower number of splits. Furthermore, we aim to study the effects of dimension division for multi-dimensional BDDs



■ **Figure 4** Distributional result of size, training accuracy, and testing accuracy across datasets for different  $\beta$  values.

by considering one balanced diagram with three multi-dimensional splits (2-2-2) and one unbalanced diagram with one 3-dimensional split followed by three 1-dimensional splits (3-1-1-1). Finally, we have used the size encoding in Section 5 to optimally decide the labels of empty terminals towards compactness in a second stage. Note that the size of a multi-dimensional BDD is considered to be its number of decision nodes ( $|\mathcal{N}_D^f|$ ) after it is unfolded and reduced.

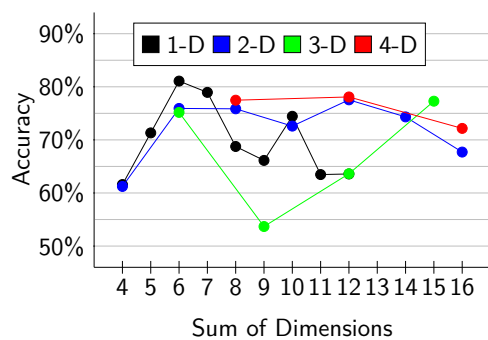


■ **Figure 5** Distributional result of size, accuracy, number of clauses, and average clause length for BDDs of two number of splits and Multi-dimensional BDDs of two dimensions.

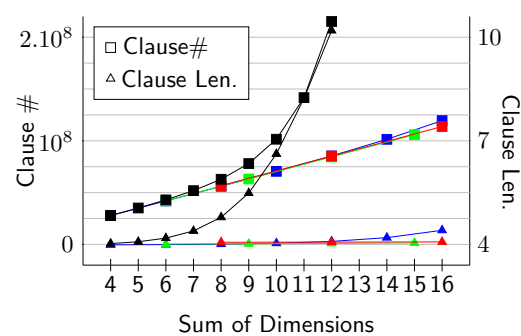
The results of the third set of experiments presented in Figure 5 show the two multi-dimensional approaches achieve training accuracy that is higher than a one-dimensional diagram with 5 splits and lower than a one-dimensional diagram with 6 splits. Since the sum of dimensions equal 6 both case, the upper bound is expected according to Theorem 11. However, the comparison against 5 splits, shows that the multi-dimensional approaches

perform better than the theoretical lower bound (resp. 3 and 4 splits) in practice. The same comparison is observed in testing accuracy and size. However, the 3-1-1-1 approach is able to achieve smaller variability in testing accuracy while still being close to the best approach in size. Finally, we see a lower number of clauses and a significantly shorter average clause length for our multi-dimensional approaches against ordinary BDDs of similar expressiveness. We refer the reader to Table 3 in Appendix A.1 for the complete results of this experiment with a larger set of dimension sequences considered.

Next, we run experiments for a significantly larger dataset, namely the Adult dataset ( $|X| = 32561$ ,  $|F| = 105$ ,  $|K| = 2$ ), to investigate the difference in encoding size and performance between one-dimensional and multi-dimensional approaches on large datasets. Given the scale of the tasks, we increase the timeout limit to 60 minutes. We compare, one-dimensional BDDs against ones with 2, 3, and 4-dimensional splits according to their total number of dimensions in Figure 6 and Figure 7. Figure 7 depicts the exponential growth of encoding size for the one-dimensional approach. The training accuracy presented in Figure 6 shows that the exponential size causes the ordinary approach to decrease in quality and finally cease to produce any solutions due to memory overflow. We observe that by using two dimensional splits, we maintain a higher accuracy over a significantly larger number of dimensions. Three-dimensional and four-dimensional splits prove to be more challenging compared to two-dimensional splits. However, we are still able to find solutions for higher total number of dimensions compared to the one-dimensional splits. In this experiment, we avoid 5-fold cross-validation as we focus on the optimization performance of the two methods.



■ **Figure 6** Training accuracy for the Adult dataset for different dimensions and number of total splits.



■ **Figure 7** Number and average length of clauses of the encoding for the Adult dataset for different dimensions and number of total splits.

## 8 Conclusion

In this paper, we present a novel MaxSAT encoding for learning Binary Decision Diagrams. Our BDD encoding represents a tree which can be reduced to an equivalent diagram. We extend our encoding with optimization for the size of the reduced diagram. The size objective can be balanced against accuracy in a 1-stage approach or optimized as a second stage. Furthermore, we present a variant of our encoding using multi-dimensional splits, which are inner BDDs themselves. Our experiments show that we outperform the state-of-the-art SAT-based BDD learning baseline due to our direct encoding of numerical splits. We further show that our 1-stage and 2-stage size optimization approaches lead to significantly more compact solutions while maintaining testing accuracy. Finally, we show that the expressiveness of our multi-dimensional BDDs allows us to produce high quality solutions in smaller number of splits, mitigating the exponential growth in the size of the encoding.

Our work, can be extended in a number of ways. Our size encoding for the 2-stage optimization can be extended by allowing more than empty terminal labels to be changed, e.g., feature ordering or splits. Moreover, nested directional BDDs can be added to our multi-dimensional BDD encoding to improve expressiveness and avoid exponentiation even further. Other interesting directions for future work involve investigating different strategies for balancing the two objectives, e.g., producing Pareto optimal solutions, or a more comprehensive analysis of parameters including but not limited to the dimension sequences. Finally, investigating the impact of more expressive BDDs such as free BDDs, which are not constrained to be ordered [32], is also an interesting direction for future research.

---

## References

- 1 Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 3146–3153, 2020.
- 2 Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on computers*, 27(06):509–516, 1978.
- 3 Florent Avellaneda. Efficient inference of optimal decision trees. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 3195–3202, 2020.
- 4 Jeremias Berg, Emir Demirović, and Peter J Stuckey. Core-boosted linear search for incomplete MaxSAT. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, pages 39–56. Springer, 2019.
- 5 Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.
- 6 Allan Borodin, Alexander Razborov, and Roman Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3:1–18, 1993.
- 7 Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.
- 8 Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.
- 9 Gianpiero Cabodi, Paolo E Camurati, Alexey Ignatiev, Joao Marques-Silva, Marco Palena, and Paolo Pasini. Optimizing binary decision diagrams for interpretable machine learning classification. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1122–1125. IEEE, 2021.
- 10 Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.
- 11 Edmund M Clarke, Masahiro Fujita, and Xudong Zhao. Multi-terminal binary decision diagrams and hybrid decision diagrams. *Representations of discrete functions*, pages 93–108, 1996.
- 12 Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.
- 13 Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL: <http://archive.ics.uci.edu/ml>.
- 14 Alexandre M Florio, Pedro Martins, Maximilian Schiffer, Thiago Serra, and Thibaut Vidal. Optimal decision diagrams for classification. *arXiv preprint arXiv:2205.14500*, 2022.
- 15 Oktay Günlük, Jayant Kalagnanam, Minhan Li, Matt Menickelly, and Katya Scheinberg. Optimal decision trees for categorical data via integer programming. *Journal of Global Optimization*, pages 1–28, 2021.
- 16 Hao Hu, Marie-José Huguët, and Mohamed Siala. Optimizing binary decision diagrams with maxsat for classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36(4), pages 3767–3775, 2022.

- 17 Hao Hu, Marie-José Huguet, and Mohamed Siala. Optimizing binary decision diagrams with maxsat for classification, 2022. [arXiv:2203.11386](https://arxiv.org/abs/2203.11386).
- 18 Hao Hu, Mohamed Siala, Emmanuel Hébrard, and Marie-José Huguet. Learning optimal decision trees with MaxSAT and its integration in AdaBoost. In *International Joint Conference on Artificial Intelligence and Pacific Rim International Conference on Artificial Intelligence (IJCAI-PRICAI)*, 2020.
- 19 Alexey Ignatiev and Joao Marques-Silva. SAT-based rigorous explanations for decision lists. In *Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24*, pages 251–269. Springer, 2021.
- 20 Alexey Ignatiev, Filipe Pereira, Nina Narodytska, and Joao Marques-Silva. A SAT-based approach to learn explainable decision sets. In *Automated Reasoning: 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings 9*, pages 627–645. Springer, 2018.
- 21 Dmitry Ignatov and Andrey Ignatov. Decision stream: Cultivating deep decision trees. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 905–912. IEEE, 2017.
- 22 Stasys Jukna. A note on read-times branching programs. *RAIRO-Theoretical Informatics and Applications*, 29(1):75–83, 1995.
- 23 Donald E Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 2009.
- 24 Ron Kohavi. Bottom-up induction of oblivious read-once decision graphs. In *Machine Learning: ECML-94: European Conference on Machine Learning Catania, Italy, April 6–8, 1994 Proceedings 7*, pages 154–169. Springer, 1994.
- 25 Benjamin Letham, Cynthia Rudin, Tyler H McCormick, and David Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model, 2015.
- 26 Breiman LI, Jerome Friedman, RA Olshen, and C.J. Stone. Classification and regression trees (CART). *Biometrics*, 40:358, September 1984. doi:10.2307/2530946.
- 27 Joao Marques-Silva and Alexey Ignatiev. Delivering trustworthy AI through formal XAI. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36(11), pages 12342–12350, 2022.
- 28 Bernard ME Moret. Decision trees and diagrams. *ACM Computing Surveys (CSUR)*, 14(4):593–623, 1982.
- 29 J Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
- 30 J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.
- 31 Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *nat mach intell* 1: 206–215. DOI: <https://doi.org/10.1038/s42256-019-0048-x>, 2019.
- 32 Petr Savický and Ingo Wegener. Efficient algorithms for the transformation between different types of binary decision diagrams. *Acta Informatica*, 34(4):245–256, 1997.
- 33 Pouya Shati, Eldan Cohen, and Sheila McIlraith. SAT-based approach for learning optimal decision trees with non-binary features. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- 34 Pouya Shati, Eldan Cohen, and Sheila A. McIlraith. SAT-based optimal classification trees for non-binary data. *Constraints*, July 2023. doi:10.1007/s10601-023-09348-1.
- 35 Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. Decision jungles: Compact and rich models for classification. *Advances in neural information processing systems*, 26, 2013.
- 36 Hazem Torfah, Shetal Shah, Supratik Chakraborty, S Akshay, and Sanjit A Seshia. Synthesizing pareto-optimal interpretations for black-box models. In *2021 Formal Methods in Computer Aided Design (FMCAD)*, pages 153–162. IEEE, 2021.



- 37 H elene Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus. Learning optimal decision trees using constraint programming. *Constraints*, 25(3):226–250, 2020.
- 38 Yu Zhang, Peter Ti no, Ale s Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742, 2021.

## A Appendix

### A.1 Additional Experimental Results

■ **Table 2** Average 5-fold results for learning BDDs with the combined objective of accuracy and compactness.

D.S.	Splits	Size					Training Acc. (%)					Testing Acc. (%)				
		$\beta$	1	3	10	$\infty^1$	$\infty^2$	1	3	10	$\infty^1$	$\infty^2$	1	3	10	$\infty^1$
Banknote	4	1	3.6	4	4	4.4	85.3	95.8	96.8	96.8	96.8	85.2	95	96.6	96.6	96.4
	5	3	4	5	11	6	94	96.8	97.6	97.5	97.6	93.7	96.4	96.9	96.8	96.7
	6	4	6	5.8	19.8	6.4	96.8	98.6	98.4	98.6	98.6	96.5	97.4	97.4	97.2	97.2
Breast	4	1	4	5.2	6.4	6.8	72.4	88.8	90.7	90.7	90.7	71.5	75.9	75.9	78.4	77.6
	5	3.8	5.4	11.4	11.8	11	88.8	91.8	94	94	94	74.1	70.7	74.1	72.4	72.5
	6	5.6	9.4	22.2	22.4	16	92.9	96.1	97	97.2	97.2	75	71.6	63	71.6	65.5
Cryo.	4	1	3	4.8	4.8	6	86.1	94.7	97.8	97.8	97.8	75.6	88.9	92.2	92.2	94.4
	5	2.4	5.2	5.8	5.8	8	93.1	99.2	99.4	99.4	99.4	83.3	94.4	91.1	91.1	90
	6	4.6	5.8	5.8	5.8	10.4	98.3	100	100	100	100	95.6	87.8	90	90	82.2
Immuno.	4	1	2.2	5.4	5.4	6.8	87.2	91.4	95.6	95.6	95.6	82.2	83.3	83.3	83.3	85.6
	5	1.6	4.4	5.4	5.4	8.4	89.7	95.3	96.7	96.7	96.7	81.1	81.1	78.9	81.1	73.3
	6	3.8	7.2	10	10	14	94.7	98.3	99.2	99.2	99.2	80	76.7	73.3	72.2	75.6
Iono.	4	1.8	2.4	4	5.6	4	90	92	95.2	94.9	95.2	87.2	90.9	90.9	90.3	92.3
	5	2	3.6	5.2	10.2	8.4	91.2	94.4	95.9	95.7	95.9	89.7	91.7	89.5	85.8	89.2
	6	2.4	6	7	17.2	12.8	92	96.3	96.9	97.2	97.1	90.9	88.6	88	86.9	86.3
Iris	4	2	2.4	3.8	3.8	4.2	96.3	97.3	98.8	98.8	98.8	94	94	94.7	94.7	94
	5	2	2.8	5.2	5.2	6	96.3	98	99.5	99.5	99.5	94	95.3	96.7	96	96
	6	2.8	5.4	5.8	5.8	7	98	99.8	100	100	100	95.3	95.3	94.7	96	94.7
User	4	2.2	4.4	6	6	6	82.8	91.5	94.3	94.3	94.3	79.8	87.6	93.4	93.4	93.4
	5	4	6	6.2	9.2	7	90.5	95.9	96	96	96	86	93	91.9	93.8	92.2
	6	6	7	8	31.6	9.2	96.2	97.2	97.8	97.8	97.8	93	93	95.7	95.7	95.4
Vertebral	4	1.2	2.4	4	5.4	4.6	81.8	86.8	89.8	89.8	89.8	76.8	76.5	79	78.1	77.7
	5	2.2	4.2	5.2	10.6	8.6	86.1	89.4	89.8	90.8	91	74.8	81.6	81.9	79.7	76.1
	6	3.4	6.2	10.2	20.6	15	88.5	91.6	92.4	92.8	92.7	80.3	80.3	79	76.8	78.7
Wine	4	2.2	3	4	5	6.4	93.8	98.5	99.6	99.9	99.9	87.7	96.6	93.3	93.3	93.9
	5	3	3.6	4.6	4.6	9.4	98.5	99.3	100	100	100	95	95	96.7	95.5	92.7
	6	3	4.6	4.6	4.6	10	98.5	100	100	100	100	95	93.3	92.2	93.3	93.3
Car	4	2	4	4	4	5.4	85.5	92.5	92.5	92.5	92.5	85.5	92.5	92.5	92.5	92.5
	5	3.2	4	4.6	6.8	7.4	90	92.5	92.9	93.1	93.1	88.6	92.5	91.7	92.1	92.1
	6	4	6.6	6.8	12.2	9.6	89	95	95	95.4	95.4	87.6	93.8	93.9	94.7	94.7
Monk2	4	0	3.4	5.4	5.4	6.6	62.1	71.7	75.4	75.4	75.4	62.1	63.3	64.5	65.6	63.9
	5	0.6	7.8	9	9	13	64.6	84.3	84.9	84.9	84.9	59.7	79.3	76.4	77.6	74.6
	6	7.6	11.8	11.8	11.8	13.4	86.2	100	100	100	100	81.7	98.2	98.8	98.2	99.4

■ **Table 3** Results for learning max-accuracy multi-dimensional BDDs.

Dataset	Dimensions	Accuracy (%)		Size		Time (s)
		Training	Testing	Stage 1	Stage 2	
Banknote	2-2-2	<b>98.6</b>	97.5	7.4	<b>6.6</b>	TO
	3-3	98.4	97.5	<b>6.6</b>	<b>6.6</b>	TO
	1-2-3	<b>98.6</b>	96.9	8.2	6.8	TO
	3-1-1-1	<b>98.6</b>	<b>97.7</b>	8.8	6.8	TO
Breast	2-2-2	<b>94.6</b>	<b>74.1</b>	<b>8.6</b>	<b>8.6</b>	TO
	3-3	92.9	64.6	8.8	8.8	TO
	1-2-3	94.2	68.1	9.6	9.6	TO
	3-1-1-1	94.2	72.4	11	9.8	TO
Cryotherapy	2-2-2	99.7	86.7	10.6	10.2	<b>6.18</b>
	3-3	99.4	<b>92.2</b>	10.6	10	25.29
	1-2-3	99.7	90	10.6	9.8	13.66
	3-1-1-1	<b>100</b>	86.7	<b>10</b>	<b>9.4</b>	6.45
Immunotherapy	2-2-2	97.2	75.6	<b>9.2</b>	<b>8.6</b>	515.83
	3-3	96.9	76.7	10.2	10.2	750.07
	1-2-3	97.2	76.7	11.6	11.2	767.69
	3-1-1-1	<b>98.1</b>	<b>84.4</b>	9.8	9.6	<b>489.51</b>
Ionosphere	2-2-2	96.9	90	<b>6.6</b>	<b>6.6</b>	TO
	3-3	95.4	88.3	8.8	8.8	TO
	1-2-3	<b>97</b>	90.3	7.4	7.4	TO
	3-1-1-1	96.7	<b>91.5</b>	8.2	8.2	TO
Iris	2-2-2	99.5	<b>95.3</b>	<b>10.4</b>	8.4	1.17
	3-3	99.5	94.7	11.2	11.2	0.67
	1-2-3	99.8	94	14.6	<b>8</b>	1.2
	3-1-1-1	<b>100</b>	93.3	12.4	8.8	<b>0.44</b>
User	2-2-2	95.6	91.1	<b>12</b>	11.6	TO
	3-3	93	88.7	12.2	12.2	<b>143.96</b>
	1-2-3	<b>97.7</b>	<b>96.1</b>	18.2	18.2	494.02
	3-1-1-1	<b>97.7</b>	95.4	13.6	<b>9.8</b>	783.39
Vertebral	2-2-2	91.3	79.4	7.6	<b>7.2</b>	TO
	3-3	<b>91.8</b>	<b>80</b>	<b>7.4</b>	7.4	TO
	1-2-3	91.1	79.4	9.4	8.2	TO
	3-1-1-1	90.9	77.4	10.6	8.6	TO
Wine	2-2-2	<b>100</b>	93.9	<b>9</b>	8.8	3.43
	3-3	<b>100</b>	90.5	9.6	9.6	<b>1.03</b>
	1-2-3	<b>100</b>	93.3	14.2	11.6	5.29
	3-1-1-1	<b>100</b>	<b>95</b>	<b>9</b>	<b>7.6</b>	9.03
Car	2-2-2	94	92.5	6.4	6.4	TO
	3-3	93	92.3	6.4	6.4	TO
	1-2-3	92.8	92.6	<b>6.2</b>	<b>5.8</b>	TO
	3-1-1-1	<b>94.1</b>	<b>93.2</b>	6.8	6.8	TO
Monk2	2-2-2	87.9	79.3	12.2	12.2	TO
	3-3	89.6	80.5	<b>10.6</b>	<b>10.6</b>	TO
	1-2-3	90.1	74.6	13	13	TO
	3-1-1-1	<b>92.9</b>	<b>85.8</b>	11.8	11.8	<b>695.58</b>



# Constraint Programming with External Worst-Case Traversal Time Analysis

Pierre Talbot ✉ 

University of Luxembourg, Luxembourg  
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg

Tingting Hu ✉

University of Luxembourg, Luxembourg

Nicolas Navet ✉ 

University of Luxembourg, Luxembourg

---

## Abstract

The allocation of software functions to processors under compute capacity and network links constraints is an important optimization problem in the field of embedded distributed systems. We present a hybrid approach to solve the allocation problem combining a constraint solver and a worst-case traversal time (WCTT) analysis that verifies the network timing constraints. The WCTT analysis is implemented as an industrial black-box program, which makes a tight integration with constraint solving challenging. We contribute to a new multi-objective constraint solving algorithm for integrating external under-approximating functions, such as the WCTT analysis, with constraint solving, and prove its correctness. We apply this new algorithm to the allocation problem in the context of automotive service-oriented architectures based on Ethernet networks, and provide a new dataset of realistic instances to evaluate our approach.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Computer systems organization → Real-time systems; Networks → Network performance evaluation

**Keywords and phrases** Constraint programming, external function, multi-objective optimization, network analysis, worst-case traversal time analysis, abstract interpretation

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.34

**Supplementary Material Software:** <https://github.com/ptal/automotive-network-cp/tree/cp2023>, archived at `swb:1:dir:e0fe246fdd3c6654293c9f5183cae7782b540d07`

**Funding** *Pierre Talbot:* This work is supported by the Luxembourg National Research Fund (FNR) – COMOC Project, ref. C21/IS/16101289.

**Acknowledgements** We are grateful to the reviewers for their detailed comments.

## 1 Introduction

The hardware architecture of automobiles consists of dozens of interconnected electronic control units (ECUs). An important optimization problem in the field of distributed embedded system, called the *deployment problem*, is to allocate software functions to the ECUs without overloading their compute capacity and overloading the network communication links. Worst-case traversal time analysis (WCTT) is critical to ensure the communications among software functions meet hard deadlines. In most works on the deployment problem, the network considered is a controller area network (CAN), whose operating principles are relatively simple and for which an exact WCTT analysis is available [7, 39]. Therefore, the constraint model can specify both the allocation problem and the WCTT analysis. However, the newest automotive electrical-electronic (E/E) architectures rely on high-speed switched Ethernet networks. WCTT analysis already exists for Ethernet networks but is much harder to model



© Pierre Talbot, Tingting Hu, and Nicolas Navet;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 34; pp. 34:1–34:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

as a constraint problem, especially considering the wealth of complicated quality-of-service mechanisms available in the time-sensitive networking standards (TSN, see [22]) that are used on top of standard Ethernet.

Constraint programming is a declarative paradigm for solving combinatorial problems. In general, the solvers are designed to work with a closed-world assumption, that is, they do not interact with external entities during solving. However, in practice, there are often parts of the model that are difficult to express as constraints, and are already programmed in another language. This is the case of the WCTT analysis. Industrial constraint solvers such as IBM ILOG CP OPTIMIZER<sup>1</sup> and LOCAL SOLVER<sup>2</sup> both propose *black-box expressions* to plug external functions in the model. However, these extensions are “last resort solutions” as they do not come with a semantics, and there is no guarantee on when the function is called, and how it is used within the solver. We contribute to a rigorous approach to this problem when the external function is under-approximating, i.e., it only produces valid solutions but not necessarily all.

In this work, we integrate a constraint solver for the allocation problem and a WCTT analyser for the timing constraints. We propose a general framework, based on abstract interpretation [6], for integrating external under-approximating functions (here, the WCTT analyser) in a constraint solving algorithm, and prove its correctness. The under-approximating external function validates each solution produced by the constraint solver. Moreover, when the external function can explain its failure, we dynamically add a new constraint to the constraint model, approximating the reason of the failure of the external function, to improve the quality of the subsequent solutions. In the following, we call these constraints *conflicts*<sup>3</sup>. Because the WCTT analysis is a black-box function, deriving useful *over-approximating conflicts* – which do not remove solutions from the problem, but might accept non-solutions – can be difficult, or even impossible depending on the information provided by the analyser. Our main contribution is to propose CUSOLVE\_MO a multi-objective constraint solving algorithm that is *over-approximating* even if the generated conflicts are not over-approximating. This algorithm extends the well-known multi-objective constraint programming algorithm of Gavanelli [13] which has been frequently used in constraint optimization [19, 31, 14]. Further, our framework can be used on top of any constraint solvers. Finally, we contribute to a new set of benchmarks for the deployment problem and evaluate our solving algorithms on them.

## 2 Service Deployment Problem

For the sake of conciseness, we present the *service deployment problem* in mathematical notation. In Appendix A, we give the constraint model in the MINIZINC constraint modelling language [25]. The MINIZINC model is very close from the mathematical definition given here and does not contain any particular modelling trick.

Let  $\langle H, L, hc, lc \rangle$  be a weighted graph where  $H$  is a set of hardware units connected by communication links  $L \subseteq H \times H$ . Moreover, each unit  $h \in H$  has a compute capacity  $hc(h)$  and each link  $\ell \in L$  has a link capacity  $lc(\ell)$ . This graph represents a network of connected heterogeneous hardware units such as processors and switches.

<sup>1</sup> <https://www.ibm.com/docs/en/icos/20.1.0?topic=2010-cp-optimizer-black-box-expressions>

<sup>2</sup> <https://www.localsolver.com/docs/last/modelingfeatures/externalfunctions.html>

<sup>3</sup> We avoid using the terminology of *nogood* because as we will see later, these conflicting constraints might not always preserve all solutions of the problem (over-approximating).

Let  $\langle S, Com, sc, cc \rangle$  be a weighted graph where  $S$  is a set of software functions that we call *services* and  $Com \subseteq S \times S$  is the set of communications between the services. Each service  $s \in S$  consumes a certain amount of computational power  $sc(s)$  and for any communication  $c \in Com$ ,  $cc(c)$  represents the network utilization due to this communication.

The core of the *service deployment problem* is to find a deployment function  $d : S \rightarrow H$  allocating each service on a processor. We illustrate this problem in Figure 1 where the software graph (Figure 1b) must be deployed on the hardware graph (Figure 1a) while satisfying a number of constraints – several solutions to this particular instance are shown on Figure 4. The first constraint is on the compute capacity of each processor:

$$\forall h \in H, \quad \sum_{s \in d^{-1}(h)} sc(s) \leq hc(h)$$

It guarantees that the sum of the computational power required by all services allocated on processor  $h$  does not exceed the compute capacity of  $h$ .

The second constraint is on the communication network:

$$\forall \ell \in L, \quad \sum_{c \in Com} com(c, \ell) \leq lc(\ell)$$

where the function  $com(c, \ell)$  returns the cost on the link  $\ell$  of communication  $c$ , and is defined by:

$$com(c, \ell) = \begin{cases} cc(c) & \text{iff } \ell \in path(d(x), d(y)), c = (x, y) \\ 0 & \text{otherwise} \end{cases}$$

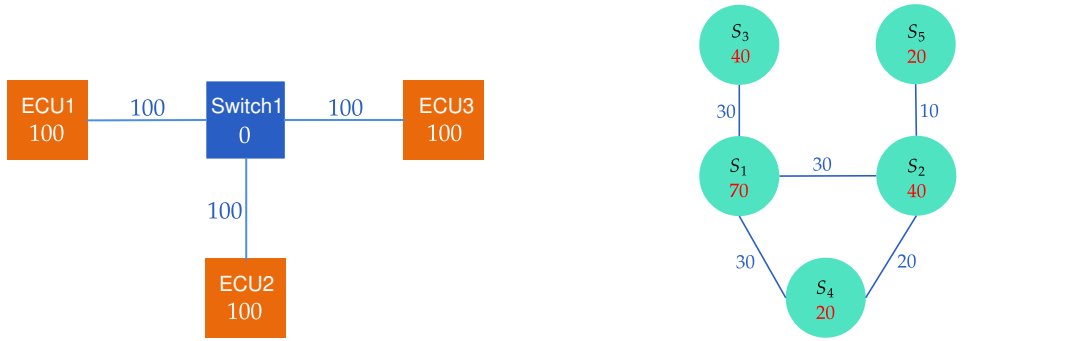
This constraint guarantees that the maximal capacity of a network link  $\ell$  is never exceeded by all communications  $c \in Com$  deployed on processors communicating through this link  $\ell$ . The function  $path(h_1, h_2)$  returns a path in the hardware graph between two hardware units  $h_1$  and  $h_2$ . In our implementation, this function represents the routing table and is given by the user as a parameter of the model. Shortest path is the standard routing strategy in automotive networks, that we use in our experiments. Follow-up work may consider the routing table as a decision variable of the model.

Finally, we note that additional constraints may be considered in similar deployment problems [15, 11]. For instance, a service might need to be allocated on a specific processor (locality constraint) or on the same processor than another service (co-location constraint). For brevity and because these constraints can be taken into account in standard ways, we choose to focus on the core problem presented above.

## 2.1 Multi-Objective Optimization

In automotive applications, we usually want to find a deployment function  $d$  optimizing various objectives such as reliability [21], extensibility and cost reduction [11, 17]. We focus on two new extensibility objectives and a well-known cost reduction objective. Typically, once the services are deployed on the processors, they cannot be moved to other processors. This poses challenges when updating the system with new services. Therefore, an important goal is that the deployment function favours extensibility, that is the ability to add further services over the lifetime of the vehicle. This requirement is captured by two extensibility objectives as follows:

$$\min \max_{h \in H} \sum_{s \in d^{-1}(h)} sc(s)$$



(a) Hardware graph with 3 ECUs and 1 switch. The values of  $hc$  are displayed in white and those of  $lc$  in blue.

(b) Software graph with 5 services and 5 communications. The values of  $sc$  are displayed in red and those of  $cc$  in blue.

■ **Figure 1** An example of the software deployment problem.

which minimizes the maximum utilization rate of a processor, and

$$\min \max_{\ell \in L} \left( \sum_{c \in Com} com(c, \ell) \right) / lc(\ell)$$

which minimizes the maximum utilization rate of a network link. The maximum value is considered as the corresponding hardware resource will be the bottleneck of the system. Indeed, we want to avoid a processor to be fully occupied in case a new service requires to be placed on this processor in a system update, e.g., due to the proximity of a sensor.

At the same time, we want to minimize the number of processors in order to reduce the costs:

$$\min |d(S)|$$

Of course, this objective directly conflicts with the first one. The tradeoffs between extensibility and cost of the architecture are exposed to the system expert through a Pareto front of solutions. The final decision will involve many factors that are external to the model, such as re-use of existing processors and safety concerns.

### 3 Constraint Programming with External Function

#### 3.1 Constraint Programming

A constraint satisfaction problem (CSP) is a tuple  $(X, D, C)$  where  $X$  is a set of variables,  $D = D_1 \times \dots \times D_n$  the sets of values taken by each variable  $x_i \in X$ , and  $C$  a set of relations over variables, called *constraints*. An assignment is a function  $asn(x_i) = v_i$  from variable  $x_i \in X$  to values where  $v_i \in D_i$ . Let  $asn$  be an assignment and  $c \in C$  a constraint defined on the variables  $x_1, \dots, x_n$ . Then the constraint  $c$  is *satisfied* when  $c(asn(x_1), \dots, asn(x_n))$  holds, or for short  $c(asn)$ . An assignment  $asn$  is a *solution* when each constraint is satisfied. We write  $ASN$  the set of all assignments. We call the *concrete domain* the powerset lattice  $D^b = \langle \mathcal{P}(ASN), \supseteq \rangle$  where the least element is the set of all assignments  $ASN$  and the greatest element is the empty set (no solution). The set of solutions of a CSP  $P = (X, D, C)$  is an element of  $D^b$  computed by the following function:

$$sol(P) := \{asn \in ASN \mid \forall c \in C, c(asn)\}$$

A multi-objective constraint optimization problem  $M = (X, D, C, \preceq)$  extends the previous definition with a partial order relation  $\preceq: \text{ASN}|_{\text{OBJ}} \times \text{ASN}|_{\text{OBJ}}$  where  $\text{ASN}|_{\text{OBJ}}$  is the restriction of the assignments<sup>4</sup> to a set of objective variables  $\text{OBJ} \subseteq X$ . In contrast to a single-objective optimization problem, there can be several solutions such that none is better than the others, which is why we need a partial order. Let  $a, b \in \text{ASN}|_{\text{OBJ}}$ . In the case of maximization over integer variables, we can define  $a \preceq b \Leftrightarrow \forall x \in \text{OBJ}, a(x) \leq b(x)$ , that is, all the objectives of  $b$  are greater or equal to the ones of  $a$ . We say that  $b$  dominates  $a$  when  $a \preceq b$ . We write  $a \succ b$  for  $a \neq b \wedge a \succeq b$  and we have  $a \succeq b \Leftrightarrow b \preceq a$  and  $a \succ b \Leftrightarrow b \prec a$ . Importantly, due to the partial order, the fact that  $a$  does not dominate  $b$  ( $a \not\preceq b$ ) does not imply that  $a$  is dominated by  $b$  ( $a \prec b$ ).

For the sake of clarity, we overload  $\preceq$  to work on arbitrary assignments. For any  $a, b \in \text{ASN}$ , we have  $a \preceq b \Leftrightarrow a|_{\text{OBJ}} \preceq b|_{\text{OBJ}}$ , and similarly for  $\prec, \succeq$  and  $\succ$ . In that case,  $\preceq$  is a preorder since two assignments with the same objective values might not be equal:  $\preceq$  lacks antisymmetry. This is not an issue since antisymmetry can be recovered by considering classes of equivalent assignments, but we do not need this construction here. The solution function for a multi-objective constraint optimization problem is then defined as:

$$\text{sol}(X, D, C, \preceq) := \{a \in \text{sol}(X, D, C) \mid \forall b \in \text{sol}(X, D, C), b \not\prec a\}$$

Multi-objective optimization in the context of constraint programming is described in greater length in, e.g., [13, 31, 14].

In the following, we will also need to merge and generate constraints from the Pareto front. The type of a Pareto front is a set of assignments  $\text{PF} := \mathcal{P}(\text{ASN})$ . We define the operator  $\sqcup: \text{PF} \times \text{PF} \rightarrow \text{PF}$  merging two Pareto fronts as  $A \sqcup B := \{c \in A \cup B \mid \forall d \in A \cup B, d \not\prec c\}$ . An equivalent definition of the solutions set is possible using  $\sqcup$ :

$$\text{sol}(X, D, C, \preceq) := \bigsqcup \{\{a\} \mid a \in \text{sol}(X, D, C)\}$$

where  $\bigsqcup \{s_1, \dots, s_n\} := s_1 \sqcup \dots \sqcup s_n$ .

Finally, we define the function  $\text{opt}: \text{ASN} \rightarrow C$  which returns a constraint ensuring that for all solutions  $a \in \text{sol}(X, D, C)$ , no solution in  $b \in \text{sol}(X, D, C \wedge \text{opt}(a))$  is dominated by  $a$ , i.e.  $a \not\preceq b$ . This function is defined by:

$$\text{opt}(a) := \bigvee_{x \in \text{OBJ}} x < a(x)$$

It generates a constraint requiring at least one of the objective variables to be strictly better than the one obtained in  $a$ . This approach to multi-objective optimization was pioneered by Gavanelli [13].

## 3.2 Abstract Constraint Programming

In general, the set  $\text{sol}(P)$  might not be efficiently computed on the concrete domain. In constraint reasoning by abstract interpretation [10, 27, 33, 35], they design an abstract solving function  $\text{sol}_o^\sharp(P)$  which *over-approximates* the solution set, i.e.,  $\text{sol}_o^\sharp(P) \supseteq \text{sol}(P)$ . Dually, we can also design an *under-approximating* solving function such that  $\text{sol}_u^\sharp(P) \subseteq \text{sol}(P)$ . Over-approximation contains all solutions but might contains non-solution assignments as well, while under-approximation only contains solutions but not necessarily all solutions. For

<sup>4</sup> Formally,  $\text{ASN}|_{\text{OBJ}} := \{asn|_{\text{OBJ}} \mid asn \in \text{ASN}\}$  where  $asn|_{\text{OBJ}}(x) = asn(x)$  for all  $x \in \text{OBJ}$ .



instance, discrete constraint programming solvers are both under- and over-approximating [10, 35], and continuous constraint programming solvers are over-approximating [27]. Incomplete discrete solvers, such as those based on local search, can be viewed as under-approximating solving functions.

### 3.3 Abstract Constraint Model

We can also use the abstraction framework at the level of the constraint model. In industry, some elements of a constraint model might already be available and tested, and it is usually not practical to spend time redeveloping those parts as a constraint problem. Sometimes, the problem is just too difficult to be expressed as a constraint model in a reasonable amount of time; this is the case of the WCTT analysis for instance. In these cases, the problem  $P$  is never explicitly written as a constraint model. Instead, we can rely on an over-approximating model  $O$  of  $P$ , such that  $\text{sol}(O) \supseteq \text{sol}(P)$ . We often have an idea of some constraints that must be satisfied in any solution of the model but we do not necessarily know them all. This model  $O$  can be solved by an over-approximating function  $\text{sol}_o^\sharp(O) \supseteq \text{sol}(O)$  – although  $O$  simplifies  $P$ , it might still not be efficiently computable. If  $O$  is unsatisfiable ( $\text{sol}_o^\sharp(O) = \{\}$ ), then the problem  $P$  is unsatisfiable as well since only  $\text{sol}(P) = \{\}$  satisfies  $\text{sol}_o^\sharp(O) \supseteq \text{sol}(P)$ . In the following, we denote  $\text{OSOLVE}(O) \in \text{sol}_o^\sharp(O)$ , the solving algorithm computing a single solution of  $O$  and returning  $\{\}$  if  $O$  is unsatisfiable. Its definition in terms of abstract interpretation can be found in [1, 10, 27]. Dually, we can also propose an under-approximating model  $U$  of  $P$  and its solving function  $\text{sol}_u^\sharp(U)$  such that  $\text{sol}_u^\sharp(U) \subseteq \text{sol}(U) \subseteq \text{sol}(P)$ . If  $U$  is satisfiable, then the problem  $P$  is satisfiable as well. An under-approximation makes additional assumptions about the reality, and therefore might discard solutions of  $P$ . Therefore, the real problem  $P$  is framed between an over-approximating model  $O$  and an under-approximating model  $U$ , which is summarized by  $\text{sol}_o^\sharp(O) \supseteq \text{sol}(O) \supseteq \text{sol}(P) \supseteq \text{sol}(U) \supseteq \text{sol}_u^\sharp(U)$ .

### 3.4 Under-Approximating External Function

We must go one step further for our abstract framework to be useful in practice. If we cannot explicitly list the constraints of the concrete problem  $P$ , it seems unlikely that we could list *more constraints* in an under-approximating model  $U$ . Nevertheless, when given a solution to  $O$ , it can often be validated by existing code developed by domain experts. For instance, worst-case analysis such as WCTT and feasibility tests fall in this category. They conservatively analyse the network architecture, and discard some solutions that would be valid but could not be proven valid by the analysis. In practice, worst-case analysis are not directly working with constraints and domains, and thus do not explicitly define an under-approximating constraint model  $U$ , but they work on assignments.

We formally define the analysis as an under-approximating function  $uf : \text{ASN} \rightarrow C$ . The function  $uf$  returns a conflict constraint when the assignment generated by  $\text{OSOLVE}$  is not in  $\text{sol}(U)$ , or *true* if it is in  $\text{sol}(U)$ . Actually, we define the solutions of the under-approximating model  $U$  as the set of all solutions accepted by  $uf$ :

$$\text{sol}(U) := uf^{-1}(\text{true}) = \{asn \in \text{ASN} \mid uf(asn) = \text{true}\}$$

A general conflict automatically available to all functions  $uf$ , is the logical negation of the assignment (**NA**):  $\neg asn := \neg(x_1 = asn(x_1) \wedge \dots \wedge x_n = asn(x_n)) \Leftrightarrow x_1 \neq asn(x_1) \vee \dots \vee x_n \neq asn(x_n)$ . However, it is a weak conflict since it only prevents  $\text{OSOLVE}$  from returning to this assignment, without providing additional pruning. A conflict is *over-approximating* if it does not remove valid solutions: for all assignments  $asn$ , we have  $\text{sol}(U) \subseteq \text{sol}(O \wedge uf(asn))$ .

<pre> <b>function</b> USOLVE(<math>O, ufo</math>)   <math>S \leftarrow \{\}</math>   <math>asn \leftarrow OSOLVE(O)</math>   <b>while</b> <math>asn \neq \{\}</math> <b>do</b>     <b>if</b> <math>ufo(asn) = \text{true}</math> <b>then</b>       <math>S \leftarrow S \cup \{asn\}</math>       <math>O \leftarrow O \wedge \neg asn</math>     <b>else</b>       <math>O \leftarrow O \wedge ufo(asn)</math>     <b>end if</b>     <math>asn \leftarrow OSOLVE(O)</math>   <b>end while</b>   <b>return</b> <math>S</math> <b>end function</b> </pre>	<pre> <b>function</b> USOLVE_MO(<math>O, ufo, \sqcup, opt</math>)   <math>F \leftarrow \{\}</math>   <math>asn \leftarrow OSOLVE(O)</math>   <b>while</b> <math>asn \neq \{\}</math> <b>do</b>     <b>if</b> <math>ufo(asn) = \text{true}</math> <b>then</b>       <math>F \leftarrow F \sqcup \{asn\}</math>       <math>O \leftarrow O \wedge opt(asn)</math>     <b>else</b>       <math>O \leftarrow O \wedge ufo(asn)</math>     <b>end if</b>     <math>asn \leftarrow OSOLVE(O)</math>   <b>end while</b>   <b>return</b> <math>F</math> <b>end function</b> </pre>
--	--

(a) Find all satisfiable solutions.

(b) Multi-objective version of USOLVE.

■ **Figure 2** Constraint solving with external under-approximating function producing over-approximating conflicts.

We say that a conflict  $c$  is *sound* if it implies **NA**; in other terms, it excludes the current assignment:  $asn \notin sol(c)$ . **NA** is a sound over-approximating conflict. We denote by  $ufo$  an under-approximating external function returning sound over-approximating conflicts.

Let  $O$  be an over-approximating model and  $ufo$  a sound under-approximating external function. The function USOLVE presented in Algorithm 2a constructs the solutions set  $S$  of the under-approximating model  $U$ . Constraint programming helps us navigating in the under-approximated solution space of the external function more efficiently. Without it, we would need to call  $ufo$  on many more unsatisfiable assignments, since those would not be removed by a constraint solver. The next proposition shows that USOLVE computes an under-approximation of  $P$ .

► **Proposition 1.** *USOLVE is a sound under-approximating function, that is,  $USOLVE(O, ufo) = ufo^{-1}(true) \subseteq sol(P)$ .*

**Proof.** Let  $O_i$  and  $S_i$  be the variables  $O$  and  $S$  at the  $i$ th iteration of the loop where  $O_0 = O$  and  $S_0 = \{\}$ . We must show that at the final iteration  $n$ , we have  $ufo^{-1}(true) = S_n$ . We proceed inductively by defining  $O_{i+1}$  and  $S_{i+1}$  as follows:

1. If  $asn$  is a solution to  $ufo$ :  $O_{i+1} = O_i \wedge \neg asn$  and  $S_{i+1} = S \cup \{asn\}$ . In that case we have  $sol(O_i) \cup S_i = sol(O_{i+1}) \cup S_{i+1}$ .
2. If  $asn$  is not a solution to  $ufo$ :  $O_{i+1} = O_i \wedge ufo(asn)$  and  $S_{i+1} = S$ . In that case we have  $sol(O_i) \cup S_i \supseteq sol(O_{i+1}) \cup S_{i+1}$ . Since the conflict must be over-approximating, no assignment removed from  $O_i$  is in  $ufo^{-1}(true)$  and therefore  $ufo^{-1}(true) \subseteq sol(O_{i+1}) \cup S_{i+1}$ .

The final iteration is necessarily with  $O_n = \{\}$ , hence we must have  $ufo^{-1}(true) \subseteq S_n$ . Since we only add in  $S_n$  the assignments  $asn$  such that  $ufo(asn) = true$ , we also have  $ufo^{-1}(true) \supseteq S_n$ . ◀

The extension to multi-objective optimization is a small modification of USOLVE. The set  $F$  represents the Pareto front of the problem. To compute the Pareto front, we introduce the algorithm USOLVE\_MO in Figure 2b, and we highlight the differences with USOLVE in green.

► **Proposition 2.** *USOLVE\_MO is a sound under-approximating function. Moreover, we have:*

1.  $USOLVE\_MO(O, ufo, \sqcup, opt) = sol(U, \preceq)$ ,
2.  $USOLVE\_MO(O, ufo, \sqcup, opt) \subseteq USOLVE(O, ufo)$ , and
3.  $USOLVE\_MO(O, ufo, \sqcup, opt) = \{\} \Leftrightarrow USOLVE(O, ufo) = \{\}$ .

**Proof.** We prove each statement in turn:

1. Each time we reach a solution  $asn$ , we add the constraint  $opt(asn)$  to  $O$ . By definition, all solutions removed by this constraint are dominated by  $asn$ , and therefore it cannot remove solutions from  $sol(U, \preceq)$ .
2. Notice that  $opt(asn) \Rightarrow \neg asn$ , and therefore it can only prune more solutions in comparison to  $USOLVE$ .
3. Before reaching the first solution, the algorithm behaves in the same way than  $USOLVE$ . ◀

The risk when navigating in an under-approximating solution space is to find no solution at all. Therefore, it is useful to notice that the multi-objective algorithm returns an empty set of solution only if  $USOLVE$  does as well (by Proposition 2(3)).

## 4 Worst-Case Traversal Time Analysis

We focus on the worst-case traversal time (WCTT) analysis, which verifies if the network communications among the deployed services meet timing constraints, i.e., deadline constraints in this work. In an automotive network, we must ensure the deadlines of network packets are met, which is crucial for safety reasons (e.g., a message sent to an airbag arrives on time) and other non-functional requirements (e.g., the speakers must be synchronized when playing music). WCTT analysis is a formal method which provides upper bounds on the worst-case delay of every packet sent in the network. It is therefore an under-approximating external function because all deployments passing this analysis will also fulfill the real-time constraints in reality. But some deployments, that in fact meets all timing constraints, will not pass the WCTT analysis because it only gives an upper-bound on the delay: it is a sufficient but not necessary condition.

The WCTT analysis for Ethernet networks, based on *network calculus* [18], is mathematically complicated (see for instance [28]). We think it would take tremendous efforts to model WCTT analysis as a constraint problem if the goal is to develop an implementation that is sufficiently accurate to be used on real-world problems, given the complexity of the WCTT analysis after 30 years of research. As an illustration, the network calculus engine from the company RTAW we use in the paper has been developed for 15 years and implements state-of-the-art techniques such as [34, 3]. We take a more pragmatic approach where we reuse an existing WCTT analyser, and integrate it in our framework as an under-approximating external function.

Let us first describe the output of a WCTT analysis. For each communication  $(x, y) \in Com$ , the WCTT analysis outputs the worst-case end-to-end delay of a packet traversing the network from  $d(x)$  to  $d(y)$ . A negative delay means the deadline for that communication cannot be met and thus the deployment  $d$  is unsatisfiable from the point of view of the analysis. From an unsatisfiable assignment, we can think of various conflicts such as forcing the network load of the problematic link to be smaller, or forbidding to allocate the services  $x$  or  $y$  on their current processors. Unfortunately, conflicts that are intuitive are often not over-approximating. To complicate the finding of over-approximating conflicts, the WCTT analysis is non-monotonic w.r.t. the network load. Indeed, it can happen that increasing the load of an unsatisfiable network turns it into a satisfiable network according to the WCTT analysis. This is a well-known phenomenon referred to as *timing anomaly* (see, e.g., [23]), that may occur with non-preemptive scheduling as in communication networks.

```

function CUSOLVE_MO( $F, O, C, uf, \sqcup, opt$ )
   $asn \leftarrow OSOLVE(O \wedge C)$ 
  if  $asn \neq \{\}$  then
     $co \leftarrow uf(asn)$ 
    if  $co = \text{true}$  then
       $F \leftarrow F \sqcup \{asn\}$ 
       $O \leftarrow O \wedge opt(asn)$ 
      CUSOLVE_MO( $F, O, C, uf, \sqcup, opt$ )
    else
      CUSOLVE_MO( $F, O, C \wedge co, uf, \sqcup, opt$ )
      CUSOLVE_MO( $F, O, C \wedge \neg co \wedge \neg asn, uf, \sqcup, opt$ )
    end if
  end if
  return  $F$ 
end function

```

■ **Figure 3** Multi-objective constraint solving with under-approximating external function returning conflicts that are not over-approximating.

Although the conflicts mentioned above are not over-approximating, they can nevertheless be useful as heuristics to find a solution faster. In Figure 3, we extend USOLVE\_MO in the case where  $uf$  is returning sound conflicts that are not over-approximating. The intuition is that the conflict is viewed as a branching decision, and thus backtracked when the sub-problem has been fully explored. Doing so, we do not lose the completeness of our solving algorithm. We provide an example unrolling this algorithm in Section 4.2.

We note that if the conflict  $co$  is over-approximating, then  $O \wedge \neg co$  is necessarily unsatisfiable. In that case, CUSOLVE\_MO remains correct but the second recursive call CUSOLVE\_MO( $F, O, C \wedge \neg co, uf, \sqcup, opt$ ) is unnecessary.

► **Proposition 3.**  $CUSOLVE\_MO(\{\}, O, \{\}, uf, \sqcup, opt) = sol(U, \preceq)$

**Proof.** The difference with Proposition 2, is that  $uf$  does not necessarily produce over-approximating conflicts. However, given any constraint problem  $O$  and any constraint  $co$ , we always have  $sol(O \wedge co) \cup sol(O \wedge \neg co) = sol(O)$ . Therefore, the same solution space is eventually explored, and the same Pareto front is found. However, the conflicting assignment  $asn$  might be reexplored in the right branch. Indeed,  $sol(\neg co) = ASN \setminus sol(co)$  and therefore  $asn \in sol(\neg co)$ . By forbidding revisiting this assignment in both the left and right branches, we guarantee progress and thus termination of the algorithm. ◀

## 4.1 Conflicts for WCTT

Let  $asn$  be the current assignment for which WCTT detected that the communication  $(x, y) \in Com$  does not meet its deadline. Among the possible conflicts, we experiment with the following ones in the next section:

- Forbid the source service  $x$  to be allocated on its current hardware unit or the one of  $y$ :

$$d(x) \notin \{asn(x), asn(y)\} \quad (\mathbf{FS})$$

- Forbid the target service  $y$  to be allocated on its current hardware unit or the one of  $x$ :

$$d(y) \notin \{asn(x), asn(y)\} \quad (\mathbf{FT})$$

- Decrease the number of hops in the network between  $x$  and  $y$ :

$$|\text{path}(d(x), d(y))| < |\text{path}(\text{asn}(x), \text{asn}(y))| \quad (\text{DH})$$

We also test two conflicts that are global to the network, thus do not consider a specific communication. It is based on the observation that a communication can fail to meet its deadline because of *other* communications. We let the variable  $\text{load}_\ell = \sum_{c \in \text{Com}} \text{com}(c, \ell)$  to be the network load of the link  $\ell$  in the current assignment  $\text{asn}$ .

- Decrease the load of at least one network link:

$$\bigvee_{\ell \in L} \sum_{c \in \text{Com}} \text{com}(c, \ell) \leq \text{load}_\ell \quad (\text{D1L})$$

- Decrease the load of the most occupied network link, with  $m = \text{maxarg}_{\ell \in L}(\text{load}_\ell / \text{lc}(\ell))$ :

$$\sum_{c \in \text{Com}} \text{com}(c, m) \leq \text{load}_m \quad (\text{DML})$$

Taking the conjunction of any two conflicts will further prune the search tree, while taking their disjunction will create a weaker conflict. In the experiments, we test the conjunction of **FS** and **FT** that we name **FST**. In our investigations, we found that, in general, the disjunction of conflicts did not help to reach a (better) solution faster.

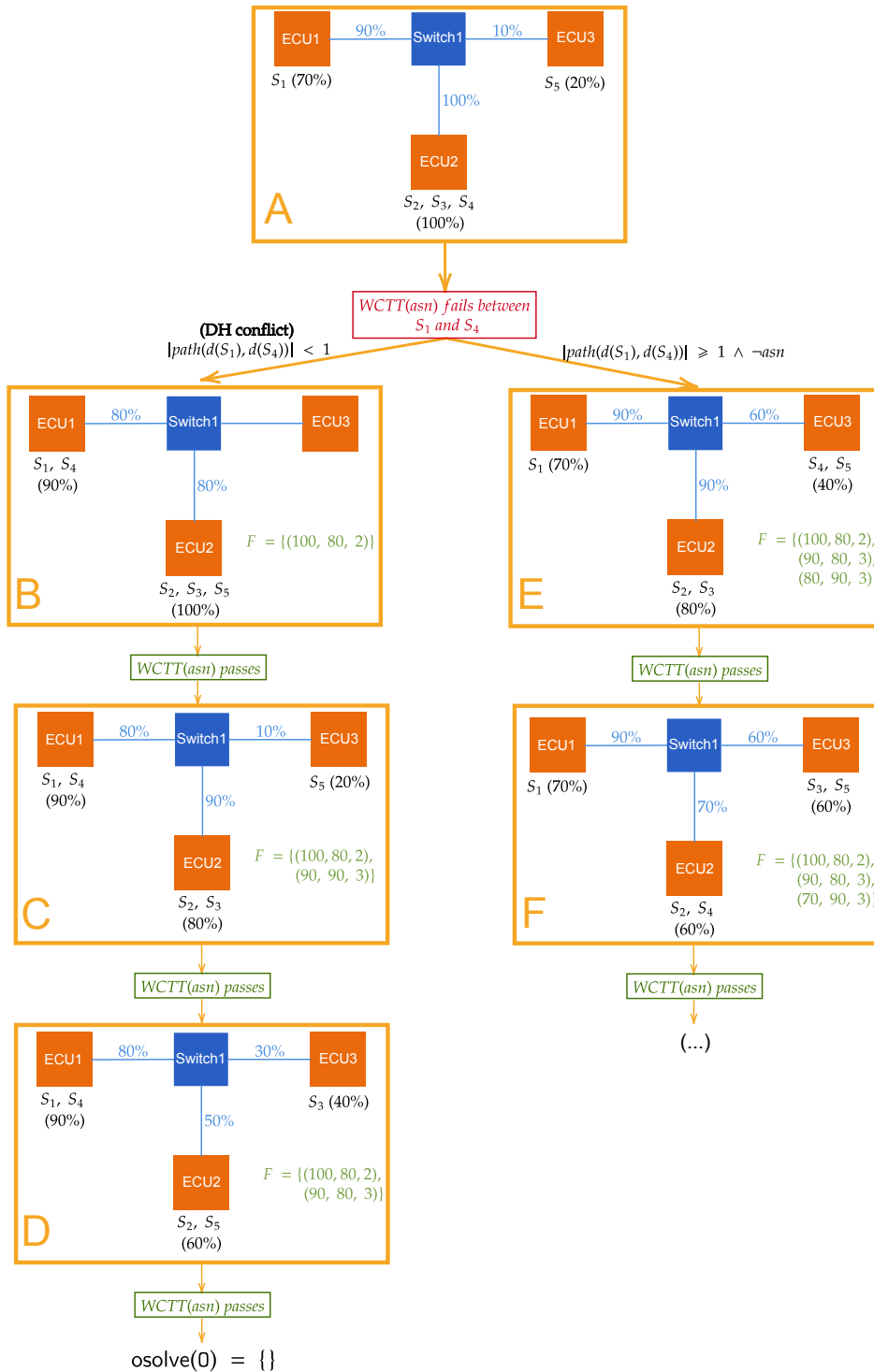
## 4.2 An Example of the Algorithm `cusolve_mo`

We unroll the algorithm `CUSOLVE_MO` with the **DH** conflict strategy on the software deployment problem given in Figure 1. Initially, the Pareto front  $F$  and the set of conflicts  $C$  are empty. The first call to `OSOLVE` returns a solution to the problem as depicted in the orange box labelled A. Because one of the network link is used at 100% capacity, we suppose the communication between  $S_1$  and  $S_4$  fails to meet its deadline, hence the WCTT analysis fails on this solution. The conflict  $|\text{path}(d(S_1), d(S_4))| < 1$ , reducing the number of hops between  $S_1$  and  $S_4$ , is added to the conflicts set  $C$ . In this case, this conflict forces both services to be allocated on the same processor.

The model is solved again with this new conflict and `OSOLVE` returns a solution as depicted in the box B. This time the WCTT analysis succeeds, and the solution is added to the Pareto front  $F$ . The objectives are  $(100, 80, 2)$  where 100 is the maximum utilization rate among all processors, 80 is the maximum utilization among all network links and 2 is the number of cores used.

As long as the WCTT analysis succeeds, the `OSOLVE` procedure is iteratively called with the updated Pareto front. In the box C, we have a solution  $(90, 90, 3)$  incomparable to  $(100, 80, 2)$ , hence the Pareto front now contains both. In the box D, we find the solution  $(90, 80, 3)$  which dominates the previous one. Afterwards, the `OSOLVE` procedure finds the problem unsatisfiable, which means there is no solution better than the ones found previously. However, we have previously added a conflict which was not over-approximating, and therefore we might have missed solutions of the problem. Therefore, we need to backtrack and explore the problem with the negation of the conflict. In box E, the model is solved again with the latest Pareto front, and a new non-dominated solution  $(80, 90, 3)$  is found. This is repeated and a better solution  $(70, 90, 3)$  is found in box F. As long as the WCTT analysis succeeds, the solving procedure continues, and when it fails we branch as we did in the first node.

This example demonstrates that conflicts are heuristics which are used in a way that do not prevent to find all solutions. This is also why the non-monotonicity of the WCTT analysis is not an issue: the entire solution space is eventually explored.



■ **Figure 4** An example unrolling CUSOLVE\_MO on a small network. The orange boxes represent the solutions found by OSOLVE.  $F$  is the current Pareto front and is automatically added to the model before solving; note that the Pareto front is preserved on backtracking.

## 5 Implementation and Experiments

The code of the MINIZINC model, the data of the instances and the implementation of the algorithms used can be found online at <https://github.com/ptal/automotive-network-cp/tree/cp2023>.

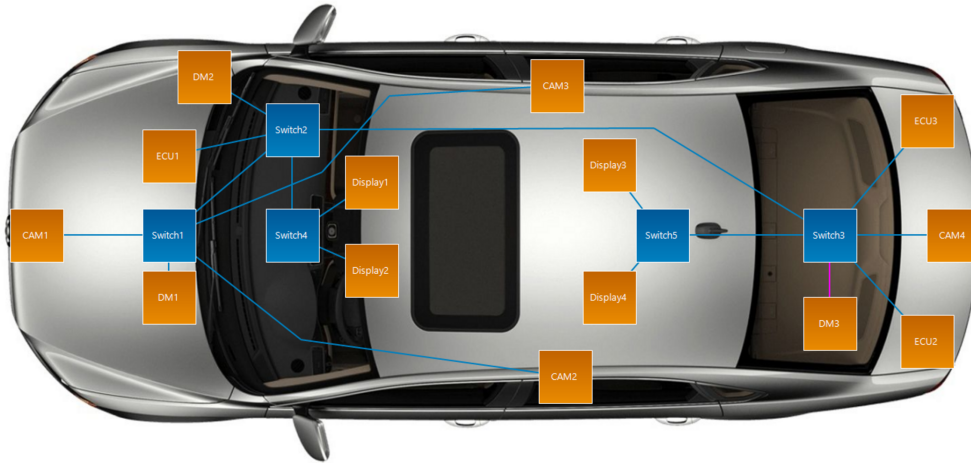
### 5.1 Experimental Setting

We run all the experiments on an AMD Epyc ROME 7H12 processor (64 cores, 280W). The constraint programming solver implementing the OSOLVE solving function is GECODE 6.3.0 [32] in parallel mode with 8 cores and 16 threads. We use a first-fail variable selection strategy (the variable with the smallest domain is chosen first) and a random value selection, as it shows better performance than the free search strategy of GECODE. We also tried CHUFFED 0.10.4 [26], a hybrid solver between SAT and constraint programming, but it did not outperform Gecode on our problem. Due to the lack of open-source alternative, the WCTT analysis is performed by the proprietary software RTAW-PEGASE-4.3.7 [29], which implements state-of-the-art network calculus algorithms [18, 2, 4]. The WCTT analysis takes on average 1.5 seconds to run, and this time remains stable across instances. The algorithms presented in this paper – USOLVE, USOLVE\_MO and CUSOLVE\_MO – are implemented in Python using MINIZINC PYTHON 0.9.0 [9]. In addition we provide OSOLVE\_MO which implements the multi-objective optimization solving procedure of [13] – it is the same than USOLVE\_MO but without the external function filtering. Although multi-objective optimization is very important in practice, it is not natively available in every constraint programming solver (for instance in GECODE, CHUFFED or ORTOOLS). Similarly to [14], our approach does not require to modify the constraint solver, but the solver state is lost between two calls to OSOLVE which might be less efficient – although the issue is mitigated since we use a random search strategy. It can be seen as a restart strategy triggered on every solution. It is also similar to what is done in MINISEARCH [30] to design search strategies generically across solvers.

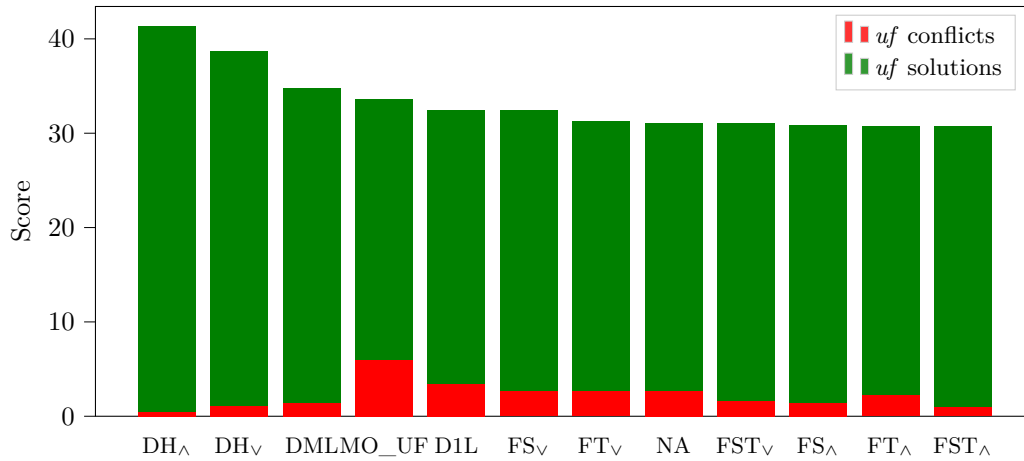
### 5.2 Dataset Description

The instances are derived from a realistic automotive Ethernet network, shown in Figure 5, consisting of 19 network devices (14 ECUs and 5 switches) provided by the company RealTime-at-Work<sup>5</sup>. The experiments consider 5 problem instances of 50 services, 5 instances of 75 services and 8 instances of 100 services. The numbers of communications vary among the instances, but are between 125% and 135% of the number of services. An information missing in the network description is the CPU usage for each service. For each of the 18 instances, we generated 10 versions where the sum of all computational requirements is 20%, 40%, 60%, 80% and 90% of the total computational capacity of all ECUs with a uniform distribution among services. To summarize, an instance named I5\_75-14-u60 has 75 services allocated on 14 processors and using 60% of the total computational power, and I5 denotes the fifth instance with 75 services. In total, we have a new dataset of 90 MiniZinc instances for the deployment problem. In the following, we present experimental results for a subset of these instances (I{1,2,5}\_{50,75,100}-14-u{20,40,60,80,90}), totalizing 45 instances. We set a timeout on the constraint solver of 30 minutes for each instance, and unlimited time for the WCTT analysis.

<sup>5</sup> <https://www.realtimeatwork.com/>



■ **Figure 5** Realistic automotive Ethernet network used in the experiments.



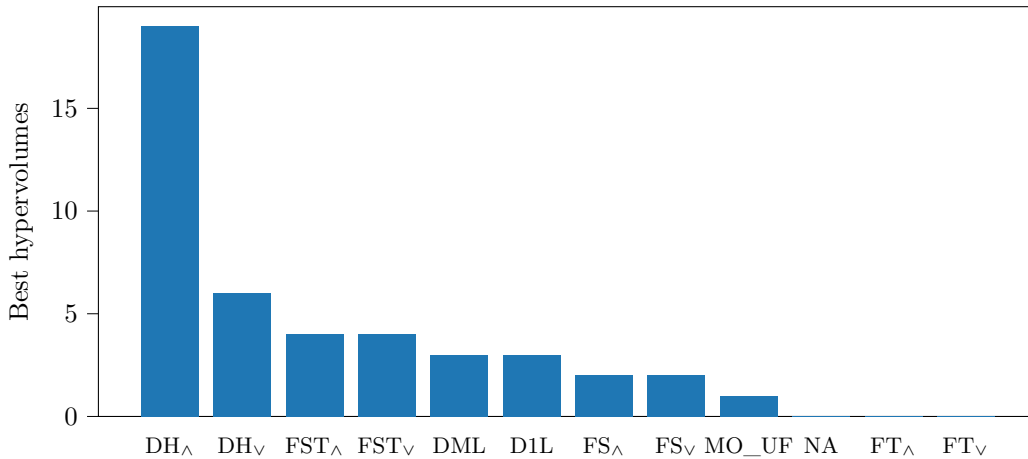
■ **Figure 6** Cumulated hypervolume score for each experiment over all instances.

### 5.3 Evaluation of `cusolve_mo`

We evaluate the algorithm `CUSOLVE_MO` on **NA** and the seven strategies presented in Section 4.1. For a single assignment, it is possible that several communications cannot meet their deadlines, and thus several conflicts are generated. We write **FS<sub>∨</sub>** when these conflicts are combined disjunctively and **FS<sub>∧</sub>** when they are combined conjunctively. It only impacts the conflicts that are local to a communication (**FS**, **FT**, **DH** and **FST**), thus we have 11 conflicts in total.

Our main comparison metrics is the hypervolume of the Pareto front which is standard in multi-objective optimization. For all 45 instances, none of the algorithms tested could find the optimum within the time limit. We give a general picture of the situation in Figures 6 and 7. Overall, the decreasing hops strategy is the best, and finds the best hypervolumes on 19 of the 45 instances. We also witness a smaller number of conflicts, which means that **DH** is effective to search the state-space of  $sol(U)$ . When considering the score, forbidding the





■ **Figure 7** Number of times each experiment computed the best hypervolume.

source and target services on a particular ECU are usually not better than simply using the **NA** conflict. These strategies are still superior regarding the number of times they find the best hypervolume.

In addition, we evaluate `CUSOLVE_MO` against a more straightforward *two-steps* algorithm `OSOLVE_MO_THEN_UF` (denoted by **MO\_UF**) where the Pareto front is first fully generated, and then filtered by the  $uf$  function. To improve this method, we keep all intermediate solutions when building the Pareto front in a set  $S$ . During the filtering step, if a solution  $a$  is discarded by the function  $uf$ , we remove  $a$  from the Pareto front and reconstruct it with  $\sqcup\{\{b\} \mid b \in S \setminus \{a\}\}$ . It does not make this algorithm over-approximating as it can still discard assignments accepted by  $uf$ , but it improves the filtered Pareto front.

As shown in Figure 6, **MO\_UF** ranks fourth, and therefore is a good approach to solve the deployment problem when we do not seek (or cannot find) the true optimal solution. Interestingly, for 18 instances over the 45, the hypervolume before and after filtering is the same, which means that all solutions of the Pareto front were valid w.r.t.  $uf$ . This is particularly true with 50 services where 14/15 instances have the same hypervolume before and after filtering. This result is explained by noticing that adding more services has a higher impact on the network load, and thus the WCTT analysis fails more often. Over the 30 instances with 75 and 100 services, there are 24 instances that have a filtered hypervolume within 3% of the unfiltered hypervolume. It is not always the case as for the instances `I1_100-14-u60` and `I1_100-14-u80`, the filtered hypervolume is respectively 57% and 73% of the unfiltered hypervolume. An advantage of the offline filtering proposed by **MO\_UF** is to call  $uf$  an order of magnitude less than with **DH**. Over all instances, **MO\_UF** calls  $uf$  1180 times while **DH** calls  $uf$  12245 times. Therefore, depending on the time taken by the external function and the number of conflicts, **MO\_UF** can be better – especially for low number of conflicts and long evaluation time.

From an implementation perspective, using `MINIZINC PYTHON` allowed us to implement an algorithm generic across solvers, but it incurs a cost. Besides losing the state of the solver between calls, we must call the `MINIZINC` (source of the model) to `FLATZINC` (simpler format supported by solvers) translator, and it takes on average around 40% of the total solving time. An improvement to `MINIZINC PYTHON` would be to directly add `FLATZINC` constraints to avoid recompiling the `MINIZINC` model each time.

## 6 Related Work

### 6.1 CAN Networks

The deployment problem has been extensively studied due to its importance in distributed real-time embedded systems. It was pioneered in [38] for tasks allocation on *controller area network* (CAN). In CAN network, the hardware units are all connected on a broadcast bus, and therefore the hardware network is fully connected. The main difference with our work is that we consider a more general switch-based network. In the context of real-time and critical systems, such as those found in the automotive industry, it is crucial to ensure the network will not be overloaded by communication, and when required, that the network packet deadlines are met. The WCTT analysis on switch-based networks is more complicated and under-approximating (it does not give an exact upper bound), while it is an exact analysis for CAN network [37, 7, 39].

Due to its simpler nature, schedulability analysis, such as WCTT, over CAN networks has been directly incorporated in the constraint model before. The work of Hladik et al. [15] is the first to model the deployment problem over CAN network using constraint programming. They model the schedulability analysis as a global constraint. Alternatively, they also use a method inspired by logic-based Benders decomposition (LBBD) [16] to separate the allocation problem solved using constraint programming and the schedulability analysis solved by an ad-hoc algorithm. It differs from our approach mainly because there is no notion of approximation, and the conjunction of both parts models the problem exactly. Moreover, they consider only the satisfiability of the problem, and they do not seek to optimize one or more objectives.

Other techniques were proposed to solve the deployment over CAN networks with multi-objective optimization, for instance, evolutionary optimization [21], ant colony system with constraint propagation and without searching [36], mixed integer linear programming (MIP) [24] and satisfiability modulo theories (SMT) [11]. These methods are either incomplete (no proof of optimality or unsatisfiability) and thus under-approximating, or they are complete (MIP and SMT) which is only possible because they model a simpler problem (CAN network).

### 6.2 Switched Networks

To the best of our knowledge, Kugele et al. [17] are the first to configure and analyse an application distributed over a switched network using a SMT solver. Similarly to OSOLVE\_MO, they generate a Pareto front and then verify the produced solutions. However, the verification is performed using simulation, which does not give a formal worst-case guarantee. Therefore, their solving method is over-approximating and the obtained solutions are not guaranteed to be valid. Besides, they do not provide the constraint model and only tested their algorithm on a small network of 3 ECUs and 25 services.

### 6.3 Other Applications

Campeanu et al. [5] study the deployment problem in heterogeneous architectures (CPU, GPU and FPGA), but with communication still happening over a CAN network. *Satisfiability Modulo Discrete Event Simulation* [20] combines a SAT solver with discrete event simulation (DES) for a railway construction planning problem. The combination of both techniques share similarities with CUSOLVE\_MO since the DES simulator is encapsulated as a theory and provide conflict to the SAT solver – but, like us, only on full assignments. However, simulation is an over-approximating technique and therefore the global method remains over-approximating. Moreover, the algorithms are specialized to the railway construction problem and no general algorithm or correctness proof is given.

## 6.4 Online and Dynamic Constraint Programming

Our work is related to online constraint programming [12] – also called dynamic constraint programming [8] – as in both approaches the model is incrementally refined. A difference is that online constraint programming is primarily designed when the solutions generated are used in real-time, and variables impacting the past decisions cannot be modified in the subsequent solving steps. We do not have such real-time requirements since the new data are obtained from an offline analysis. Moreover, in [12], they propose to internalize the dynamic part of the problem inside the model. Here, we purposely delegated a part of the model to an external function, which would have been prohibitively complicated to model otherwise.

## 7 Conclusion

We study the deployment problem, an important problem in the field of distributed real-time embedded systems, and more specifically in the automotive industry. This problem has a task allocation part, which is efficiently solved by constraint programming, and a scheduling network analysis part, which is efficiently solved by a WCTT analysis. The integration of both techniques is difficult since both parts are black-box functions. We propose the algorithm CUSOLVE\_MO which combines both parts in a loosely coupled manner, thus making our framework reusable on other similar problems. Our approach is based on abstract interpretation, a formal method allowing us to prove properties of our algorithms. Finally, we evaluated our approach on a new dataset for the deployment problem – since none existed before – and conclude that the cooperation scheme proposed by CUSOLVE\_MO works better than solving each part in sequence.

---

## References

- 1 Krzysztof R. Apt. The essence of constraint propagation. *Theoretical computer science*, 221(1-2):179–210, 1999. doi:10.1016/S0304-3975(99)00032-8.
- 2 Anne Bouillard and Éric Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, 2008. doi:10.1007/s10626-007-0028-x.
- 3 Marc Boyer and Hugo Daigmonte. Improved service curve for element with known transmission rate. *IEEE Networking Letters*, 5(1):46–49, 2023. doi:10.1109/LNET.2022.3150649.
- 4 Marc Boyer, Jörn Migge, and Nicolas Navet. An efficient and simple class of functions to model arrival curve of packetised flows. In *Proceedings of the 1st International Workshop on Worst-Case Traversal Time, WCTT '11*, pages 43–50, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/2071589.2071595.
- 5 Gabriel Campeanu, Jan Carlson, and Severine Sentilles. Component Allocation Optimization for Heterogeneous CPU-GPU Embedded Systems. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 229–236, Verona, Italy, 2014. IEEE. doi:10.1109/SEAA.2014.29.
- 6 Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL '77*, pages 238–252, New York, NY, USA, 1977. Association for Computing Machinery. doi:10.1145/512950.512973.
- 7 Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Systems*, 35(3):239–272, 2007. doi:10.1007/s11241-007-9012-7.

- 8 Rina Dechter and Avi Dechter. Belief Maintenance in Dynamic Constraint Networks. In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*, AAAI'88, pages 37–42. AAAI Press, 1988.
- 9 Jip J. Dekker. MiniZinc Python, 2023. URL: <https://github.com/MiniZinc/minizinc-python>.
- 10 Vijay D'Silva, Leopold Haller, and Daniel Kroening. Abstract satisfaction. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '14*, pages 139–150, San Diego, California, USA, 2014. ACM Press. doi:10.1145/2535838.2535868.
- 11 Johannes Eder, Sebastian Voss, Andreas Bayha, Alexandru Ipatiov, and Maged Khalil. Hardware architecture exploration: automatic exploration of distributed automotive hardware architectures. *Software and Systems Modeling*, 19(4):911–934, 2020. doi:10.1007/s10270-020-00786-6.
- 12 Alexander Ek, Maria Garcia de la Banda, Andreas Schutt, Peter J. Stuckey, and Guido Tack. Modelling and Solving Online Optimisation Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1477–1485, 2020. doi:10.1609/aaai.v34i02.5506.
- 13 Marco Gavanelli. An algorithm for multi-criteria optimization in CSPs. In *ECAI 2002: 15th European Conference on Artificial Intelligence, July 21-26, 2002, Lyon France: Including Prestigious Applications of Intelligent Systems (PAIS 2002): Proceedings*, volume 77, page 136. IOS Press, 2002.
- 14 Tias Guns, Peter J. Stuckey, and Guido Tack. Solution Dominance over Constraint Satisfaction Problems, 2018. arXiv:1812.09207 [cs]. URL: <http://arxiv.org/abs/1812.09207>.
- 15 Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche, and Narendra Jussien. Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software*, 81(1):132–149, 2008. doi:10.1016/j.jss.2007.02.032.
- 16 J.N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003. doi:10.1007/s10107-003-0375-9.
- 17 Stefan Kugele, Philipp Oberfell, and Eric Sax. Model-based resource analysis and synthesis of service-oriented automotive software architectures. *Software and Systems Modeling*, 20(6):1945–1975, December 2021. doi:10.1007/s10270-021-00896-9.
- 18 Jean-Yves Le Boudec and Patrick Thiran. Network Calculus. In *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, pages 3–81. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. doi:10.1007/3-540-45318-0\_1.
- 19 Martin Lukasiewicz, Michael Glaß, Christian Haubelt, and Jürgen Teich. Solving Multi-objective Pseudo-Boolean Problems. In *Theory and Applications of Satisfiability Testing – SAT 2007*, pages 56–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-72788-0\_9.
- 20 Bjørnar Luteberget, Koen Claessen, Christian Johansen, and Martin Steffen. SAT modulo discrete event simulation applied to railway design capacity analysis. *Formal Methods in System Design*, 57(2):211–245, August 2021. doi:10.1007/s10703-021-00368-2.
- 21 Irene Moser and Sanaz Mostaghim. The automotive deployment problem: A practical application for constrained multiobjective evolutionary optimisation. In *IEEE Congress on Evolutionary Computation*, pages 1–8, Barcelona, Spain, July 2010. IEEE. doi:10.1109/CEC.2010.5585991.
- 22 Ahmed Nasrallah, Akhilesh S. Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham ElBakoury. Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research. *IEEE Communications Surveys & Tutorials*, 21(1):88–145, 2019. doi:10.1109/COMST.2018.2869350.
- 23 Mitra Nasri, Sanjoy Baruah, Gerhard Fohler, and Mehdi Kargahi. On the Optimality of RM and EDF for Non-Preemptive Real-Time Harmonic Tasks. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems - RTNS '14*, pages 331–340, Versailles, France, 2014. ACM Press. doi:10.1145/2659787.2659806.

- 24 Asef Nazari, Dhananjay Thiruvady, Aldeida Aleti, and Irene Moser. A mixed integer linear programming model for reliability optimisation in the component deployment problem. *Journal of the Operational Research Society*, 67(8):1050–1060, August 2016. doi:10.1057/jors.2015.119.
- 25 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming—CP 2007*, pages 529–543. Springer, 2007.
- 26 Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via Lazy Clause Generation. *Constraints*, 14(3):357–391, September 2009. doi:10.1007/s10601-008-9064-x.
- 27 Marie Pelleau, Antoine Miné, Charlotte Truchet, and Frédéric Benhamou. A Constraint Solver Based on Abstract Domains. In *Verification, Model Checking, and Abstract Interpretation*, pages 434–454. Springer, 2013. doi:10.1007/978-3-642-35873-9\_26.
- 28 R. Queck. Analysis of Ethernet AVB for automotive networks using Network Calculus. In *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*, pages 61–67, July 2012. doi:10.1109/ICVES.2012.6294261.
- 29 RealTime-at-Work. Rta-pegase, 2022. URL: <https://www.realtimework.com/rta-pegase/>.
- 30 Andrea Rendl, Tias Guns, Peter J. Stuckey, and Guido Tack. MiniSearch: a solver-independent meta-search language for MiniZinc. In *Principles and Practice of Constraint Programming*, pages 376–392. Springer, 2015. doi:10.1007/978-3-319-23219-5\_27.
- 31 Pierre Schaus and Renaud Hartert. Multi-Objective Large Neighborhood Search. In *Principles and Practice of Constraint Programming*, volume 8124, pages 611–627. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-40627-0\_46.
- 32 Christian Schulte, Guido Tack, and Mikael Lagerkvist. *Modeling and Programming with Gecode*, 2020.
- 33 Joseph Scott. *Other Things Besides Number: Abstraction, Constraint Propagation, and String Variable Types*. PhD thesis, Acta Universitatis Upsaliensis, Uppsala, 2016. OCLC: 943721122.
- 34 Seyed Mohammadhossein Tabatabaee, Marc Boyer, Jean-Yves Le Boudec, and Jörn Migge. Efficient and accurate handling of periodic flows in time-sensitive networks. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 303–315, 2023. doi:10.1109/RTAS58335.2023.00031.
- 35 Pierre Talbot, Éric Monfroy, and Charlotte Truchet. Modular Constraint Solver Cooperation via Abstract Interpretation. *Theory and Practice of Logic Programming*, 20(6):848–863, 2020. doi:10.1017/S1471068420000162.
- 36 Dhananjay Thiruvady, I. Moser, Aldeida Aleti, and Asef Nazari. Constraint Programming and Ant Colony System for the Component Deployment Problem. *Procedia Computer Science*, 29:1937–1947, 2014. doi:10.1016/j.procs.2014.05.178.
- 37 Tindell, Hansson, and Wellings. Analysing real-time communications: controller area network (CAN). In *Proceedings Real-Time Systems Symposium REAL-94*, pages 259–263, San Juan, Puerto Rico, 1994. IEEE Comput. Soc. Press. doi:10.1109/REAL.1994.342710.
- 38 K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: An NP-Hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992. doi:10.1007/BF00365407.
- 39 P. M. Yomsi, D. Bertrand, N. Navet, and R. Davis. Controller Area Network (CAN): Response Time Analysis with Offsets. In *9th IEEE International Workshop on Factory Communication Systems*, pages 43–52, United States, May 2012. IEEE. doi:10.1109/WFCS.2012.6242539.

## **A** MiniZinc Model

We describe the full MINIZINC constraint model implementing the mathematical model given in Section 2. We first give the parameters of the model with the corresponding mathematical notations in blue comments:

```

% I. The hardware graph  $\langle H, L, hc, lc \rangle$ .
int: locations;
set of int: LOCATIONS = 1..locations;           % H
int: num_links;
set of int: NUM_LINKS = 1..num_links;          % L
array[LOCATIONS] of int: cpu_capacity;          % hc
array[NUM_LINKS] of int: capacity;              % lc

% II. The software graph  $\langle S, Com, sc, cc \rangle$ .
% Com is implicitly represented by the adjacency matrix coms where
%   coms[si][sj] = 0 if the services si and sj do not communicate.
int: services;
set of int: SERVICES = 1..services;            % S
array[SERVICES] of int: services_cpu_usage;     % sc
array[SERVICES, SERVICES] of int: coms;        % cc

% III. The path function
% shortest_path[hi, hj] contains all the edges belonging to the shortest path
%   between hi and hj.
% Interestingly, we do not need to know the order of the edges on the
%   shortest path, thus we can use a set.
array[LOCATIONS, LOCATIONS] of set of NUM_LINKS: shortest_path;

% IV. Not part of the mathematical specification: this is to display the
%   solutions with locations and services names instead of indexes.
array[LOCATIONS] of string: locations2names;
array[SERVICES] of string: services2names;

```

The decision variable is the function  $d : S \rightarrow H$  which is modelled as a MiniZinc array:

```

array[SERVICES] of var LOCATIONS: services2locs; % d : S → H

```

The constraints are defined using intermediate arrays of variables to simplify their definitions.

```

% I. CPU load constraint.
%  $\forall h \in H, \sum_{s \in d^{-1}(h)} sc(s) \leq hc(h)$ 
array[LOCATIONS] of var int: cpu_usage;
constraint forall(l in LOCATIONS)
  (cpu_usage[l] =
    sum(s in SERVICES)
      (services_cpu_usage[s] * (services2locs[s] == l)))
;
constraint forall(l in LOCATIONS)(cpu_usage[l] >= 0 /\ cpu_usage[l] <=
  cpu_capacity[l]);

% II. Network load constraint.
%  $\forall \ell \in L, \sum_{c \in Com} com(c, \ell) \leq lc(\ell)$ 
array[NUM_LINKS] of var int: slack;
constraint forall(link in NUM_LINKS)(
  slack[link] = capacity[link] -
    sum(s1, s2 in SERVICES)(
      coms[s1, s2] * (link in shortest_path[services2locs[s1],
        services2locs[s2]])
    ));
% Then we ensure the slack is always greater or equal to 0.
constraint forall(link in NUM_LINKS)(slack[link] >= 0 /\ slack[link] <=
  capacity[link]);

```

## 34:20 Constraint Programming with External Worst-Case Traversal Time Analysis

The multi-objective aspect of the problem is not treated within the MINIZINC model itself, but by the MiniZinc Python interface. To communicate which objectives we seek to minimize, we use a special array variable `objs` describing all three objectives described in Section 2.1.

```
array[1..3] of var int: objs;

% min maxh∈H ∑s∈d-1(h) sc(s)
constraint objs[1] = max(1 in LOCATIONS)(cpu_usage[1]);

% min maxℓ∈L (∑c∈Com com(c,ℓ))/lc(ℓ)
% We use an intermediate array charge and channeling constraint to
% represent the charge of a link in percentage.
array[NUM_LINKS] of var 0..100: charge;
constraint forall(link in NUM_LINKS)(charge[link] == (capacity[link] -
    slack[link]) div (capacity[link] div 100));
constraint objs[2] = max(link in NUM_LINKS)(charge[link]);

% min |d(S)|
constraint objs[3] = sum(1 in LOCATIONS)(cpu_usage[1] > 0);
```

# Efficient Enumeration of Fixed Points in Complex Boolean Networks Using Answer Set Programming

Van-Giang Trinh<sup>1</sup> ✉ 

LIS, Aix-Marseille University, Marseille, France

Belaid Benhamou ✉

LIS, Aix-Marseille University, Marseille, France

Sylvain Soliman<sup>1</sup> ✉ 

Lifeware team, Inria Saclay, Palaiseau, France

---

## Abstract

Boolean Networks (BNs) are an efficient modeling formalism with applications in various research fields such as mathematics, computer science, and more recently systems biology. One crucial problem in the BN research is to enumerate all fixed points, which has been proven crucial in the analysis and control of biological systems. Indeed, in that field, BNs originated from the pioneering work of R. Thomas on gene regulation and from the start were characterized by their asymptotic behavior: complex attractors and fixed points. The former being notably more difficult to compute exactly, and specific to certain biological systems, the computation of stable states (fixed points) has been the standard way to analyze those BNs for years. However, with the increase in model size and complexity of Boolean update functions, the existing methods for this problem show their limitations. To our knowledge, the most efficient state-of-the-art methods for the fixed point enumeration problem rely on Answer Set Programming (ASP). Motivated by these facts, in this work we propose two new efficient ASP-based methods to solve this problem. We evaluate them on both real-world and pseudo-random models, showing that they vastly outperform four state-of-the-art methods as well as can handle very large and complex models.

**2012 ACM Subject Classification** Computing methodologies → Logic programming and answer set programming; Applied computing → Computational biology; Applied computing → Systems biology

**Keywords and phrases** Computational systems biology, Boolean network, Fixed point, Answer set programming

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.35

**Supplementary Material** *Software (Source Code)*: <https://github.com/giang-trinh/fASP>  
archived at `swh:1:dir:86a458f51d92dda88a71499b7e6bea7178377da0`

**Funding** *Van-Giang Trinh*: This work is supported by Institut Carnot STAR, Marseille, France.

## 1 Introduction

In molecular biology, the regulation of the transcription of a gene is the process by which a cell modulates the conversion of its DNA into RNA. Transcription is a vital process in all living organisms and leads to orchestrating the whole gene activity. Its regulation can take many different forms, mostly affecting the binding of the RNA polymerase on the DNA.

The lack of precise quantitative information about transcriptional regulation and the sigmoid nature of its kinetics led, about fifty years ago, to the idea to represent models of gene regulation as discrete event systems. Those gene regulation networks use thresholds or equivalently logical functions to represent the different regulations [22, 39, 41, 40]. Over the years, Boolean Network (BN) modelling has proven that it can bring powerful analyses and

---

<sup>1</sup> Corresponding authors





corresponding insight to the many cases where enough quantitative biological data is not available [48], even for modelling post-transcriptional mechanisms. This is even more true for very large models where such data is frequently missing and led to a constant increase in size of logical models *à la* Thomas [2] and more and more complex logical formulae to describe the dynamics of those models.

Besides simulation, the analysis of such models is mostly based on *attractor* computation, since those correspond roughly to observable biological phenotypes [48]. An attractor of a BN is a minimal set of states from which the dynamics of this BN cannot escape once entered [39, 48]. An attractor of size one is called a stable state or *fixed point*. Otherwise, it is called a cyclic attractor or complex attractor. To date, the analysis of the set of fixed points of a BN remains a very useful tool in understanding the behavior of those complex biological models. This is not only due to the fact that in some cases the full computation of complex attractors remains intractable, but also because for many biological systems, the expected long-term behavior is not cyclic (as in the Cell Cycle, or Circadian rhythms for instance) but rather a stabilization to an observable *phenotype* (cell differentiation, apoptosis, proliferation, signal transduction, protein transcription, etc.). See for instance [33, 15, 13, 43] for some recent publications using stable states as main validation. It is also worth noting that the fixed point computation is the crucial starting point for several state-of-the-art methods for computing complex attractors of BNs [21, 42].

Answer Set Programming (ASP) [19] has been widely applied in the field of computational systems biology [46] because of its declarative characteristics as well as strong tools' support [18]. Very early, ASP has been used to model biological networks [14, 37]. Since BNs have become a popular modeling formalism in systems biology, it is naturally that ASP has been quickly applied to modeling and analysis of BNs. One of the first connections between ASP and BNs is the theoretical work by [26], but nowadays we can find in the literature many references showing the successful application of ASP to model and reason over biological systems modeled as BNs. The notable use of ASP in the analysis of BNs ranges from enumerating fixed points [28, 1, 34], enumerating or approximating attractors [30, 28, 1, 34], and inferring BNs from biological data [35, 46, 47, 12], to controlling BNs [27, 47].

There is a rich history of research on enumerating fixed points of BNs since this modeling formalism was proposed [22, 39]. The fixed point enumeration problem has attracted researchers from various communities and many methods have been proposed [29]. We can classify the existing methods into the following main approaches: algorithmic [29], structure-based [10, 45, 25, 7], Boolean resolution-based [24, 32, 31], integer linear programming-based [5], and ASP-based [28, 1, 34]. A more detailed summary of the existing methods shall be given in Section 3. Note however that with the increase in model size and complexity of Boolean update functions, the existing methods for this problem show their limitations [29]. One reason is that they require an intermediate representation of the original BN that may be computationally expensive or even intractable to obtain, e.g., prime implicants [28], transition-based representations [1], disjunctive normal forms [34].

Inspired by the above elements along with the fact that the most recent and most efficient fixed point enumeration methods all rely on ASP, in this work we propose two new ASP-based methods for efficiently enumerating all fixed points of a BN. The first method is based on conjunctive ASP, and the second method is a modification of the first one to handle the case of a large number of source nodes. If a BN has many source nodes, its number of fixed points may be extremely large, leading to both long running time and high memory consumption. The main advantage of the two proposed methods is that they rely on negative normal forms of Boolean functions whose computation is more efficient than that of other intermediate

representations used by the previous methods. After some preliminaries on BNs and the problem of enumerating their fixed points, we present the two new ASP-based methods and benchmark them against four other state-of-the-art tools. The experimental results on both real-world and pseudo-random models show that they vastly outperform the state-of-the-art and can handle very large and complex models.

## 2 Preliminaries

We start by recalling some classical definitions.

### 2.1 Boolean networks

► **Definition 1** (Boolean network). A Boolean Network (BN) [40] is a pair  $\mathcal{N} = (V, F)$  where:

- $V = \{v_1, \dots, v_n\}$  is the set of nodes. We use  $v_i$  to denote both the node  $v_i$  and its associated Boolean variable.
- $F = \{f_1, \dots, f_n\}$  is the set of Boolean update functions. Each function  $f_i$  is associated with node  $v_i$  and satisfies  $f_i: \mathbb{B}^{|IN(v_i)|} \mapsto \mathbb{B}$  where  $\mathbb{B} = \{0, 1\}$  and  $IN(v_i)$  denotes the set of input nodes of  $v_i$ . If  $v_j \in IN(v_i)$ , we say that there is a regulation between  $v_j$  and  $v_i$ , and  $v_j$  is a regulator of  $v_i$ . Note that a node  $v_i \in V$  is called a source node if and only if  $f_i$  is an identity function on Boolean variable  $v_i$  (i.e.,  $f_i = v_i$ ).

► **Example 2.** We give a BN  $\mathcal{N} = (V, F)$ , where  $V = \{v_1, v_2\}$  and  $F = \{f_1, f_2\}$  with  $f_1 = (v_1 \wedge v_2) \vee (\neg v_1 \wedge \neg v_2)$ ,  $f_2 = (v_1 \wedge v_2) \vee (\neg v_1 \wedge \neg v_2)$ .

► **Definition 3** (local monotonicity). A Boolean function is locally-monotonic if it can be represented by a formula in Disjunctive Normal Form (DNF) in which all occurrences of any given literal are either negated or non-negated [34].

A BN is said to be locally-monotonic if all its update functions are locally-monotonic. Otherwise, this model is said to be non-locally-monotonic.

The BN of Example 2 is non-locally-monotonic.

### 2.2 Fixed points

A state  $x \in \mathbb{B}^n$  is as a mapping  $x: V \mapsto \mathbb{B}$  that assigns either 0 (inactive) or 1 (active) to each node. We also write  $x_i$  to denote  $x(v_i)$  for short and for simplicity we write  $f_i(x)$  even when  $IN(v_i) \subsetneq V$ , i.e.,  $IN(v_i)$  does not contain some nodes of  $V$ .

► **Definition 4** (fixed point). A fixed point of  $\mathcal{N}$  is a state  $s$  such that  $s_i = f_i(s)$  for every  $v_i \in V$ .

The state space of the BN of Example 2 includes four states: 00, 01, 10, and 11. However, this BN has only one fixed point: 11.

### Complexity

Note that the fixed points do not depend on the choice of *update scheme* of the BN, they are the same for synchronous, asynchronous or even generalized updates [20]. These *update schemes* are what precisely defines a transition relation on states from the update functions. In general they allow one (asynchronous), all (synchronous) or any number (generalized) of nodes to change their value  $v_i$  to their update value specified by  $f_i$ . However, if running a simulation in the synchronous update is a feasible way to find *the only* reachable fixed point starting from a completely known initial state, this does not scale up to big networks with many source nodes with unknown values, or to other update schemes.

From a theoretical view point, the problems of detecting a fixed point and enumerating all fixed points of a general BN have been shown to be respectively NP-hard and #P-hard [3]. In fact, for general BNs, there is no existing method that works faster than  $k \times 2^n$  for any  $k \geq 1$  [29].

### 3 Related work

The recent review of Mori and Akutsu [29] shows quite well that because of the above complexity result, a certain amount of work has been done on restricted versions of the problem, limiting the type of BN to simpler regulations like [6, 23], or simpler Boolean formulae like for instance nested canalizing functions [4]. However, when working with real-world models, built by biologists, such restrictions are often impossible to enforce.

Hence, various methods exploiting the structure of a BN have been proposed, using feedback vertex sets [3, 7], subspaces [10], graph-reductions for low connectivity [45], network decomposition [8, 25], etc. Unfortunately, these still do not scale to the size of the most recent BNs (above 1000 nodes) with average connectivity and complex logical formulae.

Other methods with broader generality are also common, using classical Boolean resolution techniques, like BDD or SAT. That is the case for instance of the BioLQM library [31] that is at the core of the GINsim Boolean modelling tool [24] and of the CoLoMoTo Docker images [32]. Since integer linear programming is another useful method to efficiently solve Boolean constraints, it has been applied to addressing the fixed point enumeration [5]. The evaluation in [5] shows that this method can handle well models of up to 200 nodes with small average connectivity.

However, the most recent and most efficient fixed point enumeration methods all rely on ASP [19]. This is probably due to the fact that it links the efficiency of SAT for the Boolean constraint solving, having adapted and implemented some techniques like lazy clause generation to the point of winning certain categories of the SAT competition, and the ease of enumeration of all solutions, which is crucial here, in a declarative language.

More precisely, while there is indeed a direct encoding of the fixed-point problem into SAT [29], it creates two issues. First a SAT solver needs to convert the original Boolean formula into a CNF. It is of course possible to use a polynomial transformation like our conjunctive ASP encoding (see Section 4) or Tseitin's transformation, but this introduces auxiliary variables. This in turn leads to enumerating models that encode no fixed points or other redundant models that encode the same fixed points. A step to eliminate spurious and redundant SAT models is therefore necessary to guarantee the correctness and this would add complexity to the SAT/CP approach. In contrast, the ASP approach can avoid the above issue because of the stable model semantics, i.e., only searching for minimal Herbrand models, since the set of Herbrand models one-to-one corresponds to the set of SAT/CP models, whereas the set of minimal ones one-to-one corresponds directly to the set of fixed points.

One of the first connections between ASP and BNs is the theoretical work by [26], but nowadays ASP is used for many different BN analyses, from computing fixed point as we will show, to trap-spaces [28] which are an approximation of complex attractors, and even for representing sets of BNs [12].

By constraining the number of ground atoms in a stable model, the trap-space computation method [28] was adapted to compute fixed points. Note however that this method still requires to compute prime implicants of a Boolean function, and the number of prime implicants may be exponential in the number of inputs of this function. Moreover, the computation of prime implicants of a Boolean function is also a computationally demanding task, and gets intractable when the number of source nodes exceeds 10.

It is worth noting that Paulevé *et al.* [34] have proposed a new method for computing minimal trap spaces/fixed points that can avoid computing prime implicants. This method has been implemented in the tool `mpbn`<sup>2</sup> demonstrated in [34] for handling medium-sized models from the literature and very large synthetic models (up to 100,000 nodes) with respect to minimal trap spaces. Although there is no benchmark designed for the fixed point computation in [34], `mpbn` should handle well very large models with respect to fixed points. However, there are two drawbacks limiting the applicability of `mpbn`. First, it requires that the original BN is locally-monotonic. The class of locally-monotonic BNs is too small as compared to the class of all possible BNs, since a BN is non-locally-monotonic if just one of its Boolean functions is non-locally-monotonic (see Definition 3). Moreover, we also found many non-locally-monotonic Boolean models in the literature (see Section 5 for some of them). Second, it requires a DNF of a Boolean function. Note that obtaining a single DNF may be exponential in the size of the Boolean function (i.e., the number of inputs  $|IN|$  of this Boolean function).

Not using the concept of trap spaces, the method by [1] characterizes fixed points of a Boolean network as *dead* configurations (or deadlocks) of its corresponding Automata Network (AN). ANs are formal models similar to Petri nets with transitions representing the updates of the whole system. A transition includes the current configuration, the next configuration, and the condition for enabling this transition. A configuration is said to be dead if and only if there is no transition whose enabling condition is satisfied by this configuration. Then the above characterization is encoded as an ASP. This method has been reported to be able to handle well large-scale models [1]. However, its bottleneck lies in the construction of the corresponding AN, which in general requires to obtain two DNFs for each Boolean variable, one for the update function  $f_i$  and one for its negation  $\neg f_i$ .

## 4 Answer set programming-based methods

We will now describe the two new ASP-based encodings that we propose for the fixed point enumeration problem.

### 4.1 Conjunctive encoding

Let  $\mathcal{N} = (V, F)$  be a BN. We intend to build an ASP encoding for  $\mathcal{N}$  such that a stable model of the encoded ASP (say  $\mathcal{L}$ ) is equivalent to a fixed point of  $\mathcal{N}$ . First, for each node  $v_i$ , we introduce two atoms  $p_i$  and  $n_i$ . The translation from a stable model  $A$  of  $\mathcal{L}$  to a state  $x$  of  $\mathcal{N}$  is that for every  $v_i \in V$ ,  $x_i = 1$  if and only if  $p_i \in A$ , and  $x_i = 0$  if and only if  $n_i \in A$ . The below ASP rules ensure that a stable model of  $\mathcal{L}$  corresponds to a state of  $\mathcal{N}$ :

$$\text{: } \neg p_i, n_i. \tag{1}$$

meaning  $\text{false} \leftarrow p_i \wedge n_i$ , and

$$p_i, n_i. \tag{2}$$

meaning  $p_i \vee n_i \leftarrow \text{true}$ , for every  $v_i \in V$ . Recall that state  $x$  is a fixed point of  $\mathcal{N}$  if and only if the relation  $x_i = f_i(x)$  holds for all  $v_i \in V$ . This relation can be seen as the conjunction of  $x_i \leftarrow f_i(x)$  and  $\neg x_i \leftarrow \neg f_i(x)$ , which can be characterized by  $v_i \leftarrow f_i$  and  $\neg v_i \leftarrow \neg f_i$ , respectively. Hereafter, we show how to encode the two parts for every  $v_i \in V$  as conjunctive ASP rules.

<sup>2</sup> <https://github.com/bnediction/mpbn>

## 35:6 Efficient Enumeration of Fixed Points in Complex Boolean Networks

For the former part, to avoid the presence of negation, we convert  $f_i$  into its Negative Normal Form (NNF). The NNF is obtained by recursively applying De Morgan laws until all negations that remain are on literals. We now associate ASP rules to  $f_i$  as follows:

$$\gamma(v_i) :- \gamma(\text{NNF}(f_i)).$$

where we define function  $\gamma$  as

$$\begin{aligned} \gamma(v_i) &= p_i \\ \gamma(\neg v_i) &= n_i \\ \gamma\left(\bigwedge_{1 \leq j \leq J} \alpha_j\right) &= \gamma(\alpha_1), \dots, \gamma(\alpha_J) \\ \gamma\left(\bigvee_{1 \leq j \leq J} \alpha_j\right) &= \text{aux}_k \text{ where } \text{aux}_k \text{ is a new atom and for each } j \text{ add the rule } \text{aux}_k :- \gamma(\alpha_j). \end{aligned}$$

Note that  $k$  is here a global counter starting from 1 and will be increased by 1 after a new atom is created. For the latter part, we similarly apply the above process with  $\neg v_i$  and  $\neg f_i$  instead of respectively  $v_i$  and  $f_i$ . Note that it also requires to convert  $\neg f_i$  into an NNF first.

Listing 1 shows the encoded ASP of the BN shown in Example 2 following the above encoding. Atoms  $p_1$  and  $n_1$  (resp.  $p_2$  and  $n_2$ ) correspond to node  $v_1$  (resp.  $v_2$ ). Lines 1 and 2 represent the rules shown in Equation (1) and Equation (2), respectively. The rules for the part  $v_1 \leftarrow f_1$  (resp.  $\neg v_1 \leftarrow \neg f_1$ ) are presented in Lines 4–5 (resp. Lines 6–8). Similarly, Lines 10–14 represent the rules for node  $v_2$ . Line 16 indicates that we omit auxiliary atoms in the resulting stable models. This ASP has only one stable model:  $\{p_1, p_2\}$ , which corresponds to the sole fixed point (i.e., 11) of  $\mathcal{N}$ .

■ **Listing 1** Conjunctive ASP encoding for the BN shown in Example 2.

```

:- p1, n1.                :- p2, n2.                1
p1, n1.                   p2, n2.                2
                                                                    3
p1 :- aux1.               4
aux1 :- p1, p2.           aux1 :- n1, n2.         5
n1 :- aux2, aux3.         6
aux2 :- n1.               aux2 :- n2.             7
aux3 :- p1.               aux3 :- p2.             8
                                                                    9
p2 :- aux4.               10
aux4 :- p1, p2.           aux4 :- n1, n2.         11
n2 :- aux5, aux6.         12
aux5 :- n1.               aux5 :- n2.             13
aux6 :- p1.               aux6 :- p2.             14
                                                                    15
#show p1/0. #show n1/0.   #show p2/0. #show n2/0.   16

```

We here discuss the advantages of the above ASP encoding. First, the ASP  $\mathcal{L}$  has no negation besides Equation (1) that may hinder the efficiency of ASP solvers. Note also that obtaining the NNF of a Boolean function is linear in its size and thus quite efficient. Second, except the rules of Equation (2), all the rules in  $\mathcal{L}$  are conjunctive. This is the reason we name the above encoding as the *conjunctive* encoding.

The next result shows that our ASP encoding is sound and complete with respect to the fixed points of a BN.

► **Proposition 5.** *The set of stable models of  $\mathcal{L}$  one-to-one corresponds to the set of fixed points of  $\mathcal{N}$ .*

**Proof.** Note that the only negations in the conjunctive encoding come from the state constraints of Equation (1), i.e.,  $\text{:-} p_i, n_i$ . Since this equation is coupled with Equation (2), all stable models will contain exactly one of  $p_i$  or  $n_i$ , i.e., they will correspond with the state  $x$  where  $x_i$  is true or false depending on the atom in the stable model.

The remainder of the ASP has no negation, no disjunction in heads and is logically equivalent to  $\forall v_i \in V, x_i = f_i(x)$  via the introduction of some existentially quantified  $aux_j$ . Hence there is for each state  $x$  at most a single stable model, equal to the smallest Herbrand model containing the  $p_i$  or  $n_i$  corresponding to  $x$ , the necessary  $aux_j$ , and that satisfies  $x = f(x)$  by construction.

All stable models of  $\mathcal{L}$  will therefore correspond one to one with states  $x$  described by the  $p_i$  or  $n_i$  that are true, and such that  $\forall v_i \in V, x_i = f_i(x)$ , hence they correspond one to one with fixed points of  $\mathcal{N}$ . ◀

## 4.2 Source encoding

Recall that the number of fixed points of a BN may be extremely large if it has many source nodes (see Definition 1). Specifically, that number may be exponential in the number of source nodes. In the conjunctive encoding as well as those of the state-of-the-art methods [28, 1, 34], a resulting stable model always corresponds to a single fixed point. Hence, having many source nodes is actually a bottleneck for these methods. To overcome this issue, we propose a new encoding based on the conjunctive encoding of Section 4.1.

Let  $\mathcal{L}_c$  be the encoded ASP of the BN following the conjunctive encoding. Let  $V^s$  be the set of source nodes of the BN. Our main idea is to group two stable models  $A_1$  and  $A_2$  of  $\mathcal{L}_c$  into a stable model  $A$  if they only differ in the atoms corresponding to a source node. More specifically, if there is a source node  $v_i$  such that  $p_i \in A_1, n_i \in A_2$ , and  $A_1 \setminus \{p_i\} = A_2 \setminus \{n_i\}$ , then we can group  $A_1$  and  $A_2$  into a stable model  $A$  such that  $A = A_1 \cup \{n_i\} = A_2 \cup \{p_i\}$ . For example, let  $A_1$  and  $A_2$  be the stable models respectively corresponding to fixed points 01 and 11 of the BN shown in Example 7. Herein,  $A_1 = \{n_1, p_2\}$  and  $A_2 = \{p_1, p_2\}$ . They can be grouped into stable model  $A = \{p_1, n_1, p_2\}$ . Now, we add  $A$  to the set of stable models of  $\mathcal{L}_c$ , and then repeat the grouping process until there is no new stable model. Note that this process introduces more stable models than before, e.g., we need to consider all  $A, A_1$ , and  $A_2$ . However, the new stable model covers all the fixed points represented by the two stable models constituting it. Hence, we just need to consider the maximal set-inclusion stable models. We adjust the conjunctive encoding to make the above approach fully automated in the ASP solver. Since the new encoding aims to handle the case of many source nodes, we name it the *source* encoding.

Similar to the conjunctive encoding, for each node  $v_i \in V$ , we introduce two atoms  $p_i$  and  $n_i$ . For each node in  $V$ , we associate to this node the ASP rules identical to those of the conjunctive encoding. For each  $v_i \in V^s$ , we remove from the encoded ASP (say  $\mathcal{L}_s$ ) the rule of Equation (1), i.e.,  $\text{:-} p_i, n_i$ . By releasing this condition,  $\mathcal{L}_s$  can have Herbrand models that contain both  $p_i$  and  $n_i, v_i \in V^s$ . To make such Herbrand models to be stable models of  $\mathcal{L}_s$ , we add to  $\mathcal{L}_s$  the choice rules  $\{p_i\}$ . and  $\{n_i\}$ . for all  $v_i \in V^s$ . A choice rule  $\{p_i\}$ . is equivalent to the rule  $p_i \text{ :- not not } p_i$ . where not denotes the default negation. Finally, we add to  $\mathcal{L}_s$  the rules  $\#show p_i/0$  and  $\#show n_i/0$  for all  $v_i \in V$ , which indicate that we omit auxiliary atoms in the resulting stable models.

Note that a stable model of  $\mathcal{L}_s$  may correspond to multiple fixed points of the BN. Given a stable model  $A$  of  $\mathcal{L}_s$ , the set  $F$  of fixed points represented by  $A$  is specified as follows. For each node  $v_i \in V^s$ , it can receive value 0 if  $n_i \in A$  and  $p_i \notin A$ , value 1 if  $p_i \in A$  and  $n_i \notin A$ , both if  $n_i \in A$  and  $p_i \in A$ . For each node  $v_i \in V \setminus V^s$ , it can receive value 0 if  $n_i \in A$  and value 1 if  $p_i \in A$ . Then,  $F$  is equivalent to the set of all possible value combinations of all nodes.

The next result shows that our ASP encoding is correct.

► **Proposition 6.** *The set of maximal set-inclusion stable models of  $\mathcal{L}_s$  with respect to the shown atoms exactly covers all fixed points of the BN.*

**Proof.** First, we see that  $\mathcal{L}_s$  still contains all Herbrand models of  $\mathcal{L}_c$ . However, by releasing the condition of Equation (1) for all source nodes,  $p_i$  and  $n_i$  can both appear in a Herbrand model of  $\mathcal{L}_s$  if  $v_i \in V^s$ . Assume that  $A_1$  and  $A_2$  are two stable models of  $\mathcal{L}_s$  such that  $p_i \in A_1$ ,  $n_i \in A_2$ , and  $A_1 \setminus \{p_i\} = A_2 \setminus \{n_i\} = B$  for a node  $v_i \in V^s$ . The ASP rules corresponding to node  $v_i$  are tautology. In all the remaining rules,  $p_i$  and  $n_i$  never appear in the left hand side. Hence,  $A = B \cup \{p_i, n_i\}$  is also a Herbrand model of  $\mathcal{L}_s$ . By introducing the choice rules for all source nodes, such Herbrand models will be stable models of  $\mathcal{L}_s$ . Hence, the set of all stable models of  $\mathcal{L}_s$  is equivalent to the set of stable models obtained by the grouping approach on the set of stable models of  $\mathcal{L}_c$ . All fixed points represented by a stable model of  $\mathcal{L}_s$  are also covered by a maximal set-inclusion stable model of  $\mathcal{L}_s$  with respect to the shown atoms (corresponding to nodes in the BN). Hence, the set of all maximal set-inclusion stable models of  $\mathcal{L}_s$  exactly covers all fixed points of the BN. ◀

For illustration, consider the BN shown in Example 7. Listing 2 shows the encoded ASP of this BN following the above encoding. Atoms  $p_1$  and  $n_1$  (resp.  $p_2$  and  $n_2$ ) correspond to node  $v_1$  (resp. node  $v_2$ ). Line 1 represents the rule of Equation (2). Note that the rule  $:-p_1, n_1$  is removed because  $v_1$  is a source node. Instead, two choice rules  $\{p_1\}$ . and  $\{n_1\}$ . are added in Line 2. Line 3 represents the rules shown in Equation (1) and Equation (2) of node  $v_2$ . The rules for the parts  $v_1 \leftarrow f_1$  and  $\neg v_1 \leftarrow \neg f_1$  are presented in Line 5. Similarly, Lines 7–9 represent the rules for node  $v_2$ . Line 11 indicates that we omit auxiliary atoms in the resulting stable models. The encoded ASP has four stable models including:  $\{n_1, n_2\}$  (corresponding to fixed point 00),  $\{n_1, p_2\}$  (corresponding to fixed point 01),  $\{p_1, p_2\}$  (corresponding to fixed point 11), and  $\{p_1, n_1, p_2\}$  (corresponding to fixed points 01 and 11). From these results, we can see that the encoded ASP has two maximal set-inclusion stable models ( $\{n_1, n_2\}$  and  $\{p_1, n_1, p_2\}$ ), which cover all the fixed points of the BN.

► **Example 7.** We give a BN  $\mathcal{N} = (V, F)$ , where  $V = \{v_1, v_2\}$  and  $F = \{f_1, f_2\}$  with  $f_1 = v_1, f_2 = v_1 \vee v_2$ .  $v_1$  is a source node of  $\mathcal{N}$ .  $\mathcal{N}$  has three fixed points: 00, 01, 11.

■ **Listing 2** Source ASP encoding for the BN shown in Example 7.

```

p1, n1.                                     1
{p1}.                                       {n1}.                                     2
:- p2, n2.                                  p2, n2.                                  3
                                                                                               4
p1 :- p1.                                   n1 :- n1.                                 5
                                                                                               6
p2 :- aux1.                                  7
aux1 :- p1.                                  aux1 :- p2.                               8
n2 :- n1, n2.                                9
                                                                                               10
#show p1/0.      #show n1/0.      #show p2/0.      #show n2/0.                               11

```

### 4.3 Post-processing

The set of maximal set-inclusion stable models of the encoded ASP  $\mathcal{L}_s$  with respect to the shown atoms can be seen as a meta result from which we can easily retrieve the fixed points. Note that a stable model can be group-able with multiple ones, and thus the sets of fixed points of two maximal set-inclusion stable models of  $\mathcal{L}_s$  can intersect. In other words, a fixed point of the BN may be included in two different maximal set-inclusion stable models of  $\mathcal{L}_s$ . Hence, it is not straightforward to obtain the number of fixed points, which by itself is a common difficult problem related to #SAT (*model-counting*, see [44]). Moreover, many more precise analysis questions can be answered, but not directly from the above meta result. For example, given a state  $x^n$  on  $V \setminus V^s$ , return the set of states  $x^s$  on  $V^s$  such that  $(x^n, x^s)$  is a fixed point of the BN. With this motivation, we propose a post-processing step as follows.

We maintain a hash table (denoted by  $H$ ) with as *key* a state  $x^n$  on  $V \setminus V^s$  and as associated *value* the set of states  $x^s$  on  $V^s$  such that  $(x^n, x^s)$  is a fixed point of the BN. For each stable model  $A$  of the meta result, we extract  $x^n$  from it by simply checking either  $p_i$  or  $n_i$  belongs to  $A$  for all  $v_i \in V \setminus V^s$ . For each  $v_i \in V^s$ ,  $x_i$  can receive value 0 if  $n_i \in A$  and value 1 if  $p_i \in A$ . Then we get the set of states on  $V^s$  (denoted by  $S^s$ ) as the combinations of all possible values of all  $v_i \in V^s$ . If  $x^n$  is not a key of  $H$ , we just add the pair  $(x^n, S^s)$  to  $H$ . Otherwise, we replace the current associated value of  $x^n$  in  $H$  by the union of it and  $S^s$  because the current associated value and  $S^s$  may intersect. When there are many nodes in  $V^s$ ,  $S^s$  may contain a large number of states, even exponential in the number of nodes in  $V^s$ . Binary Decision Diagrams (BDDs) [11] are an efficient data structure for representing a set of states as well as performing set operations. Hence, we store  $S^s$  as a BDD where each node  $v_i \in V^s$  corresponds to a BDD variable.

Now, let us show how to answer some analysis questions from the hash table  $H$ . First, for the example analysis question mentioned in the beginning of Subsection 4.3, if  $x^n$  is not a key of  $H$ , we return the empty set. Otherwise, the set of all states on  $V^s$  can be easily retrieved from the BDD as the value of  $x^n$  in  $H$ . Specifically, we list all satisfying valuations of this BDD. Analogously, by traversing all items in  $H$ , we can also answer the question, given a combination of values on source nodes, to return the set of fixed points of the BN restricted by this combination.

Second, we can efficiently compute the number of fixed points in the BN based on  $H$ . For a given BDD, we can efficiently compute its number of satisfying valuations. Such procedure is linear in the number of nodes in this BDD [11]. Since any two keys in  $H$  are distinct, the sets of fixed points on  $V$  corresponding to them are also distinct. Hence, the number of fixed points of the BN is equivalent to the sum of all numbers of satisfying valuations for all BDDs in  $H$ . This is in contrast with most model-counting problems, where approximate methods are usually necessary to minimize the number of calls to a SAT solver.

Note that an alternative approach might be to represent the hash table as a sole BDD where every node of the BN corresponds to a BDD variable. As such, the new BDD will have  $|V|$  variables, whereas each BDD in  $H$  has  $|V^s|$  variables. Recall that the size of a BDD may be exponential in its number of variables [11]. Hence, splitting a BDD into multiple BDDs with smaller numbers of variables is a good strategy in most cases, especially when  $|V|$  is much larger than  $|V^s|$ . Indeed, in our experiments shown in Section 5, we observed that the alternative approach using a sole BDD gave poorer performance than the approach using a hash table in most cases.

To conclude this section, we discuss the advantages of the source encoding. First, it inherits all the advantages of the conjunctive encoding, which can be seen as its core part. Second, the number of maximal set-inclusion stable models of the encoded ASP may be much



smaller than the number of fixed points. Hence, the proposed method can have memory benefits as compared to other ASP-based methods. Third, with a much smaller number of stable models, the solving time can be much smaller. Although the proposed method needs to spend time for the post-processing, it can have running-time benefits as compared to other ASP-based methods. The second and third points shall be analyzed in our experiments shown in Section 5. Note that even if some of the extreme examples of Section 5 do not correspond to plausible biological processes, finding common patterns among them could give biological insights. Since the source method can provide a compact representation of the set of fixed points, it can be used to find such common patterns. For example, we can know that a key-value of the hash table corresponds to a set of fixed points that only differ in values of source nodes. This is a very useful result since different source nodes values usually represent different (environmental) states of a biological system. Some analyses made possible by this encoding are: listing all fixed points, counting the number of fixed points, listing/counting all fixed points under a specific value combination of source nodes, listing the *core* fixed points projected on only normal nodes (this is exactly the list of key values of the hash table).

## 5 Experimental results

We implemented the newly proposed methods as a Python package named `fASP`<sup>3</sup>. For convenience, we name the method based the conjunctive encoding presented in Section 4.1 as `fASP-conj` and the method based on the source encoding for the case of source nodes presented in Section 4.2 as `fASP-src`. To evaluate their performance, we compared them with the four state-of-the-art methods for fixed point enumeration in BNs, including `PyBoolNet` [28], `mpbn` [34], `AN-ASP` [1], and `FPCollector` [7].

We benchmarked all the compared methods on the `BBM` repository<sup>4</sup>, a collection of real-world Boolean models from various sources used in systems biology. `BBM` consists of 211 models, peaking at 321 variables, 1100 regulations, and 133 source nodes, respectively. Furthermore, we also included a selection of 13 real-world models that are not covered by the `BBM` repository. The BNs of this selection peak at 3158 variables, 43642 regulations, and 237 source nodes, respectively. To our knowledge, the `BBM` repository along with this selected set is a highly representative sample of Boolean models currently available in the literature.

To solve the ASP problems, we used the same ASP solver `Clingo` [18] and the same configuration as that used in `PyBoolNet`, `mpbn`, and `AN-ASP`. Specifically for computing maximal set-inclusion stable models, we used the configuration `-heuristic=Domain -enum-mod=domRec -dom-mod=3` (subset maximality, equivalent to the deprecated `-dom-pref=32 -heuristic=domain -dom-mod=7` used by `PyBoolNet`). We ran all the benchmarks on a machine whose environment is CPU: Intel® Core™ i9-11950H 2.60GHz × 16, 16 GB DDR4 RAM, Ubuntu 20.04.5 LTS. Note that for the methods `PyBoolNet`, `mpbn`, `AN-ASP`, and `fASP-conj`, we can control the maximum number of fixed points returned because a resulting stable model corresponds to a single fixed point. In contrast, `FPCollector` requires to compute all fixed points, and `fASP-src` allows its user to set the maximum number of resulting stable models but not of resulting fixed points. Since a model can have a huge number of fixed points due to many source nodes, which might not be biologically plausible, obtaining a sample of those can prove to be very useful to invalidate the model and lead to further refinement. Hence, to obtain a relevant, reliable and fair comparison, in our

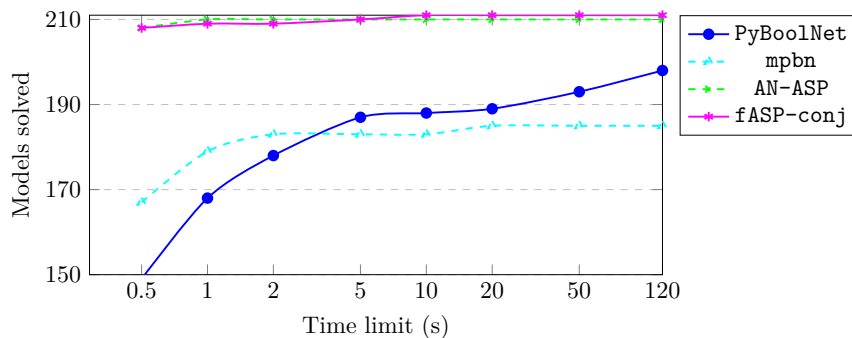
<sup>3</sup> The source code and benchmarks are freely accessible at <https://github.com/hiang-trinh/fASP>.

<sup>4</sup> <https://github.com/sybila/biodivine-boolean-models>

benchmarks, we searched for both all the fixed points and the first 1000 fixed points for each model. In addition, existing analysis shown in the literature usually revolves then around fixing some source nodes to plausible values and reducing the model accordingly. Although this approach biologically makes sense, it relies on potentially arbitrary decisions, and *hides away* critical modelling choices that were actually not part of the original BN. Hence, we did not fix specific values for source nodes in all the considered models. Finally, we set a time limit of two minutes (resp. ten minutes) for each model with respect to enumerating the first 1000 fixed points (resp. all the fixed points).

## 5.1 BBM repository

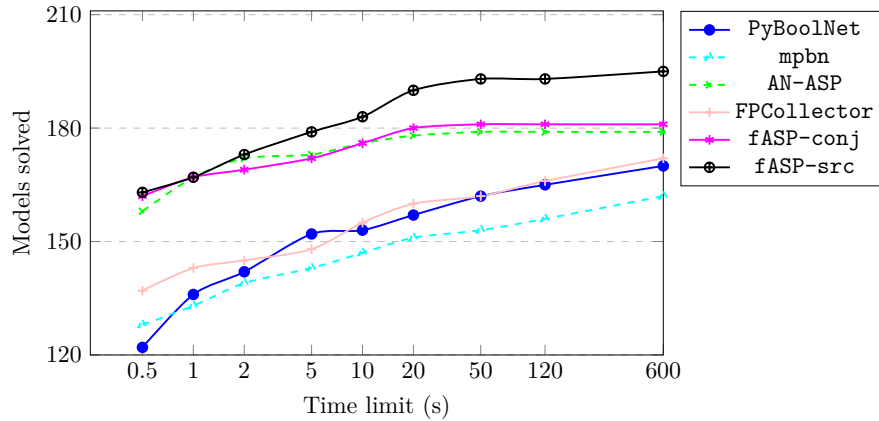
With regards to enumerating the first 1000 fixed points, the number of BBM models solved within two minutes of each method is: PyBoolNet (198), mpbn (185), AN-ASP (210), and fASP-conj (211). Note that there are 24 non-locally-monotonic models that mpbn cannot handle. Figure 1 shows the cumulative numbers of models solved by the four compared methods with respect to enumerating the first 1000 fixed points. As it can be observed, the AN-ASP and fASP-conj methods are comparable and they vastly outperform the PyBoolNet and mpbn methods. For every time limit, their numbers of solved models are always greater than those of PyBoolNet and mpbn, especially the difference is large for the time limit of 0.5s. In particular, AN-ASP could handle all but one model and fASP-conj could handle all models within 10s.



■ **Figure 1** Cumulative numbers of BBM models solved by the four compared methods with respect to enumerating the first 1000 fixed points.

With regards to enumerating all the fixed points, the number of BBM models solved within ten minutes of each method is: FPCollector (172), PyBoolNet (170), mpbn (162), AN-ASP (179), fASP-conj (181), and fASP-src (195). We can see that fASP-src solved more models than all the other methods. One can note that the models for which fASP-src was the only successful method, have extremely large numbers of fixed points, each time because of many source nodes. This confirms our expectation when proposing fASP-src to deal with the case of many source nodes. Upon closer inspection, Figure 2 depicts the cumulative numbers of models solved by the six studied methods with respect to enumerating all the fixed points. As it can be observed, the AN-ASP and fASP-conj methods are still comparable and they outperform the FPCollector, PyBoolNet, and mpbn methods. Furthermore, for every time limit, the number of solved models using fASP-src is always the highest.

It is worth noting that, most models in the BBM repository have moderate numbers of nodes ( $n < 200$ ) and quite simple Boolean functions. Hence, the difference in performance among the three best methods in terms of BBM models (i.e., AN-ASP, fASP-conj, and fASP-src) is not much exhibited. We shall test more in the following subsections.



■ **Figure 2** Cumulative numbers of BBM models solved by the six compared methods with respect to enumerating all the fixed points.

## 5.2 Selected real-world models

Table 1 shows the running time of the four compared methods on the 13 selected BNs with respect to enumerating the first 1000 fixed points. Columns  $n$ ,  $s$ , and  $|A|$  denote the number of nodes, the number of source nodes, and the number of fixed points, respectively. “DNF” means that the method did not finish the computation within the time limit of two minutes. “NM” indicates a non-locally-monotonic model. We can easily see that `fASP-conj` is the best method in all models as it is much faster than all the other methods on every model. In particular, it only took 35.24s to enumerate the first 1000 fixed points of the hardest model (i.e., the Cell-Cycle-Control model), whereas none of the other methods could finish the computation. Upon closer inspection, `AN-ASP` is the second best method with running time less than 1s in most models. However, it ran quite slowly on the Insulin model, could not handle the Yeast-Pheromone model, and even got an *Out of Memory* (OOM) error on the Cell-Cycle-Control model before finishing the automata network construction. We note that the above problems all come from the bottleneck of `AN-ASP`, i.e., the high number of transitions of the corresponding AN. Finally, consistent with the observations reported in the previous subsection, `PyBoolNet` and `mpbn` still performed much slower than the `AN-ASP` and `fASP-conj` methods, and there are four non-locally-monotonic models (out of the 13) that `mpbn` cannot handle.

Table 2 shows the running time of the six benchmarked methods on the selected BNs with respect to enumerating all the fixed points. Column  $|A|$  denotes the number of fixed points, and a “?” denotes the case where none of the competing methods returned the result. We can first see that for 6 of the 13 models, none of the competing methods returned all the fixed points. This is clearly because the numbers of fixed points of these models are extremely large. `FPCollector` only succeeded for two models, with each less than 100 nodes. Of course, it is difficult to handle models of larger size. `PyBoolNet` and `mpbn` only succeeded for three models each. All these models are easy for the other methods. Hereafter, we shall present closer inspection on the three most efficient methods: `AN-ASP`, `fASP-conj`, and `fASP-src`.

First, `fASP-conj` is still much faster than `AN-ASP` for all the models where they both succeeded. Second, `fASP-src` is the sole method that could return all the fixed points of the T-Cell-Co-Receptor model within the time limit. Note that the number of fixed points of this model is huge, and it is apparent that none of the other methods can handle it. Moreover,

■ **Table 1** Timing comparisons (in seconds) among PyBoolNet (PBN), mpbn, AN-ASP, and fASP-conj (f-conj) on the selected models from the literature for enumerating the first 1000 fixed points.

	model	$n$	$s$	$ A $	PBN	mpbn	AN-ASP	f-conj
1	Cell-Cycle-Control [36]	3158	61	1000 <sup>+</sup>	DNF	DNF	OOM	35.24
2	EMT-Mechan [38]	136	4	82	6.02	0.17	0.22	0.05
3	EMT-Mechan-TGFbeta [38]	150	6	478	7.12	0.39	0.27	0.08
4	Alzheimer [2]	762	237	1000 <sup>+</sup>	DNF	NM	0.55	0.31
5	MAPK [2]	181	37	1000 <sup>+</sup>	8.81	0.91	0.23	0.09
6	Mast-Cell-Activation [2]	73	19	1000 <sup>+</sup>	0.07	0.30	0.14	0.03
7	Cholocystokinin [2]	383	74	1000 <sup>+</sup>	0.60	1.47	0.29	0.13
8	HOG [36]	43	5	1000 <sup>+</sup>	11.08	NM	0.17	0.07
9	Insulin [36]	82	7	1000 <sup>+</sup>	DNF	DNF	13.65	0.21
10	Leishmania [17]	342	81	1000 <sup>+</sup>	DNF	1.33	0.37	0.14
11	Pluripotency [49]	36	7	412	DNF	NM	0.22	0.02
12	T-Cell-Co-Receptor [16]	206	39	1000 <sup>+</sup>	DNF	0.79	0.28	0.09
13	Yeast-Pheromone [36]	246	17	1000 <sup>+</sup>	DNF	NM	DNF	1.01

■ **Table 2** Timing comparisons (in seconds) among FPCollector (FP), PyBoolNet (PBN), mpbn, AN-ASP, fASP-conj (f-conj), and fASP-src (f-src) on the selected models from the literature with respect to enumerating all the fixed points.

	model	$ A $	FP	PBN	mpbn	AN-ASP	f-conj	f-src
1	Cell-Cycle-Control	?	DNF	DNF	DNF	OOM	DNF	DNF
2	EMT-Mechan	82	DNF	5.82	0.15	0.30	0.05	0.08
3	EMT-Mechan-TGFbeta	478	DNF	7.05	0.38	0.26	0.08	0.09
4	Alzheimer	?	DNF	DNF	NM	OOM	OOM	DNF
5	MAPK	?	DNF	DNF	DNF	OOM	OOM	DNF
6	Mast-Cell-Activation	524288	5.45	DNF	145.87	11.84	8.96	6.34
7	Cholocystokinin	?	DNF	OOM	DNF	OOM	OOM	DNF
8	HOG	25632	149.20	17.77	NM	0.54	0.79	1.81
9	Insulin	563200	DNF	DNF	DNF	81.93	36.73	107.60
10	Leishmania	?	DNF	DNF	DNF	OOM	OOM	DNF
11	Pluripotency	412	1.30	DNF	NM	0.27	0.05	0.02
12	T-Cell-Co-Receptor	441039454208	DNF	DNF	DNF	OOM	OOM	349.15
13	Yeast-Pheromone	?	DNF	DNF	NM	DNF	DNF	DNF

both AN-ASP and fASP-conj met the OOM error, which confirms the memory advantage of fASP-src. For the six other models where it succeeded, fASP-src is comparable to AN-ASP and fASP-conj, with a bit slower running time on average. It is apparent because fASP-src suffers from the overhead of its post-processing based on BDDs. Indeed, we confirmed that in most of these models (also of other models considered in our experiments), the ASP solving time of fASP-src is negligible and most of its running time was spent for the post-processing. Note that the running time of this BDD-based post-processing depends on several factors, such as, the number of resulting stable models, the number of source nodes (the number of BDD variables), and the BDD variable ordering.

We also note that for the six failed models, the difference among the three best methods (i.e., AN-ASP, fASP-conj, and fASP-src) is not clear because they all did not finish or met the OOM error. Hence, we conduct new analysis on these failed models by restricting the maximum number of stable models for fASP-src. We then use the number of fixed points obtained by fASP-src as the maximum number of fixed points for the case of AN-ASP or fASP-conj. The experimental settings are the same as those used for the case of enumerating

all fixed points. Table 3 shows the results of this analysis. For `fASP-src`, we consider two maximum numbers: 100 and 200. For each case, Column  $|A|$  denotes the number of fixed points obtained by `fASP-src`, and the other columns denotes the running time of the corresponding methods to obtain that number of fixed points. We can see that for the four models (Cell-Cycle-Control, Alzheimer, Cholocystokinin, and Leishmania), both `AN-ASP` and `fASP-conj` met the OOM error, whereas `fASP-src` can handle them in reasonable time. For the MAPK and Yeast-Pheromone models, `fASP-src` is much faster than both `fASP-conj` and `AN-ASP`. This is also true for the Mast-Cell-Activation and Pluripotency models in the case of enumerating all fixed points (see Table 2). The above observations confirm an advantage in both memory and run-time of using `fASP-src` in the case of many source nodes.

■ **Table 3** Timing comparisons (in seconds) among `AN-ASP`, `fASP-conj` (`f-conj`), and `fASP-src` (`f-src`) on the six failed models.

model	100				200			
	$ A $	f-src	AN-ASP	f-conj	$ A $	f-src	AN-ASP	f-conj
1 Cell-Cycle-Control	159744	21.98	OOM	OOM	323584	23.73	OOM	OOM
2 Alzheimer	$> 10^{32}$	10.89	OOM	OOM	$> 10^{32}$	29.93	OOM	OOM
3 MAPK	31744	0.19	1.74	1.37	52480	0.42	2.77	2.11
4 Cholocystokinin	$> 10^{13}$	0.28	OOM	OOM	$> 10^{13}$	0.55	OOM	OOM
5 Leishmania	$> 10^{13}$	0.46	OOM	OOM	$> 10^{13}$	0.79	OOM	OOM
6 Yeast-Pheromone	1552	0.91	DNF	1.14	3152	0.91	DNF	1.45

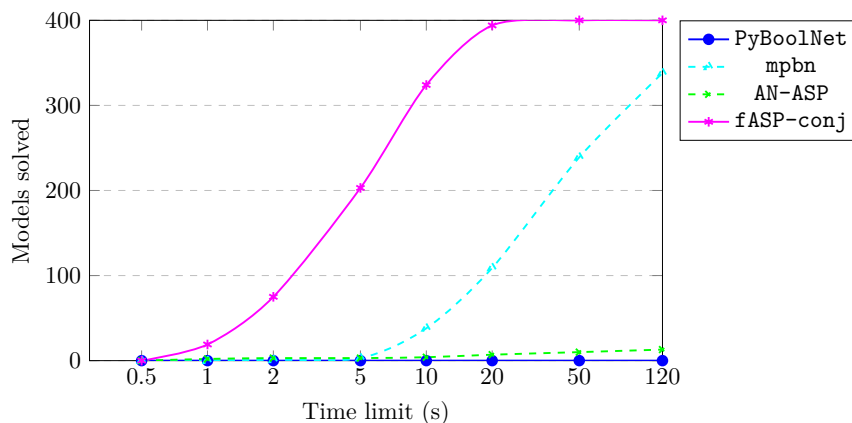
### 5.3 Pseudo-random models

The results on the 224 real-world models reported in the two previous subsections draw a quite clear picture about the performance of the six compared methods. However, we observed that there is only one model with more than 1000 nodes (i.e., the Cell-Cycle-Control model). Moreover, in most of them, the Boolean functions are quite simple, even sometimes just simple conjunctions/disjunctions of literals. The reason for these facts may be that the modelers were restricted by the limited performance of the tools supported at the time they created the models. This is not the case of the Cell-Cycle-Control model, since its authors only conducted simulations instead of formal analysis [36]. Since in the present work we target large and complex BNs, we set out to test the performance of our proposed methods on larger and more complicated models than the ones available in the literature to date. Specifically, we wanted to test models with 1000 or more nodes and Boolean functions in complicated forms. Such a model is arguably not possible to achieve yet with hand-made modeling, even with a fully or semi-automated inference technique [2], but might be in the near future.

To create a benchmark set of larger and more complex models, we decided to generate pseudo-random models following the generation approach proposed by the research group who created and is maintaining the `BBM` repository. This generation approach is described in detail in [9] and its implementation is provided at [https://github.com/daemontus/artifact\\_cav2021](https://github.com/daemontus/artifact_cav2021). In general, it generates Boolean models structurally similar to the real-world models in the `BBM` repository. To ensure this structural similarity, the generator uses a node-degree distribution sampled from the `BBM` repository, as opposed to other theoretical random network models. Once the regulators of a node are specified, its Boolean function is generated by randomly choosing between  $\wedge$  and  $\vee$  when connecting the positive/negative

literals of the regulators. Note that this option does not cover the full spectrum of possible Boolean functions, but it can make the generated Boolean functions complicated enough for evaluation.

In the end, we created 400 pseudo-random models ranging from 1000 to 5000 variables, 4145 to 63507 regulations, and 127 to 1171 source nodes, respectively. We then tested all the competing methods on these models. We first reported that all the competing methods failed to obtain all the fixed points as they quickly met the OOM error. The reason is that the number of all fixed points is actually too large due to a lot of source nodes ( $> 100$ ). Hence, we here only searched for the first 1000 fixed points, which might also be more biologically relevant. The time limit for each model was set to two minutes. The other settings are the same as those used in the two previous subsections.



■ **Figure 3** Cumulative numbers of pseudo-random models solved by the competing methods with respect to enumerating the first 1000 fixed points.

With regards to enumerating the first 1000 fixed points, the number of pseudo-random models solved within two minutes of each competing method is: `PyBoolNet` (0), `mpbn` (338), `AN-ASP` (13), and `fASP-conj` (400). `PyBoolNet` could not handle any model, and it failed at the phase of computing prime implicants in most cases. This is not surprising since the models are large in size and the formulae quite complex. Interestingly, `mpbn` could handle far more models than `AN-ASP`. This can be explained by the fact that the number of transitions in the corresponding AN is very large in most models, whereas the size of the DNF for each Boolean function is still moderate. Moreover, the models are all locally-monotonic, which is an assumption of the generator [9]. Upon closer inspection, Figure 3 depicts the cumulative numbers of pseudo-random models solved by the four competing methods with respect to enumerating the first 1000 fixed points. We can see that `fASP-conj` is the best method as it vastly outperforms the three other methods for every time limit except the time limit of 0.5s where all the methods could handle no model. In particular, it could handle all the 400 pseudo-random models within 50s.

## 6 Conclusion and future work

In this work we have proposed two new ASP-based methods called `fASP-conj` and `fASP-src` for efficiently enumerating fixed points of Boolean networks, which are crucial in modeling and analysis of biological systems. `fASP-conj` is based on conjunctive ASP and `fASP-src` is a modification of `fASP-conj` to handle the case of a large number of source nodes. The

main advantage of these methods is that they only rely on NNFs of Boolean functions, which are much more efficient to obtain than other representations used by previous methods (e.g., prime-implicants, disjunctive normal forms, automata networks). The main advantage of `fASP-src` is that it provides a more compact representation of the results based on BDDs, which can give both memory and run-time benefits. We have also formally proved the correctness of the above new methods.

We have then benchmarked their performance against the four other state-of-the-art tools: `FPCollector`, `PyBoolNet`, `mpbn`, and `AN-ASP`. The experimental results on both real-world and pseudo-random models show that the new methods vastly outperform the state-of-the-art as they can robustly handle various types of large and complex models, whereas the other methods cannot. In particular, they can handle models of up to 5000 nodes with very complicated Boolean update functions. In terms of enumerating the first 1000 fixed points (resp. all fixed points), the experimental results show that `fASP-conj` is the best (resp. second best) method. `fASP-src`, which is based on `fASP-conj`, shows its superiority to all the other methods in enumerating all the fixed points of models with many source nodes.

Boolean network models of biological systems usually contain many source nodes, which might be hard to avoid in the modeling process [2]. Currently, there are many such models that `fASP-src` cannot handle. Hence, improving `fASP-src` is necessary. Note that, in some cases, the number of auxiliary atoms in the core encoding of `fASP-conj` and `fASP-src` can be reduced. Such optimization will be looked into in the future. Furthermore, we also plan to extend the methods proposed in this present paper to those for computing trap spaces of Boolean networks, which are more general than fixed points and useful approximations for complex attractors in Boolean networks. It is crucial because the state-of-the-art methods for the trap space computation are all unable to robustly handle large and complex models, for instance, the models used in our experiments here.

---

## References

- 1 Emna Ben Abdallah, Maxime Folschette, Olivier F. Roux, and Morgan Magnin. ASP-based method for the enumeration of attractors in non-deterministic synchronous and asynchronous multi-valued networks. *Algorithms Mol. Biol.*, 12(1):20:1–20:23, 2017. doi:10.1186/s13015-017-0111-2.
- 2 Sara Sadat Aghamiri, Vidisha Singh, Aurélien Naldi, Tomás Helikar, Sylvain Soliman, Anna Niarakis, and Jinbo Xu. Automated inference of Boolean models from molecular interaction maps using CaSQ. *Bioinform.*, 36(16):4473–4482, 2020. doi:10.1093/bioinformatics/btaa484.
- 3 Tatsuya Akutsu, Satoru Kuhara, Osamu Maruyama, and Satoru Miyano. A system for identifying genetic networks from gene expression patterns produced by gene disruptions and overexpressions. *Genome Informatics*, 9:151–160, 1998. doi:10.11234/gi1990.9.151.
- 4 Tatsuya Akutsu, Avraham A. Melkman, Takeyuki Tamura, and Masaki Yamamoto. Determining a singleton attractor of a Boolean network with nested canalizing functions. *J. Comput. Biol.*, 18(10):1275–1290, 2011. doi:10.1089/cmb.2010.0281.
- 5 Tatsuya Akutsu, Yang Zhao, Morihiro Hayashida, and Takeyuki Tamura. Integer programming-based approach to attractor detection and control of Boolean networks. *IEICE Trans. Inf. Syst.*, 95-D(12):2960–2970, 2012. doi:10.1587/transinf.E95.D.2960.
- 6 Julio Aracena. Maximum number of fixed points in regulatory Boolean networks. *Bull. Math. Biol.*, 70:1398–1409, 2008. doi:10.1007/s11538-008-9304-7.
- 7 Julio Aracena, Luis Cabrera-Crot, and Lilian Salinas. Finding the fixed points of a Boolean network from a positive feedback vertex set. *Bioinform.*, 37(8):1148–1155, 2021. doi:10.1093/bioinformatics/btaa922.

- 8 Ferhat Ay, Günhan Gülsoy, and Tamer Kahveci. Finding steady states of large scale regulatory networks through partitioning. In *International Workshop on Genomic Signal Processing and Statistics*, pages 1–4. IEEE, 2010. doi:10.1109/GENSIPS.2010.5719669.
- 9 Nikola Benes, Lubos Brim, Samuel Pastva, and David Safránek. Computing bottom SCCs symbolically using transition guided reduction. In *International Conference on Computer Aided Verification*, pages 505–528. Springer, 2021. doi:10.1007/978-3-030-81685-8\_24.
- 10 Nikolaos Berntenis and Martin Ebeling. Detection of attractors of large Boolean networks via exhaustive enumeration of appropriate subspaces of the state space. *BMC Bioinform.*, 14(1):1–10, 2013. doi:10.1186/1471-2105-14-361.
- 11 R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.
- 12 Stéphanie Chevalier, Vincent Noël, Laurence Calzone, Andrei Yu. Zinovyev, and Loïc Paulevé. Synthesis and simulation of ensembles of Boolean networks for cell fate decision. In *International Conference on Computational Methods in Systems Biology*, pages 193–209. Springer, 2020. doi:10.1007/978-3-030-60327-4\_11.
- 13 Karla Fabiola Corral-Jara, Camille Chauvin, Wassim Abou-Jaoudé, Maximilien Grandclaudon, Aurélien Naldi, Vassili Soumelis, and Denis Thieffry. Interplay between SMAD2 and STAT5A is a critical determinant of IL-17A/IL-17F differential expression. *Mol. Biomed.*, 2(1):9, 2021. doi:10.1186/s43556-021-00034-3.
- 14 Steve Dworschak, Susanne Grell, Victoria J. Nikiforova, Torsten Schaub, and Joachim Selbig. Modeling biological networks by action languages via answer set programming. *Constraints An Int. J.*, 13(1-2):21–65, 2008. doi:10.1007/s10601-007-9031-y.
- 15 Swann Floc’Hlay, Maria Dolores Molina, Céline Hernandez, Emmanuel Haillet, Morgane Thomas-Chollier, Thierry Lepage, and Denis Thieffry. Deciphering and modelling the TGF- $\beta$  signalling interplays specifying the dorsal-ventral axis of the sea urchin embryo. *Dev.*, 148(2):dev189944, 2021. doi:10.1242/dev.189944.
- 16 Piyali Ganguli, Saikat Chowdhury, Rupa Bhowmick, and Ram Rup Sarkar. Temporal protein expression pattern in intracellular signalling cascade during T-cell activation: A computational study. *J. Biosci.*, 40(4):769–789, September 2015. doi:10.1007/s12038-015-9561-1.
- 17 Piyali Ganguli, Saikat Chowdhury, Shomeek Chowdhury, and Ram Rup Sarkar. Identification of Th1/Th2 regulatory switch to promote healing response during leishmaniasis: a computational approach. *EURASIP J. Bioinform. Syst. Biol.*, 2015:13, 2015. doi:10.1186/s13637-015-0032-7.
- 18 Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The Potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011. doi:10.3233/AIC-2011-0491.
- 19 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.
- 20 Carlos Gershenson. Classification of random Boolean networks. In *Artificial Life VIII, Proceedings of the Eighth International Conference on Artificial Life*, pages 1–8, 2003.
- 21 Trinh Van Giang, Tatsuya Akutsu, and Kunihiko Hiraishi. An FVS-based approach to attractor detection in asynchronous random Boolean networks. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 19(2):806–818, 2022. doi:10.1109/TCBB.2020.3028862.
- 22 Leon Glass and Stuart A Kauffman. The logical analysis of continuous, non-linear biochemical control networks. *J. Theor. Biol.*, 39(1):103–129, 1973. doi:10.1016/0022-5193(73)90208-7.
- 23 Eric Goles and Lilian Salinas. Sequential operator for filtering cycles in Boolean networks. *Adv. Appl. Math.*, 45(3):346–358, 2010. doi:10.1016/j.aam.2010.03.002.
- 24 A Gonzalez Gonzalez, Aurélien Naldi, Lucas Sanchez, Denis Thieffry, and Claudine Chaouiya. GINsim: a software suite for the qualitative modelling, simulation and analysis of regulatory networks. *Biosyst.*, 84(2):91–100, 2006. doi:10.1016/j.biosystems.2005.10.003.



- 25 Changki Hong, Jeewon Hwang, Kwang-Hyun Cho, and Insik Shin. An efficient steady-state analysis method for large Boolean networks with high maximum node connectivity. *PLoS One*, 10(12):e0145734, December 2015. doi:10.1371/journal.pone.0145734.
- 26 Katsumi Inoue. Logic programming for Boolean networks. In *International Joint Conference on Artificial Intelligence*, pages 924–930. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-160.
- 27 Roland Kaminski, Torsten Schaub, Anne Siegel, and Santiago Videla. Minimal intervention strategies in logical signaling networks with ASP. *Theory Pract. Log. Program.*, 13(4-5):675–690, 2013. doi:10.1017/S1471068413000422.
- 28 Hannes Klärner, Adam Streck, and Heike Siebert. PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks. *Bioinform.*, 33(5):770–772, 2017. doi:10.1093/bioinformatics/btw682.
- 29 Tomoya Mori and Tatsuya Akutsu. Attractor detection and enumeration algorithms for Boolean networks. *Comput. Struct. Biotechnol. J.*, 20:2512–2520, 2022. doi:10.1016/j.csbj.2022.05.027.
- 30 Mushthofa Mushthofa, Gustavo Torres, Yves Van de Peer, Kathleen Marchal, and Martine De Cock. ASP-G: an ASP-based method for finding attractors in genetic regulatory networks. *Bioinform.*, 30(21):3086–3092, 2014. doi:10.1093/bioinformatics/btu481.
- 31 Aurélien Naldi. BioLQM: a Java toolkit for the manipulation and conversion of logical qualitative models of biological networks. *Front. Physiol.*, 9:1605, 2018. doi:10.3389/fphys.2018.01605.
- 32 Aurélien Naldi, Pedro T. Monteiro, Christoph Müssel, Hans A. Kestler, Denis Thieffry, Ioannis Xenarios, Julio Saez-Rodriguez, Tomás Helikar, and Claudine Chaouiya. Cooperative development of logical modelling standards and tools with CoLoMoTo. *Bioinform.*, 31(7):1154–1159, 2015. doi:10.1093/bioinformatics/btv013.
- 33 Karen J Nuñez-Reza, Aurélien Naldi, Arantza Sánchez-Jiménez, Ana V Leon-Apodaca, M Angélica Santana, Morgane Thomas-Chollier, Denis Thieffry, and Alejandra Medina-Rivera. Logical modelling of in vitro differentiation of human monocytes into dendritic cells unravels novel transcriptional regulatory interactions. *Interface Focus*, 11(4):20200061, 2021. doi:10.1098/rsfs.2020.0061.
- 34 Loïc Paulevé, Juraj Kolčák, Thomas Chatain, and Stefan Haar. Reconciling qualitative, abstract, and scalable modeling of biological networks. *Nat. Commun.*, 11(1), August 2020. doi:10.1038/s41467-020-18112-5.
- 35 Alexandre Rocca, Nicolas Mobilia, Eric Fanchon, Tony Ribeiro, Laurent Trilling, and Katsumi Inoue. ASP for construction and validation of regulatory biological networks. *Logical Modeling of Biological Systems*, pages 167–206, 2014.
- 36 Jesper C Romers and Marcus Krantz. rxncon 2.0: a language for executable molecular systems biology. *BioRxiv*, page 107136, 2017. doi:10.1101/107136.
- 37 Torsten Schaub and Sven Thiele. Metabolic network expansion with answer set programming. In *International Conference on Logic Programming*, pages 312–326. Springer, 2009. doi:10.1007/978-3-642-02846-5\_27.
- 38 Emmalee Sullivan, Marlayna Harris, Arnav Bhatnagar, Eric Guberman, Ian Zonfa, and Erzsébet Ravasz Regan. Boolean modeling of mechanosensitive Epithelial to Mesenchymal Transition and its reversal. *bioRxiv*, September 2022. doi:10.1101/2022.08.29.505701.
- 39 René Thomas. Boolean formalisation of genetic control circuits. *J. Theor. Biol.*, 42:565–583, 1973. doi:10.1016/0022-5193(73)90247-6.
- 40 René Thomas. Regulatory networks seen as asynchronous automata: a logical description. *J. Theor. Biol.*, 153(1):1–23, 1991. doi:10.1016/S0022-5193(05)80350-9.
- 41 René Thomas and Richard d’Ari. *Biological feedback*. CRC press, 1990.
- 42 Van-Giang Trinh, Kunihiko Hiraishi, and Belaid Benhamou. Computing attractors of large-scale asynchronous Boolean networks using minimal trap spaces. In *ACM International Conference*

- on *Bioinformatics, Computational Biology and Health Informatics*, pages 13:1–13:10. ACM, 2022. doi:10.1145/3535508.3545520.
- 43 Eirini Tsirvouli, Felicity Ashcroft, Berit Johansen, and Martin Kuiper. Logical and experimental modeling of cytokine and eicosanoid signaling in psoriatic keratinocytes. *iScience*, 24(12):103451, 2021. doi:10.1016/j.isci.2021.103451.
  - 44 Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
  - 45 Alan Veliz-Cuba, Boris Aguilar, Franziska Hinkelmann, and Reinhard C. Laubenbacher. Steady state analysis of Boolean molecular network models via model reduction and computational algebra. *BMC Bioinform.*, 15(1):1–8, 2014. doi:10.1186/1471-2105-15-221.
  - 46 Santiago Videla, Carito Guziolowski, Federica Eduati, Sven Thiele, Martin Gebser, Jacques Nicolas, Julio Saez-Rodriguez, Torsten Schaub, and Anne Siegel. Learning Boolean logic models of signaling networks with ASP. *Theor. Comput. Sci.*, 599:79–101, 2015. doi:10.1016/j.tcs.2014.06.022.
  - 47 Santiago Videla, Julio Saez-Rodriguez, Carito Guziolowski, and Anne Siegel. caspo: a toolbox for automated reasoning on the response of logical signaling networks families. *Bioinform.*, 33(6):947–950, 2017. doi:10.1093/bioinformatics/btw738.
  - 48 Rui-Sheng Wang, Assieh Saadatpour, and Reka Albert. Boolean modeling in systems biology: an overview of methodology and applications. *Phys. Biol.*, 9(5):055001, 2012. doi:10.1088/1478-3975/9/5/055001.
  - 49 Ayako Yachie-Kinoshita, Kento Onishi, Joel Ostblom, Matthew A Langley, Eszter Posfai, Janet Rossant, and Peter W Zandstra. Modeling signaling-dependent pluripotency with Boolean logic to predict cell fate transitions. *Mol. Syst. Biol.*, 14(1):e7952, 2018. doi:10.15252/msb.20177952.



# Guided Bottom-Up Interactive Constraint Acquisition

Dimosthenis C. Tsouros ✉ 

KU Leuven, Belgium

Senne Berden ✉ 

KU Leuven, Belgium

Tias Guns ✉ 

KU Leuven, Belgium

---

## Abstract

Constraint Acquisition (CA) systems can be used to assist in the modeling of constraint satisfaction problems. In (inter)active CA, the system is given a set of candidate constraints and posts queries to the user with the goal of finding the right constraints among the candidates. Current interactive CA algorithms suffer from at least two major bottlenecks. First, in order to converge, they require a large number of queries to be asked to the user. Second, they cannot handle large sets of candidate constraints, since these lead to large waiting times for the user. For this reason, the user must have fairly precise knowledge about what constraints the system should consider. In this paper, we alleviate these bottlenecks by presenting two novel methods that improve the efficiency of CA. First, we introduce a bottom-up approach named GROWACQ that reduces the maximum waiting time for the user and allows the system to handle much larger sets of candidate constraints. It also reduces the total number of queries for problems in which the target constraint network is not sparse. Second, we propose a probability-based method to guide query generation and show that it can significantly reduce the number of queries required to converge. We also propose a new technique that allows the use of openly accessible CP solvers in query generation, removing the dependency of existing methods on less well-maintained custom solvers that are not publicly available. Experimental results show that our proposed methods outperform state-of-the-art CA methods, reducing the number of queries by up to 60%. Our methods work well even in cases where the set of candidate constraints is 50 times larger than the ones commonly used in the literature.

**2012 ACM Subject Classification** Computing methodologies → Discrete space search; Computing methodologies → Supervised learning by classification; Theory of computation → Constraint and logic programming; Computing methodologies → Active learning settings

**Keywords and phrases** Constraint acquisition, Constraint learning, Active learning, Modelling

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.36

**Supplementary Material** *Software:* <https://github.com/Dimosts/ActiveConLearn>

**Funding** This research received funding from the European Research Council (ERC) under the EU Horizon 2020 research and innovation programme (Grant No 101002802, CHAT-Opt) and from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101070149, project Tuples

## 1 Introduction and related work

Constraint programming (CP) is considered one of the main paradigms for solving combinatorial problems, with many successful applications in a variety of domains. However, there are still challenges to be faced in order for CP technology to become even more widely used. One of the most important challenges is to ease the modeling process. The current assumption in CP is that the user first models the problem and that a solver is then used to



© Dimosthenis C. Tsouros, Senne Berden, and Tias Guns;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 36; pp. 36:1–36:20

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

solve it. However, modeling is a non-trivial task. Expressing a combinatorial problem as a set of constraints over decision variables is not straightforward and requires substantial expertise [12]. As a result, modeling is considered a major bottleneck for the widespread adoption of CP [12, 13, 14].

This obstacle has led to research into a very different approach to modeling: that of *learning* the constraint problem from data, as opposed to manually constructing it. This is the focus of the research area of *constraint acquisition (CA)*, in which CP meets machine learning. In CA, the model of a constraint problem is acquired (i.e., learned) (semi-)automatically from a set of examples of solutions, and possibly non-solutions. CA methods can be categorized as *active* or *passive* on the basis of whether a user provides feedback during learning or not.

In *passive acquisition*, a dataset of examples of solutions and non-solutions is provided by the user upfront. Based on these examples, the system learns a set of constraints modeling the problem [4, 5, 7, 9, 11, 17, 18, 19, 21]. Approaches vary in the types of constraints they are able to learn and the methodologies they employ: CONACQ.1 is a version space algorithm for learning fixed-arity constraints [7, 9, 11], ModelSeeker learns global constraints that are taken from a predefined constraint catalog [4], and COUNT-CP is a generate-and-aggregate approach that can learn expressive first-order constraints [18]. None of these approaches are robust to errors in the labeled data. To this end, SEQACQ and BAYESACQ were introduced, being robust to noise in the training set. In SEQACQ, a statistical approach based on sequential analysis is used [22], while in BAYESACQ, a naive Bayes classifier is trained, from which a constraint network is then derived [23].

In contrast to passive learning, *active* or *interactive acquisition* systems learn the constraints through interaction with the user, by asking queries. The main type of query used is the *membership query*, which asks the user to classify a given example (i.e., an assignment to the variables of the problem) as a solution or a non-solution. An early work in active CA is the Matchmaker agent [15], where users, when they answer a membership query negatively, also have to provide a violated constraint. In order to lower the expertise level required from the user, Bessiere et al. later proposed CONACQ.2 [10, 11] – an active version of CONACQ.1 that uses membership queries and does not require the user to provide any violated constraints. In [24], CONACQ.2 was in turn extended to also accept arguments regarding *why* examples should be rejected or accepted.

As the number of membership queries needed can be exponentially large for these methods [11], a new family of interactive algorithms was proposed that use *partial queries* instead [3, 8, 20, 25, 26, 27, 28, 29]. A partial query asks the user to classify a partial assignment to the variables. Using partial queries, CA systems are able to converge faster. QUACQ was the first system to use partial queries [6, 8], and was later extended into MULTIACQ [3]. MQUACQ was later introduced to reduce the number of queries needed per learned constraint [25, 29], and MQUACQ-2 further improved the performance by exploiting the structure of the constraints already learned [27].

Despite these advancements in active CA, there are still significant obstacles for the technology to become usable in practice. One of the main limitations is that it typically still requires asking a large number of queries to the user in order to find all constraints. In addition, existing systems cannot handle large sets of candidate constraints in reasonable run times, and thus require significant expertise from the user in limiting the constraints the system should consider (and thus the size of the candidate set) upfront. Finally, query generation – a highly important part of the CA process – currently requires the use of customized solvers that are not publicly available and are not as well-maintained as conventional solvers. Without the use of such customized solvers, current active CA algorithms can lead to very high query generation times or are sometimes unable to converge to the correct set of constraints when time limits are imposed [1, 25].

We focus on the above limitations, and contribute the following improvements:

- We present a novel query generation method named PQ-GEN that allows conventional constraint solvers to be used by CA algorithms while also ensuring convergence, removing the dependency on customized solvers.
- We propose a bottom-up learning approach named GROWACQ that uses any other CA algorithm to learn the constraints of an increasingly large problem. It starts learning with only a subset of variables and an associated subset of candidate constraints, and incrementally grows this set of variables and constraints. This allows it to handle significantly larger sets of candidate constraints and reduces the maximum waiting time for the user.
- Finally, we introduce a better way to *guide* the query generation process, with the goal of generating queries that learn the set of constraints faster. We propose an objective function for query generation that uses *probabilistic estimates* of whether constraints are likely to hold or not. We demonstrate the potential of this method by using a simple counting-based approach as probabilistic estimator.

The rest of the paper is structured as follows. Some background on CA is given in Section 2. Sections 3–5 present our proposed methods. An experimental evaluation is given in Section 6. Finally, Section 7 concludes the paper.

## 2 Background

We now introduce some basic notions regarding constraint satisfaction problems and interactive constraint acquisition.

### 2.1 Constraint satisfaction problems

A *constraint satisfaction problem* (CSP) is a triple  $P = (X, D, C)$ , consisting of:

- a set of  $n$  variables  $X = \{x_1, x_2, \dots, x_n\}$ , representing the entities of the problem,
- a set of  $n$  domains  $D = \{D_1, D_2, \dots, D_n\}$ , where  $D_i \subset \mathbb{Z}$  is the finite set of values for  $x_i$ ,
- a constraint set (also called constraint network)  $C = \{c_1, c_2, \dots, c_t\}$ .

A *constraint*  $c$  is a pair  $(rel(c), var(c))$ , where  $var(c) \subseteq X$  is the *scope* of the constraint and  $rel(c)$  is a relation over the domains of the variables in  $var(c)$  that specifies (implicitly or explicitly) what assignments are allowed.  $|var(c)|$  is called the *arity* of the constraint. The constraint set  $C[Y]$ , where  $Y \subseteq X$ , denotes the set of constraints from  $C$  whose scope is a subset of  $Y$ . The set of solutions of a constraint set  $C$  is denoted by  $sol(C)$ . A *redundant* or *implied* constraint  $c \in C$  is a constraint in  $C$  such that  $sol(C) = sol(C \setminus \{c\})$ .

A (partial) *assignment*  $e_Y$  is an assignment over a set of variables  $Y \subseteq X$ .  $e_Y$  is *rejected* by a constraint  $c$  iff  $var(c) \subseteq Y$  and the projection  $e_{var(c)}$  of  $e_Y$  on the variables in the scope  $var(c)$ , is not in  $rel(c)$ , that is, is not allowed by the constraint.  $\kappa_C(e_Y)$  represents the subset of constraints from  $C[Y]$  that reject  $e_Y$ , i.e.,  $\kappa_C(e_Y) = \{c \mid c \in C[Y] \wedge e_{var(c)} \notin rel(c)\}$ .

A complete assignment  $e$  that is accepted by all the constraints in  $C$  is a *solution* to  $C$ , i.e.,  $e \in sol(C)$ . A partial assignment  $e_Y$  is called a *partial solution* to  $C$  iff it is accepted by all the constraints in  $C[Y]$ . Note that a partial solution to  $C$  may not be extendable to a complete one, due to constraints not in  $C[Y]$ .

■ **Algorithm 1** Constraint Acquisition through partial queries.

---

**Input:**  $X, D, B, C_{in}$  ( $X$ : the set of variables,  $D$ : the set of domains,  $B$ : the bias,  $C_{in}$ : an optional set of known constraints)

**Output:**  $C_L$ : a learned constraint network

```

1:  $C_L \leftarrow C_{in}$ 
2: while True do
3:   Generate an  $e$  accepted by  $C_L$  and rejected by  $B$ 
4:   if  $e = \text{nil}$  then return  $C_L$  ▷ Stopping condition
5:   if  $ASK(e) = \text{Yes}$  then ▷ Ask (partial) membership query  $e$ 
6:     Remove the constraints rejecting  $e$ , namely  $\kappa_B(e)$ , from  $B$ 
7:   else
8:     Find one (or more) minimal scopes  $S$  in  $e$  for which  $|\kappa_B(e_S)| \geq 1$  and  $ASK(e_S) =$ 
       No
9:     Find all  $\{c \in C_T \mid \text{var}(c) = S\}$  through partial queries; add to  $C_L$ , remove from
        $B$ 

```

---

## 2.2 Active constraint acquisition with partial membership queries

In CA, the pair  $(X, D)$  is called the *vocabulary* of the problem at hand and is common knowledge shared by the user and the system. Besides the vocabulary, the learner is also given a *language*  $\Gamma$  consisting of *fixed-arity* constraint relations. Using the vocabulary  $(X, D)$  and the constraint language  $\Gamma$ , the system generates the *constraint bias*  $B$ , which is the set of all possible candidate constraints for the problem.

Let  $C_T$ , the target constraint network, be an unknown set of constraints such that for every assignment  $e$  over  $X$  it holds that  $e \in \text{sol}(C_T)$  iff  $e$  is a solution to the problem the user has in mind. The goal of CA is to learn a constraint set  $C_L \subseteq B$  that is equivalent to  $C_T$ . Like other works, we assume that the bias  $B$  can represent  $C_T$ , i.e., there exists a  $C \subseteq B$  s.t.  $\text{sol}(C) = \text{sol}(C_T)$ .

In active CA, the system interacts with the user while learning the constraints. A *membership query* [2] in this setting is a question  $ASK(e_X)$ , asking the user whether a complete assignment  $e_X$  is a solution to the problem that the user has in mind. A *partial query*  $ASK(e_Y)$ , with  $Y \subset X$ , asks the user to determine if  $e_Y$ , which is an assignment in  $D^Y$ , is a partial solution with respect to  $C_T[Y]$ . We use the notation  $c \in C_T$  iff  $\forall e \in D^Y$  with  $\text{var}(c) \subseteq Y \subseteq X$ ,  $ASK(e_Y) = \text{True} \implies e_{\text{var}(c)} \in \text{sol}(c)$ .

While in passive acquisition there are methods that can handle noisy answers [22, 23], this is not the case for active acquisition. For this reason, in this work, we follow the assumption that the user answers all queries correctly.

A query  $ASK(e_Y)$  is called *irredundant* iff the answer is not implied by any information already available to the system. That is, the query is irredundant iff  $e_Y$  is rejected by at least one constraint from the bias  $B$  and is not rejected by the network  $C_L$  learned thus far. The first condition captures that  $\kappa_B(e_Y)$  cannot be empty, since if  $\kappa_B(e_Y)$  would be empty, the answer to the query  $ASK(e_Y)$  would have to be “yes”, based on the assumption that  $C_T$  is representable by the constraints in  $B$ . The second condition captures that  $e_Y$  should not be rejected by any constraint in the learned network  $C_L$ , since otherwise the user would certainly answer “no” to the query.

Algorithm 1 presents the generic process followed by active CA methods with partial queries. The learned set  $C_L$  is first initialized either to the empty set or to a set of constraints given by the user that is known to be part of  $C_T$  (i.e.,  $C_{in} \subset C_T$ ) (line 1). Then the main

loop of the acquisition process begins, where, in every iteration, the system first generates an irredundant query (line 3) and posts it to the user (line 5). If the query is answered positively, then the candidate constraints from  $B$  that violate it are removed (line 6). Otherwise, the system has to find one or more constraints from  $C_T$  that violate the query. This is done in two steps. First, queries are asked to find the scope of a constraint in  $\kappa_{C_T}(e)$  (line 8). Then, queries are asked to find all constraints  $c \in C_T$  with that scope (line 9).

The acquisition process has *converged* on the learned network  $C_L \subseteq B$  iff  $C_L$  agrees with the set of all labeled examples  $E$ , and for every other network  $C \subseteq B$  that agrees with  $E$ , it holds that  $sol(C) = sol(C_L)$ . This is proved if no query could be generated at line 3, as in this case, all remaining constraints in  $B$  (if any) are redundant. If the first condition is true but the second condition has not been proved when the acquisition process finishes, *premature convergence* has occurred. This can happen when the query generation at line 3 returns  $e = nil$ , but without having proved that an irredundant query does not exist (e.g., because of a time limit).

Existing algorithms like QUACQ [6, 8], MQUACQ [25, 29] and MQUACQ-2 [27] follow this template, but differ mainly in how they implement lines 3, 8 and 9, and hence how many constraints they are able to learn in each iteration. Examples of functions used to locate the scope of a constraint (line 8) are *FindScope* [6, 8] or the more efficient *FindScope-2* [25]. To learn the constraints in the scope found (line 9), the *FindC* function is typically used [6, 8].

### 3 Using conventional solvers for query generation

Query generation (line 3 of Algorithm 1) is one of the most important parts of the CA process. It aims to find an *irredundant* membership query (i.e., a (partial) assignment that does not violate  $C_L$  but violates at least one  $c \in B$ ) that will be asked to the user. Thus, it can be formalized as follows:

$$\text{find } e_Y \text{ s.t. } e_Y \in sol(C_L[Y] \wedge \bigvee_{c_i \in B[Y]} \neg c_i),$$

which can be formulated as a CSP with variables  $Y$  and constraints  $C_L[Y] \wedge \bigvee_{c_i \in B[Y]} \neg c_i$ .

#### 3.1 Problems when using conventional solvers

In principle, this CSP could be solved using any conventional CP solver. However, this can lead to issues for the following two reasons.

**A large bias.** At the start of the acquisition process, the set of candidate constraints  $B$  can be very large. This makes the propagation of the constraint  $\bigvee_{c_i \in B[Y]} \neg c_i$  time-consuming, and severely slows down the query-generation process.

**Indirectly implied constraints.** At the end of the acquisition process, only constraints that are implied by  $C_L$  remain in  $B$ , if any. In this case, it will be impossible to generate a query that does not violate  $C_L$  and violates at least one constraint from  $B$ . However, propagation is often unable to prove such implications when they are indirect and involve multiple variables and constraints. For this reason, solvers internally end up enumerating all possible variable assignments satisfying  $C_L$  and checking if the constraint  $\bigvee_{c_i \in B[Y]} \neg c_i$  can be satisfied. This can be very time-consuming, and a time limit is usually imposed on query generation, leading to *premature convergence*.



■ **Algorithm 2** PQ-GEN: Projection-based Query Generation.

---

**Input:**  $C_L, B, l, t$  ( $B$ : the set of candidate constraints (bias),  $C_L$ : set of known constraints,  $l$ : size limit,  $t$ : time limit)

**Output:**  $e$ : the query generated

```

1: timer.start()
2:  $Y \leftarrow \bigcup_{c \in B} \text{var}(c)$ 
3: if  $|B| > l$  then
4:    $e \leftarrow \text{solve}(C_L[Y])$ 
5:   if  $\exists c \in B : e \notin \text{sol}(c)$  then
6:     return  $e$ 
7:  $e \leftarrow \text{solve}(C_L[Y] \wedge \bigvee_{c_i \in B[Y]} \neg c_i)$ 
8: if timer.end()  $< t$  then
9:    $e' \leftarrow \text{solve}(C_L[Y] \wedge \bigvee_{c_i \in B[Y]} \neg c_i, \text{maximize: } \text{obj}, \text{time limit: } t - \text{timer.end}())$ 
10:  if  $e' \neq \text{nil}$  then
11:    return  $e'$ 
12: return  $e$ 

```

---

In order to limit the large runtimes in a more advanced way than by simply imposing a time bound  $t$ , Addi et al. proposed a method using conventional solvers named TQ-GEN [1]. It iteratively tries to solve the query generation problem, by gradually reducing the number of variables taken into account by a proportion  $\alpha \in ]0, 1[$ , until a query can be generated within a small time limit  $\tau$ . This is repeated until either an irredundant query is generated, or a global time bound  $t$  is reached, leading to premature convergence. However, choosing the right hyperparameters for  $t$  and  $\tau$  is problem-specific [1] and requires tuning, and thus more interaction with the user.

### 3.2 Customized solvers

To avoid premature convergence, a CP solver can be customized to store partial assignments that satisfy every  $c \in C_L[Y]$  and violate at least one  $c \in B[Y]$  during the search. Given an objective function, such as maximizing the number of assigned variables, in every non-failing node of the search tree it will check the above property and, if fulfilled, store the best-scoring partial assignment.

As these customized solvers are guaranteed to find valid partial solutions, their use will never lead to premature convergence. In addition, finding a partial query to return is not time-consuming (especially when combined with specialized search heuristics [25]), even when the bias is large. However, such custom solvers are not publicly available and are typically not based on the latest version of state-of-the-art solvers. This also means that the corresponding active CA methods are heavily tied to those particular customized solvers.

### 3.3 Projection-based Query Generation

We now introduce a method named *Projection-based Query Generation* (PQ-GEN) that makes it possible to use state-of-the-art conventional solvers for query generation, without premature convergence. Our proposed method is shown in Algorithm 2.

**Avoiding indirectly implied constraints.** A key observation we make is that when generating a query on line 3, it might be that  $\bigcup_{c \in B} \text{var}(c) \subset X$ , that is, some variables have no more candidate constraints in  $B$ . These have become irrelevant, as both lines 6 and 8 are only

concerned with  $\kappa_B(e)$ , which will not include these variables. So, to generate an irredundant query, it is sufficient to consider only the variables in  $B$ . This is not only faster, but also avoids indirectly implied constraints, as these are indirect through variables not used in  $B$ .

Thus, our proposed query generator projects the variables down to  $Y \subseteq X$ , with  $Y = \bigcup_{c \in B} \text{var}(c)$ , thereby simplifying the problem to finding an assignment over  $Y \subseteq X$ . This will inherently result in a partial assignment when  $Y$  is a strict subset of  $X$ , without requiring a custom solver. Thus, we first compute the set of variables  $Y$  relevant to the query (line 2), and project  $C_L$  down to those variables (on lines 4 and 7). The solver then has to prove that there exists a query that satisfies  $C_L[Y]$  and violates at least one constraint from  $B$ .

**Dealing with large biases.** As mentioned above, having a large bias  $B$  can severely slow down the solver during query generation because propagating the  $\bigvee_{c_i \in B[Y]} \neg c_i$  constraint takes a long time. However, we observe that when  $B$  contains many constraints, the property that a query  $e$  violates at least one of these is usually satisfied without needing to enforce this. Hence, we propose not using this constraint when the bias is larger than some threshold (lines 3 to 6 in Algorithm 2). If in a post-hoc check, it turns out that the generated query violates at least one  $c \in B$ , it is directly returned (line 6). Otherwise, we again generated a query, this time *with* the constraint enforcing that there must exist a constraint in  $B$  that is violated (line 7).

**Optimizing the query.** The above ensures that we will always find a valid query. However, much better queries – according to some objective function – can often be found. This would take additional time, but is safe because, since a valid query has already been found, the optimization can always safely be interrupted. Given a time limit, we can hence call an optimization solver for the remaining time after a first valid query has been found (lines 8-11).

As expressed in Proposition 1, Algorithm 2 is correct.

► **Proposition 1.** *Given a bias  $B$ , with an unknown target network  $C_T$  being representable by  $B$ , and a learned constraint set  $C_L$ , if *nil* is returned by Algorithm 2, then the system has converged on  $C_T[X]$ .*

**Proof.** When *nil* is returned by Algorithm 2, it means that  $\nexists e \in \text{sol}(C_L[Y] \wedge \bigvee_{c_i \in B[Y]} \neg c_i)$ , with  $Y = \bigcup_{c \in B} \text{var}(c)$ , i.e.,  $\nexists e \in \text{sol}(C_L[Y] \wedge \bigvee_{c_i \in B[Y]} \neg c_i)$ . In order to prove convergence over all of  $X$ , we must have  $\nexists e \in \text{sol}(C_L[X] \wedge \bigvee_{c_i \in B[X]} \neg c_i)$ . We will now show that when  $Y = \bigcup_{c \in B} \text{var}(c)$ , it means that

$$\nexists e \in \text{sol}(C_L[Y] \wedge \bigvee_{c_i \in B[Y]} \neg c_i) \implies \nexists e \in \text{sol}(C_L[X] \wedge \bigvee_{c_i \in B[X]} \neg c_i)$$

Assume that Algorithm 2 returns *nil*, i.e., that no assignment exists in a  $Y \subset X$  that is accepted by  $C_L[Y]$  and rejected by  $B[Y]$ . This means that all the constraints in  $B[Y]$  are proved to be implied by the constraints in  $C_L[Y]$ . Thus, the remaining constraints in  $B$ , that are not proved to be redundant, are the constraints  $c \in B \setminus B[Y]$ . When we know that  $Y = \bigcup_{c \in B} \text{var}(c)$  it means that  $B[Y] = B$ , so  $B \setminus B[Y] = \emptyset$ . As a result, in this case, all the constraints in  $B$  are proved to be implied. Hence, no assignment that is accepted by  $C_L$  and rejected by  $B$  exists in  $X$ . ◀

## 4 Bottom-up Constraint Acquisition

We start by observing that all current active CA algorithms always consider either the full set of variables  $X$ , or a large subset  $Y \subseteq X$ , in their top-level loop (lines 2-9 in Algorithm 1). This generally leads to complete or almost-complete queries getting generated (line 3 of

---

**Algorithm 3** Growing Acquisition.

**Input:**  $\Gamma, X, D, C_{in}$  ( $\Gamma$ : the language,  $X$ : the set of variables,  $D$ : the set of domains,  $C_{in}$ : an optional set of known constraints)

**Output:**  $C_L$  : a constraint network

```

1:  $C_L \leftarrow \emptyset$ 
2:  $Y \leftarrow \emptyset$ 
3: while  $|Y| \leq |X|$  do
4:    $x \leftarrow x \in (X \setminus Y)$ 
5:    $Y \leftarrow Y \cup \{x\}$ 
6:    $B \leftarrow \{c \mid \text{rel}(c) \in \Gamma \wedge \text{var}(c) \subseteq Y \wedge x \in \text{var}(c)\}$ 
7:    $C_L \leftarrow \text{Acq}(Y, D^Y, B, C_L \cup C_{in}[Y])$ 
8: return  $C_L$ 

```

---

Algorithm 1). However, larger queries are generally harder to answer than smaller queries [25]. Also, a large initial query leads to many additional queries getting posed in the scope-finding method on line 8. That is because the worst-case complexity of the best scope-finding methods, in terms of the number of queries required, is  $\Theta(\log(|Y|))$ , where  $Y \subseteq X$  is the set of variables considered [25].

Additionally, by directly considering the whole set of variables, the CA algorithm has to represent and operate on the entire set of candidate constraints (i.e., the bias  $B$ ) at once. The bias is used in many parts of the acquisition process. Hence, the memory requirements and the run time of the acquisition process increase significantly as the bias grows, either because the problems contain more variables or because the language  $\Gamma$  given to the system includes a larger number of relations. This means that, in practice, state-of-the-art active CA methods are only applicable to problems with not too many variables or problems for which the user already has relatively precise knowledge about what constraints the system should consider (which corresponds to the bias being small).

To improve on this, we propose a novel *meta*-algorithm named GROWACQ (Algorithm 3). The key idea is to call a CA algorithm on an increasingly large subset of the variables  $Y \subseteq X$ , each time using only a relevant unexplored subset of the bias. GROWACQ begins with  $Y = \emptyset$  (line 2) and gradually incorporates more variables (lines 3-5). Once a new variable  $x_i \in X$  has been added to  $Y$ , the new problem becomes to find the new  $C_T[Y]$ . However, as  $C_T[Y \setminus \{x_i\}]$  was already found in the previous iterations, the set of constraints to seek is actually  $C_T[Y] \setminus C_T[Y \setminus \{x_i\}]$ . To find  $C_T[Y] \setminus C_T[Y \setminus \{x_i\}]$ , any existing active CA algorithm can be used. We represent this with the function *Acq* (line 7). In every iteration, only a part of the bias  $B$  is needed, namely  $B[Y] \setminus B[Y \setminus \{x_i\}]$ , and as shown in Lemma 1, the bias constructed at line 6 is equivalent to  $B[Y] \setminus B[Y \setminus \{x_i\}]$ .

► **Lemma 1.** *Let  $Y_i$  be the set of variables  $Y$  in iteration  $i$  after line 5 of Algorithm 3 and  $B_i = \{c \mid \text{rel}(c) \in \Gamma \wedge \text{var}(c) \subseteq Y_i \wedge x_i \in \text{var}(c)\}$  be the bias  $B$  constructed at line 6 in iteration  $i$ . It holds that  $B_i = B[Y_i] \setminus B[Y_{i-1}]$ .*

**Proof.** At line 6 of Algorithm 3, the bias  $B$  is constructed. For each iteration  $i$ , it is constructed as  $B_i = \{c \mid \text{rel}(c) \in \Gamma \wedge \text{var}(c) \subseteq Y_i \wedge x_i \in \text{var}(c)\}$ . For a set of variables  $Y_i$ , the full bias, which includes all candidate constraints, is  $B[Y_i] = \{c \mid \text{rel}(c) \in \Gamma \wedge \text{var}(c) \subseteq Y_i\}$ . For the previous iteration, as  $Y_{i-1} = Y_i \setminus \{x_i\}$ , we know that  $B[Y_{i-1}] = \{c \mid \text{rel}(c) \in \Gamma \wedge \text{var}(c) \subseteq Y_i \setminus \{x_i\}\}$ . Thus, the additional constraints that are in  $B[Y_i]$  and not in  $B[Y_{i-1}]$

are the ones with a scope  $var(c) \subseteq Y_i$  for which  $x_i \in var(c)$ :

$$\begin{aligned} B[Y_i] \setminus B[Y_{i-1}] &= \{c \mid rel(c) \in \Gamma \wedge var(c) \subseteq Y_i\} \setminus \{c \mid rel(c) \in \Gamma \wedge var(c) \subseteq Y_i \setminus \{x_i\}\} \\ &= \{c \mid rel(c) \in \Gamma \wedge var(c) \subseteq Y_i \wedge x_i \in var(c)\} = B_i \end{aligned}$$

Hence, it holds that  $B_i = B[Y_i] \setminus B[Y_{i-1}]$  for  $B_i = \{c \mid rel(c) \in \Gamma \wedge var(c) \subseteq Y_i \wedge x_i \in var(c)\}$ . ◀

This bottom-up approach alleviates the problems described above, i.e., starting from large initial queries and having to represent the whole bias from the beginning, in two ways. First, it naturally leads to partial queries of increasing size in the first step of the “inner” CA system (Algorithm 1 line 5). This is valuable since smaller queries are generally easier for the user to answer [25], and also a smaller initial query leads to a lower worst-case number of additional queries to locate scopes. Second, since the algorithm only stores and uses a small part of the bias at a time (line 6 of Algorithm 3), it is able to handle significantly larger biases than the state-of-the-art. Not representing the whole bias in every iteration does not affect the algorithm’s correctness, as we state in Proposition 2.

► **Proposition 2.** *Given a bias  $B$  built from a language  $\Gamma$ , with bounded arity constraints, and a target network  $C_T$  representable by  $B$ , GROWACQ is correct (i.e., will learn a constraint set  $C_L$  that is equivalent to  $C_T$ ), as long as a correct (i.e., sound and complete) CA algorithm is used in line 7.*

**Proof.** (Sketch)

Let us now prove that if any correct algorithm is used in line 7 of Algorithm 3 – like QUACQ, MQUACQ or MQUACQ-2 – GROWACQ remains correct. We will subscript sets with the number of the iteration that they occur in to distinguish between the iterations. Even though the full bias  $B$  is never constructed and never kept in memory all at once in GROWACQ, we will still refer to it in this proof and denote it with  $B$ , i.e.,  $B = \{c \mid rel(c) \in \Gamma \wedge var(c) \subseteq X\}$ . When we instead write  $B_i$ , we refer to the part of the bias that is constructed and used in iteration  $i$  (line 6 of Algorithm 3), which is  $B[Y_i] \setminus B[Y_{i-1}]$  (Lemma 1).

*Soundness.* GROWACQ adds constraints to  $C_L$  only at line 7 of Algorithm 3. At that line, only constraints returned from the inner interactive CA algorithm are added to  $C_L$ . Since the assumption is that a sound algorithm is used in the *Acq* function, GROWACQ is sound.

*Completeness.* We prove that GROWACQ is complete by proving by induction that, after each iteration  $i$ ,  $C_L$  is equivalent to  $C_T[Y_i]$ , meaning that after the last iteration,  $C_L$  is equivalent to  $C_T[X]$ . GROWACQ starts with  $Y_1 = \emptyset$ , so both  $C_T[Y_1]$  and  $B_1$  are empty. The first iteration where the algorithm has to actually learn any constraints will be the one where  $Y$  grows large enough so that  $C_T[Y] \neq \emptyset$ . Assume that this happens at iteration  $k$ . In this case,  $C_T[Y_k]$  will be representable by  $B_k$ , because  $B_k = B[Y_k] \setminus B[Y_{k-1}]$  and we know that  $C_T[Y_{k-1}] = \emptyset$ . Since  $C_T[Y_k]$  is representable by  $B_k$ , it will be successfully learned in line 7, as long as a complete interactive CA algorithm is used.

Assuming now that  $C_L = C_T[Y_n]$  holds at the end of the  $n$ -th iteration, let us now prove that  $C_L = C_T[Y_{n+1}]$  will hold at the end of the  $n+1$ -th iteration. From the assumption that  $C_L = C_T[Y_n]$ , it follows that  $(B[Y_n] \setminus C_L) \cap C_T = \emptyset$ . As a result,  $B_{n+1}$ , being equal to  $B[Y_{n+1}] \setminus B[Y_n]$  does not exclude any constraint from  $C_T[Y_{n+1}]$  that has not already been learned. From this, it follows that  $(C_T[Y_{n+1}] \setminus C_L) \subseteq B_{n+1}$ , and thus this set of constraints will be learned in line 7 as long as a complete interactive CA algorithm is used. Hence, GROWACQ is complete. ◀

## 5 Guided query generation

We now turn our attention to the objective function used at line 9 of Algorithm 2. Since when GROWACQ is used, the size of  $B$  used in every iteration is reduced, query generation is now often fast, leaving sufficient room for using optimization to find a *good* query.

The objective function used in existing query generation systems [6, 25] tries to maximize the number of constraints from  $B$  that are violated by the generated query  $e$ . The motivation is that this can potentially help shrink the bias faster. The objective function is

$$e = \arg \max_e \sum_{c \in B} \llbracket e \notin \text{sol}(\{c\}) \rrbracket$$

where  $\llbracket \cdot \rrbracket$  is the Iverson bracket which converts *True/False* into 1/0.

However, looking only at the number of violated constraints in  $B$  does not fully capture what a good query is:

- We want queries that lead to a positive answer to violate many constraints from the bias  $B$ , as these can then all be removed from  $B$ , shrinking it faster.
- On the other hand, we want queries that lead to a negative answer to violate a small number of constraints from  $B$ , as it allows the CA system to find the conflicting constraint faster.

Based on this, in order to generate good queries regardless of the user’s answer, we want query generation to minimize the violation of constraints that are in the unknown target set  $C_T$ , seeking a query to which the user’s answer will be “yes”. At the same time, we want to maximize the violation of constraints in  $B$  that are not in  $C_T$ , so that positive answers can shrink the bias faster (the first bullet point above). Note that we also have the constraint ensuring that at least one constraint from  $B$  has to be violated. This means that when  $B \setminus C_T = \emptyset$ , we want a minimum number of constraints in  $C_T$  that we have not already learned to be violated. This leads to negative queries that violate a small number of constraints in  $B$  (the second bullet point above).

Assume we have access to an oracle  $\mathcal{O}$  that tells us whether a constraint  $c$  belongs to the unknown target set or not:  $\mathcal{O}(c) = (c \in C_T)$ . Using this oracle we can formulate an objective function for query generation, using the reasoning above, as follows:

$$\sum_{c \in B} \llbracket e \notin \text{sol}(\{c\}) \rrbracket \cdot (1 - |\Gamma| \cdot \llbracket \mathcal{O}(c) \rrbracket),$$

On the one hand, every time that the oracle returns *False* for a constraint from the bias that is violated by  $e$ , the objective function is increased by 1, thereby maximizing the violation of these constraints. Conversely, for constraints where  $\mathcal{O}$  returns *True*, we aim to minimize the violations, which requires a reduction in the objective value for each such violated constraint. However, it is possible that violating a set of constraints  $C$  (where  $\forall c_i \in C \mid \mathcal{O}(c_i) = \text{False}$ ) may imply the violation of a constraint  $c_j$  with  $\mathcal{O}(c_j) = \text{True}$ . In such cases, if the reduction in the objective value for violating  $c_j$  is not large enough, the system will violate both  $C$  and  $c_j$ , maximizing the objective. To address this issue, we introduce a “penalty” of  $|\Gamma|$ , which is equal to the upper bound of the number of constraints in each scope. This ensures that the system prioritizes satisfying a constraint with  $\mathcal{O}(c_j) = \text{True}$ , over violating other constraints from  $B$ .

**Modeling the oracle.** Observe how the current objective of maximizing violations corresponds to using a model of the oracle  $M$  that always answers *False*, i.e., that assumes that none of the candidate constraints belong to  $C_T$ . On the other hand, if we used an oracle  $M$

that always answers *True*, then the query generation would try to violate as few constraints as possible. However, the  $\bigvee_{c_i \in B[Y]} \neg c_i$  constraint would still need to be satisfied, in the extreme case leading every query to violate exactly one constraint from  $B$ . Based on this observation, we propose to model the oracle using the following model  $M$ , which tries to determine for every constraint  $c$  whether violating or satisfying  $c$  would lead to the least amount of queries later on in the algorithm.

$$M(c) = \left( \frac{1}{P[c \in C_T]} \leq \log(|Y|) \right)$$

On the one hand, in the extreme case, the constraints for which  $M(c)$  answers *True* will be violated one by one in the later queries (once most of the constraints for which  $M(c)$  answers *False* have been dealt with). Let  $P[c \in C_T]$  be a probabilistic estimate of whether  $c$  is part of  $C_T$ . Then, if the generated queries would violate the constraints with that probability one by one, we would in expectation need  $1/P[c \in C_T]$  queries to find a constraint from  $C_T$ . For example, for a set of constraints that each has a probability of 25%, 1 in every 4 queries is expected to lead to a  $c \in C_T$  being learned.

On the other hand, for each constraint  $c \in C_T$  for which  $M(c)$  answers *False*, a scope-finding procedure is needed to locate the violated constraint. The most efficient functions commonly used to do it (i.e., FindScope [6] or FindScope-2 [25]) have been shown to require  $\Theta(\log(|Y|))$  queries to find a violated constraint  $c \in C_T$  in the worst case, where  $Y$  is the number of variables considered in query generation. As a result, we estimate the number of queries needed in this case as  $k \cdot \log(|Y|)$ , with  $k$  a constant. We found  $k = 1$  to work well in practice.

**Probability estimation.** To compute the probability  $P(c \in C_T)$  of a constraint  $c \in B$ , we use a simple approach, considering only information from the relations  $rel(c)$  of the constraints. More specifically, to compute  $P(c \in C_T)$ , we count the number of times a constraint with relation  $rel(c)$  has been added to  $C_L$ , and divide it by the total number of times that such a constraint has been removed from  $B$ . Much more advanced estimation techniques, including machine learning methods, can be used for more accurate estimation. We leave this for future work.

## 6 Experimental evaluation

In this section, we empirically answer the following research questions:

- (Q1) Does using PQ-GEN with conventional solvers avoid premature convergence, and how do CA systems perform when they use it?
- (Q2) Does GROWACQ (using MQUACQ-2) perform better than using MQUACQ-2 directly?
- (Q3) How does our probability-guided query generation objective function perform compared to the one used in current CA systems?
- (Q4) How does the combination of our methods perform?
- (Q5) How do our methods perform on problems with a huge bias  $B$ ?

### 6.1 Benchmarks

We used the following benchmarks:

**Jigsaw Sudoku.** The Jigsaw Sudoku is a variant of Sudoku in which the  $3 \times 3$  boxes are replaced by irregular shapes. It consists of 81 variables with domains of size 9. The target network consists of 811 binary  $\neq$  constraints, on rows, columns, and shapes. The bias  $B$  was constructed using the language  $\Gamma = \{\geq, \leq, <, >, \neq, =\}$  and contains 19 440 binary constraints.

**Murder.** The Murder puzzle problem consists of 20 variables with domains of size 5. The target network contains 4 cliques of 10  $\neq$  constraints and 12 additional binary constraints.

The bias was initialized with 760 constraints based on the language  $\Gamma = \{\geq, \leq, <, >, \neq, =\}$ .

**Random.** We used a problem with 100 variables and domains of size 5. We generated a random target network with 495  $\neq$  constraints. The bias was initialized with 19 800 constraints, using the language  $\Gamma = \{\geq, \leq, <, >, \neq, =\}$ .

**Golomb rulers.** The problem is to find a ruler where the distance between any two marks is different from that between any other two marks. We built a simplified version of a Golomb ruler with 8 marks, with the target network consisting only of quaternary constraints.<sup>1</sup> The bias, consisting of 238 binary and quaternary constraints, was created with the language  $\Gamma = \{\geq, \leq, <, >, \neq, =, |x_i - x_j| \neq |x_k - x_l|\}$ .

**Job-shop scheduling.** The job-shop scheduling problem involves scheduling a number of jobs, consisting of several tasks, across a number of machines, over a certain time horizon. The decision variables are the start and end times of each task. There is a total order over each job's tasks, expressed by binary precedence constraints. There are also constraints capturing the duration of the tasks and that tasks should not overlap on the same machine. The language  $\Gamma = \{\geq, \leq, <, >, \neq, =, x_i + c = x_k\}$  was used, with  $c$  being a constant from 0 up to the maximal duration of the jobs. We used a problem instance containing 10 jobs, 3 machines (i.e.,  $|X| = 60$ ) and a time horizon of 15 steps, leading to a bias containing 14 160 constraints.

## 6.2 Experimental setup

Let us now give some details about the experimental settings:

- All the experiments were conducted on a system carrying an Intel(R) Core(TM) i9-11900H, 2.50 GHz clock speed, with 16 GB of RAM.
- We measure the total number of queries  $\#q$ , the average time of the query generation process  $\bar{T}_{gen}$  (line 3 of Algorithm 1), the average waiting time  $\bar{T}$  per query for the user, and the total time needed (to converge)  $T_{total}$ . All times are presented in seconds. The difference between  $\bar{T}_{gen}$  and  $\bar{T}$  is that the latter takes into account also the queries posed on lines 8-9 of Algorithm 1, which are very fast to compute.
- We evaluate our methods in comparison with the state-of-the-art method MQUACQ-2 [27].
- All methods and benchmarks were implemented in Python <sup>2</sup> using the CPMpy constraint programming and modeling library [16], except for the experiments using custom solvers.<sup>3</sup>
- The results presented in each benchmark, for each algorithm, are the means of 10 runs.

We now discuss the results of our experimental evaluation, based on the questions we posed at the beginning of the section.

## 6.3 [Q1] Performance of PQ-Gen

Both PQ-GEN, our projection-based query generation approach, and TQ-GEN [1] (discussed in Section 3.1) involve hyperparameters that affect their performance. Thus, we first performed a hyperparameter sensitivity analysis to assess their performance under different

<sup>1</sup> The ternary constraints derived when  $i = k$  or  $j = l$  in  $|x_i - x_j| \neq |x_k - x_l|$  were excluded, as also done in the literature [25, 27]

<sup>2</sup> Our code is available online in: <https://github.com/Dimosts/ActiveConLearn>

<sup>3</sup> For the custom solver based query generators from [6, 25], we obtained the implementations (in C++) through personal communication with the authors.

configurations. In tandem with TQ-GEN, we also used the adjust function described in [1]. We used the JSudoku benchmark for this comparison. For TQ-GEN, we fixed the hyperparameter  $\alpha$  to 0.8 as recommended in [1], and used  $\tau = \{0.05, 0.1, 0.2, 0.3\}$  and  $t = \{0.5, 1, 1.5, 2\}$ . For PQ-GEN hyperparameters, we used  $l = \{3000, 5000, 7500, 10000\}$  and  $t = \{0.5, 1, 1.5, 2\}$ . Thus, we examined 16 different configurations for each. A summary of the results are shown in Table 1.<sup>4</sup>

■ **Table 1** A summary of the performance of TQ-GEN and PQ-GEN with different configurations.

Problem	<i>Conv</i>	<i>#q</i>	<i>T<sub>max</sub></i>	<i>T<sub>total</sub></i>
MQuAcq-2 with TQ-GEN [1]	32%	7 555	20.66	2371.40
MQuAcq-2 with PQ-GEN (ours)	100%	6 551	4.42	728.25

Confirming our analysis, with our PQ-GEN there is never a case of premature convergence, no matter what hyperparameters are used. On the other hand, when TQ-GEN is used, the system fails to converge in the majority of cases, and specific hyperparameter values have to be chosen to ensure convergence. In addition, our PQ-GEN shows much better performance both in terms of the number of queries needed and, especially, in terms of runtime.

In more detail, we compared our projection-based query generation (PQ-GEN) with a baseline where we run a conventional CP solver to directly solve the query generation problem, using a one-hour time limit, as well as with query generation methods from the literature, i.e., TQ-GEN and the custom solver based query generators from [6, 27]. For PQ-GEN and TQ-GEN we used the best configuration found in the previous experiment. That is, we run PQ-GEN with  $l = 5000$  and  $t = 1$  and TQ-GEN with  $\tau = 0.2$  and  $t = 2$ . We used benchmarks that are similar to the ones used in [6, 27]. For consistency, we used the same state-of-the-art query generation objective function across all methods that accept one, i.e., our PQ-GEN and the custom solvers, which tries to maximize the number of violated constraints from  $B$ . The results are shown in Table 2.

We can observe that convergence was reached in all cases, except for the baseline in which a conventional solver was used directly using a time limit. Our method, PQ-GEN, and the baseline show similar performance in terms of the number of queries needed. While being much better than TQ-GEN in JSudoku and Random, where  $B$  is larger, especially when considering time performance.

On the other hand, when custom solvers are used, we can see that the time performance has improved and the number of queries has decreased. This happens because the custom solver can return a partial assignment of any size, trying only to maximize the value of the objective function used, and utilizing heuristics from the literature, while when our PQ-GEN is used, the query generated has to be a solution in a specific (sub)set of variables, which takes more time to compute. As a result, we observed that custom solvers often return queries that violate more constraints from  $B$ , which helps MQUACQ-2 shrink the bias faster in terms of the number of queries needed.

## 6.4 [Q2 - Q4] Evaluating GrowAcq and guided query generation

Hereafter, we continue our experiments using PQ-GEN, as the motivation was to investigate techniques that work with any solver. As using PQ-Gen allows us to use conventional solvers, able to run on any given benchmark, in contrast to the custom solvers from [6, 27], where specific constraint relations are implemented, from now on we will use all of the benchmarks mentioned in Section 6.1.

<sup>4</sup> More details regarding this experiment can be found in Appendix A.



■ **Table 2** Comparing PQ-GEN with state-of-the-art query generators.

Method	Problem	# $q$	$\bar{T}_{gen}$	$\bar{T}$	$T_{max}$	$T_{total}$	Convergence
Using conventional solvers							
MQUACQ-2 <sub>BASELINE</sub>	JSudoku	6 337	2.05	0.21	11.67	-	0%
	Murder	347	0.09	0.01	0.31	4.69	100%
	Random	5 694	2.90	0.05	20.78	294.94	100%
MQUACQ-2 <sub>TQ-GEN [1]</sub>	JSudoku	7 153	0.26	0.13	8.46	919.34	100%
	Murder	394	0.04	0.01	0.32	5.22	100%
	Random	5787	5.26	1.23	17.61	7100.44	100%
MQUACQ-2 <sub>PQ-GEN (OURS)</sub>	JSudoku	6 458	0.77	0.10	3.19	666.91	100%
	Murder	370	0.66	0.03	1.10	12.28	100%
	Random	5 708	0.60	0.04	2.22	233.62	100%
Using custom solvers							
MQUACQ-2 <sub>GenerateQuery.cutoff [6]</sub>	JSudoku	5 321	0.99	0.06	2.04	336.42	100%
	Murder	421	0.76	0.13	1.02	53.21	100%
	Random	5 349	0.94	0.04	3.24	198.65	100%
MQUACQ-2 <sub>max<sub>B</sub> [25]</sub>	JSudoku	5 042	1.00	0.06	2.34	277.36	100%
	Murder	325	0.86	0.04	1.01	12.90	100%
	Random	5 012	0.94	0.02	1.52	95.21	100%

**[Q2] Using GrowAcq within MQuAcq-2.** We now evaluate the performance of GROWACQ, our proposed bottom-up CA approach. To evaluate it, we used MQUACQ-2, as the inner CA algorithm within GROWACQ (line 7 of Algorithm 3) and compared this to using MQUACQ-2, directly on the full-sized problem. Table 3, top two blocks, presents the results.

We can observe that the usage of GROWACQ results in a reduction of the number of queries in JSudoku, Murder, and Random, while a slight increase can be seen in Golomb. In Job-shop, the increase in the number of queries is somewhat larger (25%). This is the case because the target constraint network in this benchmark is sparse, with most of the iterations of GROWACQ in a  $Y \subset X$  not learning any constraint from  $C_T$  and only shrinking the bias. So, when the full-sized problem is looked at directly when MQUACQ-2 is used, the bias  $B$  can shrink with fewer queries. On the other hand, when the target network is not sparse, there is a decrease in the number of queries of up to 19%, due to the fact that the system can locate the scopes of the constraints faster, starting from a  $Y \subset X$  every time. Based on the above observations, we can see that using GROWACQ leads to learning constraints in a lower amount of queries, but on the other hand, needs more queries to shrink the bias.

Finally, although the total time is almost the same in most problems, and slightly increased in JSudoku and Golomb, the average time per query has not noticeably increased, while the maximum time the user has to wait between two queries has decreased significantly (up to 88% in the Job-shop benchmark), due to the overall reduction in the time needed in query generation in almost all problems (as indicated by the  $\bar{T}_{qgen}$  column). As the (maximum) waiting time for the user is of paramount importance for interactive settings, we can see that GROWACQ improves this aspect of time performance of interactive CA systems.

**[Q3] Guided query generation.** In order to evaluate the performance of our proposed objective function for guiding query generation, we compare it with the use of the most popular objective function used in state-of-the-art CA systems, i.e., maximizing violations of constraints from  $B$ . The objective functions are utilized in line 9 of Algorithm 2. For this comparison, GROWACQ is used, again with MQUACQ-2 as the inner acquisition algorithm at

■ **Table 3** Evaluation of GROWACQ and the proposed approach for guiding query generation.

Problem	# $q$	$\bar{T}_{gen}$	$\bar{T}$	$T_{max}$	$T_{total}$
MQUACQ-2					
JSudoku	6 458	0.77	0.10	3.19	666.91
Murder	370	0.66	0.03	1.10	12.28
Random	5 708	0.60	0.04	2.22	233.62
Golomb	233	0.96	0.22	1.22	50.89
Job-shop	590	1.03	0.10	5.36	56.23
GROWACQ + MQUACQ-2					
Sudoku	5 863	0.15	0.12	1.98	721.15
Murder	357	0.04	0.02	0.11	7.97
Random	4 804	0.14	0.05	1.30	230.36
Golomb	270	0.80	0.29	1.30	78.47
Job-shop	786	0.13	0.06	0.66	48.18
GROWACQ + MQUACQ-2 <i>guided</i>					
JSudoku	3 963	0.15	0.24	1.96	963.42
Murder	250	0.04	0.04	0.27	9.07
Random	4 820	0.14	0.05	1.19	229.47
Golomb	100	0.16	0.27	0.95	27.44
Job-shop	776	0.13	0.06	0.64	47.53

line 7 of Algorithm 3. The results using the guided query generation can be seen in Table 3, bottom-two blocks, comparing GROWACQ + MQUACQ-2 against GROWACQ + MQUACQ-2 *guided*.

We can see that, when using our probability-based guidance for query generation, the number of queries has significantly decreased in JSudoku, Murder, and Golomb, while it has remained nearly the same in Random and Job-Shop. In the latter cases, the number of queries has not decreased because these are under-constrained problems, and thus the probability derived from the constraints' relations was small. This led to maximizing the violations of all constraints in  $B$  (i.e., the same behavior as with the existing objective). On the other hand, in the problems that do not have a sparse constraint network, where using the simple counting method to compute the probabilities of the constraints could effectively guide the acquisition system, the decrease observed in the number of queries is substantial (32% in JSudoku, 30% in Murder, and 64% in Golomb). However, as violating constraints one-by-one leads to more queries *generated* at line 3 of Algorithm 1, yet fewer queries at lines 8-9, which are very fast to compute, there is a small increase in the total time on JSudoku.

**[Q4] Combination of our methods.** Comparing the combination of our methods (i.e., GROWACQ + MQUACQ-2 *guided*) with MQUACQ-2 (Table 3), we can see that combining our bottom-up approach with guiding the query generation greatly outperforms MQUACQ-2 in terms of the number of queries needed to achieve convergence on most of the benchmarks. The number of queries has decreased on all benchmarks except Job-shop, where, because of its sparse target network, we need 23% more queries, as GROWACQ increases the number of queries to converge in underconstrained problems, due to the reasons described in section 6.4, while guiding the query generation does not improve it, as the probabilities estimated are always low. In the rest of the problems, we observe a total decrease of 16% in Random, 39% in JSudoku, 32% in Murder, and up to 60% in Golomb.

These results demonstrate the effectiveness of the proposed methods in reducing the number of queries needed for CA algorithms, which is crucial in interactive scenarios.

## 6.5 [Q5] Dealing with larger biases

To answer this question, we evaluated GROWACQ and the combination of our methods on larger instances of the Job-shop benchmark, using the same language as before. We used two instances: one with 15 jobs, 11 machines, and 40 steps (denoted as JS-15-11), which resulted in a bias consisting of 542 850 constraints, and one with 19 jobs, 12 machines, and again 40 steps (denoted as JS-19-12), resulting in a bias of 1 037 400 constraints. The results are presented in Table 4.

On the one hand, GROWACQ needs more queries to converge (like on the smaller Job-Shop instance) because the constraint network of this problem is sparse. Yet the total time needed to converge is one order of magnitude lower than in MQUACQ-2, being 24.4 times faster in the instance with a bias size of 0.5 million constraints and 25.6 times faster in the instance with  $|B| > 1M$ . In addition, the maximum waiting time has drastically decreased by using GROWACQ (and the combination GROWACQ and guiding query generation), from 5 499 seconds to only 3 (resp. 8) seconds in JS-15-11 and from more than 20 371 seconds to only 7 (resp. 6) seconds in JS-19-12. Importantly, the average waiting time is more than 30 times lower when using GROWACQ. Note that, as in the smaller job-shop instance, guiding does not lead to improvement in terms of the number of queries. However, it does not noticeably worsen the time performance of the system.

Hence, the experiments confirm that the proposed methodology can efficiently handle significantly larger sets of candidate constraints than the state of the art, up to 50 times larger than the ones commonly used in the literature [6, 8, 27, 29].

■ **Table 4** Experimental results on instances with a large bias.

Problem	$ B $	$\#q$	$\bar{T}_{qgen}$	$\bar{T}$	$T_{max}$	$T_{total}$
MQUACQ-2						
JS-15-11	$\approx 0.5M$	5 456	66.12	6.25	5 499.76	34 085.73
JS-19-12	$\approx 1M$	8 012	80.99	9.75	20 371.74	78 124.41
GROWACQ + MQUACQ-2						
JS-15-11	$\approx 0.5M$	7 015	0.44	0.20	2.84	1 422.93
JS-19-12	$\approx 1M$	10 309	0.62	0.29	6.92	2 984.77
GROWACQ + MQUACQ-2 <sub>guided</sub>						
JS-15-11	$\approx 0.5M$	7 062	0.44	0.20	7.88	1 399.63
JS-19-12	$\approx 1M$	10 219	0.64	0.30	6.19	3 054.68

## 7 Conclusions

Some of the most important limitations of interactive CA methods are the large number of queries needed to converge, as well as the size of the candidate constraint set that they can handle efficiently. In this work, we presented novel methods to alleviate these issues, improving the efficiency of CA systems. We proposed a bottom-up approach, which allows the system to handle significantly larger biases, reducing the maximum waiting time for the user, and also reducing the total number of queries needed when the target constraint network is not sparse. We also introduced a probabilistic method to guide query generation,

further reducing the number of posted queries when our simple counting method could guide the acquisition system to learn constraints more efficiently. In addition, we presented a new query generation technique, named PQ-GEN, that allows the use of conventional CP solvers, removing the dependency of existing methods on customized solvers to converge. Our experimental evaluation showed that our proposed methods outperform state-of-the-art systems in terms of the number of queries in problems with non-sparse constraint networks, reducing this number up to 60%. In addition, the experiments show that GROWACQ can handle up to 50 times larger biases than the ones commonly used in the literature, allowing CA to tackle increasingly large and complex problems. The biggest avenue for future work is to further investigate additional ways to reduce the number of queries needed, e.g., by using guidance in all parts of the acquisition process (not just the query generation), and with more advanced probabilistic models. Another important avenue is to consider the setting in which user answers can be noisy as has been investigated for passive systems.

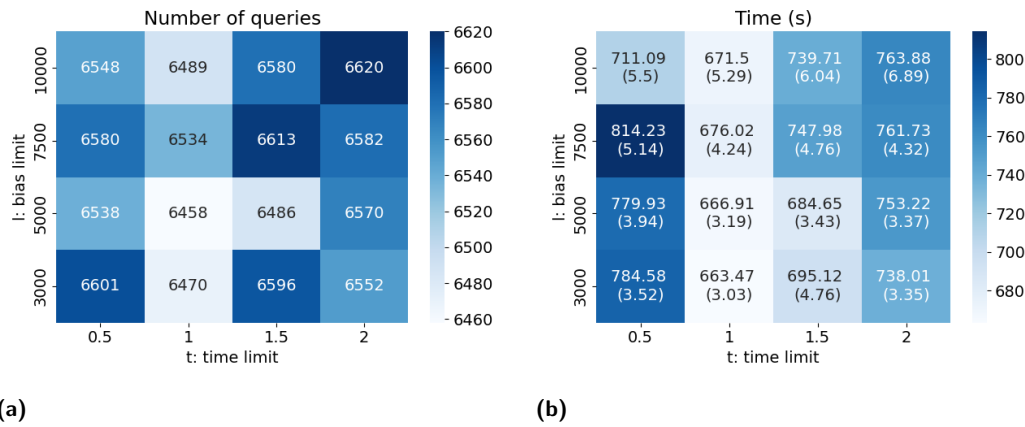
---

### References

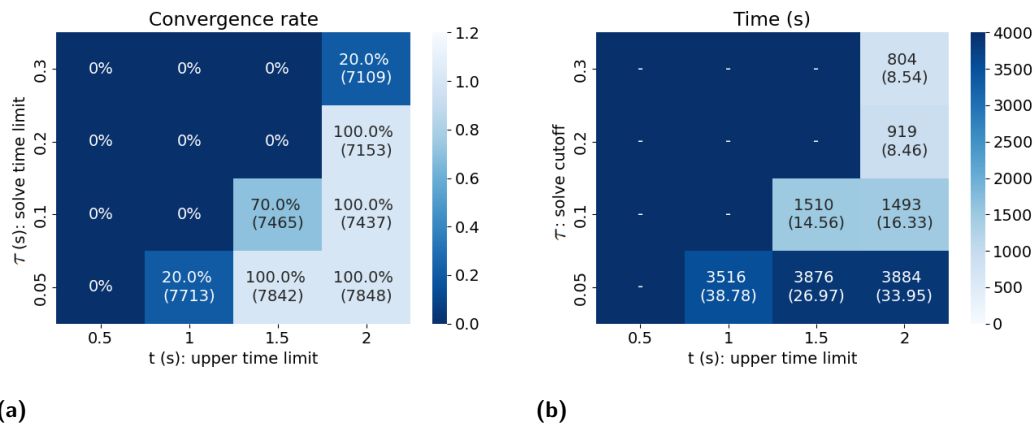
---

- 1 Hajar Ait Addi, Christian Bessiere, Redouane Ezzahir, and Nadjib Lazaar. Time-bounded query generator for constraint acquisition. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 1–17. Springer, 2018.
- 2 Dana Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.
- 3 Robin Arcangoli, Christian Bessiere, and Nadjib Lazaar. Multiple constraint acquisition. In *IJCAI: International Joint Conference on Artificial Intelligence*, pages 698–704, 2016.
- 4 Nicolas Beldiceanu and Helmut Simonis. A model seeker: Extracting global constraint models from positive examples. In *Principles and practice of constraint programming*, pages 141–157. Springer, 2012.
- 5 Senne Berden, Mohit Kumar, Samuel Kolb, and Tias Guns. Learning max-sat models from examples using genetic algorithms and knowledge compilation. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*, 2022.
- 6 Christian Bessiere, Clement Carbonnel, Anton Dries, Emmanuel Hebrard, George Katsirelos, Nina Narodytska, Claude-Guy Quimper, Kostas Stergiou, Dimosthenis C Tsouros, and Toby Walsh. Learning constraints through partial queries. *Artificial Intelligence*, 319:103896, 2023.
- 7 Christian Bessiere, Remi Coletta, Eugene C Freuder, and Barry O’Sullivan. Leveraging the learning power of examples in automated constraint acquisition. In *International Conference on Principles and Practice of Constraint Programming*, pages 123–137. Springer, 2004.
- 8 Christian Bessiere, Remi Coletta, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, Toby Walsh, et al. Constraint acquisition via partial queries. In *IJCAI*, volume 13, pages 475–481, 2013.
- 9 Christian Bessiere, Remi Coletta, Frédéric Koriche, and Barry O’Sullivan. A sat-based version space algorithm for acquiring constraint satisfaction problems. In *European Conference on Machine Learning*, pages 23–34. Springer, 2005.
- 10 Christian Bessiere, Remi Coletta, Barry O’Sullivan, Mathias Paulin, et al. Query-driven constraint acquisition. In *IJCAI*, volume 7, pages 50–55, 2007.
- 11 Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O’Sullivan. Constraint acquisition. *Artificial Intelligence*, 244:315–342, 2017.
- 12 Eugene C Freuder. Modeling: the final frontier. In *The First International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP)*, London, pages 15–21, 1999.
- 13 Eugene C Freuder. Progress towards the holy grail. *Constraints*, 23(2):158–171, 2018.
- 14 Eugene C Freuder and Barry O’Sullivan. Grand challenges for constraint programming. *Constraints*, 19(2):150–162, 2014.

- 15 Eugene C Freuder and Richard J Wallace. Suggestion strategies for constraint-based match-maker agents. In *International Conference on Principles and Practice of Constraint Programming*, pages 192–204. Springer, 1998.
- 16 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.
- 17 Mohit Kumar et al. Acquiring integer programs from data. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. IJCAI, 2019.
- 18 Mohit Kumar, Samuel Kolb, and Tias Guns. Learning constraint programming models from data using generate-and-aggregate. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.
- 19 Arnaud Lallouet, Matthieu Lopez, Lionel Martin, and Christel Vrain. On learning constraint problems. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, volume 1, pages 45–52. IEEE, 2010.
- 20 Nadjib Lazaar. Parallel constraint acquisition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3860–3867, 2021.
- 21 Michele Lombardi, Michela Milano, and Andrea Bartolini. Empirical decision model learning. *Artificial Intelligence*, 244:343–367, 2017.
- 22 Steven D Prestwich. Robust constraint acquisition by sequential analysis. *Frontiers in Artificial Intelligence and Applications*, 325:355–362, 2020.
- 23 Steven D Prestwich, Eugene C Freuder, Barry O’Sullivan, and David Browne. Classifier-based constraint acquisition. *Annals of Mathematics and Artificial Intelligence*, pages 1–20, 2021.
- 24 Kostyantyn Shchekotykhin and Gerhard Friedrich. Argumentation based constraint acquisition. In *Data Mining, 2009. ICDM’09. Ninth IEEE International Conference on*, pages 476–482. IEEE, 2009.
- 25 Dimosthenis C Tsouros and Kostas Stergiou. Efficient multiple constraint acquisition. *Constraints*, 25(3):180–225, 2020.
- 26 Dimosthenis C Tsouros and Kostas Stergiou. Learning max-csps via active constraint acquisition. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- 27 Dimosthenis C Tsouros, Kostas Stergiou, and Christian Bessiere. Structure-driven multiple constraint acquisition. In *International Conference on Principles and Practice of Constraint Programming*, pages 709–725. Springer, 2019.
- 28 Dimosthenis C Tsouros, Kostas Stergiou, and Christian Bessiere. Omissions in constraint acquisition. In *International Conference on Principles and Practice of Constraint Programming*, pages 935–951. Springer, 2020.
- 29 Dimosthenis C. Tsouros, Kostas Stergiou, and Panagiotis G. Sarigiannidis. Efficient methods for constraint acquisition. In *24th International Conference on Principles and Practice of Constraint Programming*, 2018.

**A** Hyperparameter evaluation for PQ-Gen and TQ-Gen

**Figure 1** Performance of PQ-GEN with different hyperparameters values, in terms of: a) the number of queries posted and b) the time (s) needed (in brackets we show the maximum waiting time for the user).



**Figure 2** Performance of TQ-GEN with different parameters, in terms of: a) the convergence rate (in brackets we show the number of queries posted when it converged) and b) the time (s) needed (in brackets we show the maximum waiting time for the user).

Both PQ-GEN, our projection-based query generation approach, and TQ-GEN [1] (discussed in Section 3.1) involve hyperparameters that affect their performance. As mentioned in Section 6.3, we performed a sensitivity analysis of the performance with respect to the hyperparameter configuration used of PQ-GEN and TQ-GEN [1]. In this comparison, both query generation methods were used within the state-of-the-art active CA method MQUACQ-2. We used the JSudoku benchmark for this comparison, as from the benchmarks considered in this paper, this is shown to be the hardest one to reach convergence on (see Table 2).

In more detail, we varied the hyperparameters of both PQ-GEN and TQ-GEN to assess their performance under different configurations. While we fixed the hyperparameter  $\alpha$  of [1] to 0.8 as recommended in [1], we had to try different values for the time-related hyperparameters,  $\tau$  and  $t$ . We did not use the values proposed by the authors in [1] because we use a different solver and system, and this can affect significantly the time performance.


## 36:20 Guided Bottom-Up Interactive Constraint Acquisition

In our evaluation we used  $\tau = [0.05s, 0.1s, 0.2s, 0.3s]$  and  $t = [0.5s, 1s, 1.5s, 2s]$  for TQ-GEN. We also used the adjust function described in [1], as it has been shown to improve its performance. For PQ-GEN hyperparameters, we used  $l = \{3000, 5000, 7500, 10000\}$  and  $t = \{0.5, 1, 1.5, 2\}$ . Thus, we examined 16 different configurations for each. The results of our experiments are presented in Figures 1 and 2, respectively, for PQ-GEN and TQ-GEN.


Focusing on Figure 1, we can see that the performance of PQ-GEN is stable across all configurations, both in terms of the number of queries and time performance, having also converged in all cases. Let us now shift our focus to Figure 2 and the performance of TQ-GEN. The first observation is that in the majority of the cases, MQUACQ-2 failed to converge when using TQ-GEN as the query generator. Only when the time limit was set to 2s, we see at least one run achieving convergence for all values of  $\tau$ . In addition, the performance of MQUACQ-2 using TQ-GEN is highly sensitive to changes in hyperparameter values, particularly with respect to time.

Overall, comparing the results of PQ-GEN and TQ-GEN, we observe that PQ-GEN exhibits superior performance in terms of convergence rate, fully overcoming the issue of premature convergence. PQ-GEN also requires a lower number of queries to reach convergence and offers improved time performance, resulting in reduced waiting times for the user.


# Addressing Problem Drift in UNHCR Fund Allocation

Sameela Suharshani Wijesundara ✉ 

Department of Data Science and AI, Faculty of IT, Monash University, Clayton, Australia  
ARC Industrial Training and Transformation Centre OPTIMA, Clayton, Australia

Maria Garcia de la Banda ✉ 

Department of Data Science and AI, Faculty of IT, Monash University, Clayton, Australia  
ARC Industrial Training and Transformation Centre OPTIMA, Clayton, Australia

Guido Tack ✉ 

Department of Data Science and AI, Faculty of IT, Monash University, Clayton, Australia  
ARC Industrial Training and Transformation Centre OPTIMA, Clayton, Australia

---

## Abstract

Optimisation models are concise mathematical representations of real-world problems, usually developed by modelling experts in consultation with domain experts. Typically, domain experts are only indirectly involved in the problem modelling process, providing information and feedback, and thus perceive the deployed model as a black box. Unfortunately, real-world problems “drift” over time, where changes in the input data parameters and/or requirements cause the developed model to fail. This requires modelling experts to revisit and update deployed models. This paper identifies the issue of problem drift in optimisation problems using as case study a model we developed for the United Nations High Commissioner for Refugees (UNHCR) to help them allocate funds to different crises. We describe the initial model and the challenges due to problem drift that occurred over the following years. We then use this case study to explore techniques for mitigating problem drift by including domain experts in the modelling process via techniques such as domain specific languages.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming; Software and its engineering → Constraint and logic languages

**Keywords and phrases** Fund Allocation, Problem Drift, Domain Specific Languages, MiniZinc

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.37

## 1 Introduction

Most software engineering projects suffer from one or several forms of *drift*, where a change in requirements during the lifetime of the project causes an existing, deployed software to degrade, to fail, or in the worst case, to produce incorrect solutions. Software projects that include optimisation components are no exception. In this paper, we present a real-world problem as a case study in *problem drift*, and discuss how we can mitigate its effects by enabling the problem owners to become part of the modelling and maintenance workflow.

Our case study is the United Nations High Commissioner for Refugees (UNHCR) fund allocation. UNHCR works in 135 countries and territories to provide crucial assistance to the millions of people forced to flee as a result of conflicts, crises, persecution or natural disasters; recently estimated as 112.6 million [19]. To achieve this, every year UNHCR proposes an annual *budget* for each of the many *projects* it wants to tackle and releases a Global Appeal requesting donations to cover it. For 2023 alone the Global Appeal was more than US\$10 billion [18]. The large amount of donations, or *funding*, received as a result need to be distributed among the budgeted projects according to different priorities, while ensuring the allocation respects any constraints set by the donors. For example, it is common for funding to be “earmarked” for a specific crisis, or a geographical area such as a region or a country. These allocation constraints often make it difficult for the funds to be optimally allocated.



© Sameela Suharshani Wijesundara, Maria Garcia de la Banda, and Guido Tack;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 37; pp. 37:1–37:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



As a result, excess funds would be carried forward from one year to the next rather than being directed to UNHCR’s activities such as life-saving assistance. The aim of UNHCR is to maximise the funds available for activities.

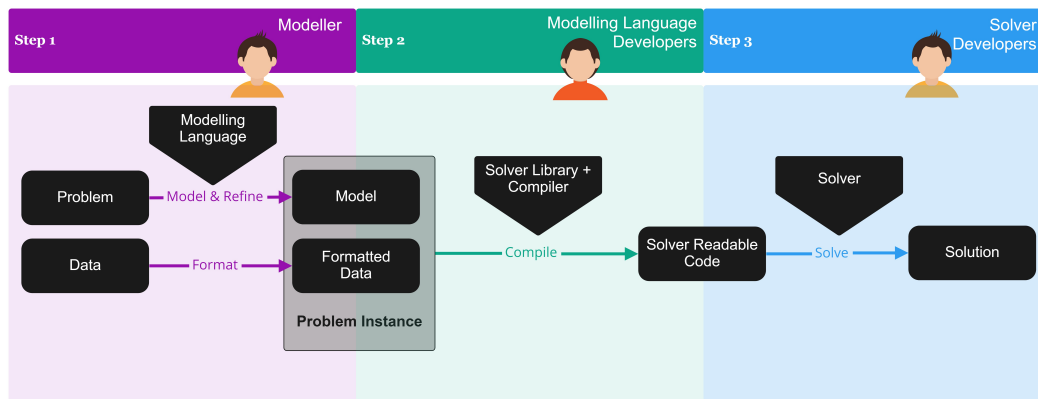
This *fund allocation problem* can be represented as a combinatorial optimisation problem. As such, it can be modelled in a high-level constraint modelling language (e.g., [5, 7, 14, 22]) and solved with state-of-the-art optimisation solving technology (e.g., [1, 17, 23]). In 2016, we developed a model for this problem that would have been able to provide US\$400 million more in allocations than their previous method, leaving only US\$100 million in unallocated funds. As it is common when developing such models, UNHCR experts were only indirectly involved in the modelling process: they provided information and feedback but did not fully understand the implemented model as they lacked the required expertise. This is unfortunate since, as it often happens with real-world problems, the problem requirements *drifted* over time, experiencing changes in the format and kind of the input data, as well as in the rules for allocating funds. After a while, the drift grew to the point where the developed model became unusable. In addition, personnel changes led to a loss of organisational memory, severing the contact between the organisation and us, the developers. As a result, UNHCR never implemented a full system using the model and continued to use their original greedy allocation algorithm, which they were able to maintain and update using in-house expertise.

This paper makes **three contributions**. The first is identifying the issue of problem drift in optimisation models and highlighting its importance to the community as an area worth investigating. The second contribution is our modelling of the UNHCR fund allocation problem: an initial model developed in the first phase of our work in 2016, and a new model developed in a second phase to incorporate the changes that caused the problem to drift. Our third and final contribution results from an analysis of this problem drift, and a subsequent exploration of techniques to mitigate this issue. We present an approach to include domain experts in the modelling process by adapting known software engineering techniques such as domain specific languages. This phase of our work is, so far, exploratory. However, we believe our approach can increase the domain experts’ trust in the developed solution, by making them an integral part of the development process, and can also enable the domain experts to adapt the model over time to keep up with problem drift.

## 2 Background

Combinatorial optimisation problems require finding a combination of choices (i.e., a **solution**) that satisfies a set of requirements and optimises the quality of the solutions. Modern approaches for solving these problems first specify a *model* of the problem that formally describes its choices, in terms of **variables** representing the decisions and their **domains** representing the choices available; its requirements, in terms of **constraints** defined over the variables; and the quality of the solutions in terms of an **objective function** also defined over the variables. Models are often specified using **parameters** so that input data can be provided independently of the model. This allows models to be used to solve any **instance** of the problem by instantiating the model parameters with the concrete input data and translating these instances into input for a **solver**, which then produces solutions to the problem. At a high level, this modelling and solving approach can be depicted as the step-wise process shown in Figure 1.

- *Step 1* formalises the real world problem via a model. This is currently a manual process where modelling experts communicate with domain experts to capture all relevant requirements, often requiring several iterations to progressively improve the formalisation (**refinement process**). The result is a model of the problem requirements written in a high-level modelling language such as MiniZinc [14], Essence [7], OPL [22] or AMPL [5].



■ **Figure 1** Step-wise illustration of the combinatorial problem modelling and solving process.

- *Step 2* takes the model along with concrete input data formatted according to the model's parameters and representing an instance of the problem, and compiles this instance into a format suitable for the solver selected by the modeller. Each of the modelling languages mentioned in *Step 1* has its own compilation methods and lower level languages such as FlatZinc for MiniZinc [14] and Essence' for Essence [7].
- *Step 3* solves the compiled instance using the selected solver. Solvers use state-of-the-art algorithms and heuristics that are efficient in tackling the unique challenges inherent in the kind of constraints they support. There are many different solving technology paradigms including MIP [23] (for Mixed Integer Programming), CP [17] (for Constraint Programming), and SAT [1] (for Boolean Satisfiability Problems).

*Steps 2* and *3* are extensively automated thanks to many years of research on modelling language and solver design. However, *Step 1* is a manual process and, in practice, considerably more complex than depicted in Figure 1, involving multiple refinement and feedback loops.

### 3 Phase 1: The 2016 UNHCR Fund Allocation Problem and Model

#### 3.1 Problem Description

In 2016 we worked with UNHCR on an optimisation model for their fund allocation. According to the rules at the time, part of or all of the donations pledged by an organisation (a fund) could be either *earmarked* or not, indicating whether the donor put some restrictions on the kind of projects the fund could cover or not, respectively. The kind of restrictions supported by the algorithm used at the time by UNHCR were based on the following fund and project attributes:

- **Level:** can take one of three values: **Region**, **Sub-region**, or **Country**, indicating the geographic scale of a project, e.g., a continent such as Europe, a part of a given continent such as Eastern Europe, or a country such as Ukraine. For a fund, the level is used to restrict its allocation, and can have an additional value **Global** to indicate that the fund is not restricted to a geographical area.
- **Region:** the region of the world where the project is located or to which the fund is allocated. Only relevant when the level is **Region**; disregarded otherwise.
- **Sub-region:** the sub-region of the world where the project is located or to which the fund is allocated. Only relevant when the level is **Sub-region**; disregarded otherwise.

## 37:4 Addressing Problem Drift in UNHCR Fund Allocation

- Area of Budgetary Control (ABC): the name of the UNHCR office which could be responsible either for a single country or a collection of countries each with fewer activities.
- Pillar: the broad target area of support. Pillar 1 stands for refugees, Pillar 2 for stateless people, Pillar 3 for reintegration, and Pillar 4 for internally displaced people. Pillars can be combined, e.g., Pillars 1-2. A fund that can be spent in any pillar is given the value `All Pillars`.
- Situation: the particular issue a project is tackling or a fund can be spent on, e.g., `South Sudan Situation`. A fund that can be spent on any situation is given the value `Country/Regular Program`.

The input data is given by an Excel sheet where each row is referred to as an `item`. The first six columns correspond to an item's pillar, situation, level, region, sub-region, and ABC. The next two columns contain the *budget* and *income*. An item with an empty budget column is a fund, and with a non-empty budget column is a project. Note that the budget of some projects can already be partially covered by money from other sources (e.g., bank interest). In such cases both the budget and income columns will have non-zero amounts.

The objective is to move money from fund items to project items to minimise the total amount of money left unspent in the funds, while ensuring the allocation constraints are satisfied. This means, ensuring money does **not** move from item A to item B if they have:

1. Different pillars, unless the pillar of item A is the generic `All Pillars`.
2. Different regions if the level of item A is `Region`.
3. Different sub-regions if the level of item A is `Sub-region`.
4. Different ABCs if the level of item A is `Country`.
5. Different situations, unless the situation of item A is `Country/Regular Programme`, its level is `Country`, its pillar is `All Pillars` and item A is in surplus for its country.

The first four preclude a fund earmarked for a non-generic pillar, region, sub-region, or country to be spent on a different one. The last one ensures that funds for generic country situations are spent first on their country. There are also common-sense rules ensuring, for example, that funds are not overspent and projects do not get more income than budgeted.

### 3.2 Model

Modelling this problem in MiniZinc was easy. We will not reproduce the full model here, but we will give enough details for later discussions.

**Parameters.** The Excel input was fed into 9 MiniZinc parameters: integer `n` representing the number of rows and used to build set `ITEM` of `1..n`, and eight one-dimensional arrays indexed by `ITEM` representing the values of the eight columns for each `ITEM`, i.e., its pillar, situation, level, region, sub-region, ABC, budget and income. The following MiniZinc code shows the definition of some of these parameters together with two enumerated data types (`enums`) whose definitions (strings from the input data) are the values for `REGION` and `SUBREGION`, an auxiliary parameter `maxinc` set to the maximum income, and an array `excess`, where `excess[i]` is the initial amount of money either available in fund `i`, or needed by project `i`:

```
set of int: ITEM = 1..n;
enum REGION;
enum SUBREGION;
array[ITEM] of REGION: region;
array[ITEM] of SUBREGION: subregion;
array[ITEM] of int: income;
array[ITEM] of int: budget;
int: maxinc = max(income);
array[ITEM] of int: excess = [ income[i] - budget[i] | i in ITEM ];
```

While enums were not used in the model, they are more intuitive and used here for brevity.

**Variables.** The decision variables are stored in a two-dimensional array `movement`, where `movement[i,j]` represents the amount of money moved from item `i` to item `j`, and is constrained to be between 0 and `maxinc`. Note that if `i` is a project, the value of `movement[i,j]` will be later set to 0. This array is defined as follows:

```
array[ITEM,ITEM] of var 0..maxinc: movement;
```

In addition, several auxiliary variables are defined based on `movement`. For example, `out[i]` and `inn[i]` represent, respectively, the amount of money moved out of item `i` or into it. They are defined as follows:

```
array[ITEM] of var 0..maxinc: out =
  [ sum(j in ITEM)(movement[i,j]) | i in ITEM ];
array[ITEM] of var 0..maxinc: inn =
  [ sum(j in ITEM)(movement[j,i]) | i in ITEM ];
```

**Constraints.** Are quite simple and modelled using the above parameters and variables. For example, the common-sense constraint ensuring funds are not overspent is modelled as:

```
constraint forall(i in ITEM where excess[i] > 0)(out[i] <= excess[i]);
constraint forall(i in ITEM where excess[i] <= 0)
  (forall(j in ITEM)(movement[i,j] = 0);
```

ensuring that if item `i` is a fund (`excess[i]>0`), the amount of money that moves out of `i` is not higher than its excess, and otherwise no money moves out of `i`. Similarly, constraint 2 which correctly earmarks funds based on region is modelled as follows:

```
constraint forall(i,j in ITEM
  where region[i] != region[j] /\ level[i] = Region)
  (movement[i,j] = 0);
```

ensuring no money moves from `i` to `j` if level of `i` is `Region` and their regions do not match. Note that constraints 3 and 4 follow exactly the same structure.

Our last example is constraint 5; the most complex. First we define auxiliary parameters `surplus_to_country_regular_situation[i]` as either a surplus if `i` is a generic country fund (i.e., one whose pillar is `all_pillars`, level `Country` and situation `country_regular_program`) or 0, otherwise. The surplus is defined as the fund's income minus the money needed by that country. We also define Boolean parameter `has_surplus_to_country_regular_situation[i]` to true iff the associated surplus parameter is positive.

```
array[ITEM] of -maxinc..maxinc: surplus_to_country_regular_situation =
  [ if situation[i] != country_regular_programme
    \ / level[i] != Country
    \ / pillar[i] != all_pillars
  then 0
  else income[i] -
    sum(j in ITEM where abc[i] = abc[j] /\
      situation[j] = country_regular_programme)
    (budget[j] - income[j])
  endif
  | i in ITEM ];
array[ITEM] of bool: has_surplus_to_country_regular_situation =
  [ surplus_to_country_regular_situation[i] > 0 | i in ITEM ];
```

These auxiliary parameters are then used to model part of constraint 5 as follows:

## 37:6 Addressing Problem Drift in UNHCR Fund Allocation

```
constraint forall(i in ITEM
    where situation[i] = country_regular_programme
    /\ level[i] = Country
    /\ pillar[i] = all_pillars
    /\ has_surplus_to_country_regular_situation[i])
    (sum(j in ITEM where situation[i] != situation[j]) (movement[i,j])
    <= surplus_to_country_regular_situation[i]);
```

If there is no surplus, the other part (not shown) sets `movement[i,j]=0` for every `j` in the same country but different situation.

**Objective.** Minimises the excess of funds and deficit of projects after allocation by means of the intermediate variable `difference`, which captures the excess for fund items (positive number) and the deficit for projects (negative number). The sum `obj` of the absolute value `absdiff` of these values is then computed and minimised as follows.

```
array[ITEM] of var -maxinc .. maxinc: difference =
    [ excess[i] - out[i] + inn[i] | i in ITEM ];
array[ITEM] of var 0..maxinc: absdiff;

constraint forall(i in ITEM)(absdiff[i] >= difference[i] /\
    absdiff[i] >= -difference[i]);

var int: obj = sum(i in ITEM)(absdiff[i]);
solve minimize obj;
```

This model solves the above fund allocation problem in less than 10s for the sample data we were given. With the 2016 sample data, the model was able to allocate US\$400 million more than the UNHCR greedy algorithm used at the time. UNHCR domain experts saw great value in the results produced by the optimisation model, and used it occasionally to provide supplementary input into their decision making process.

## 4 Phase 2: The 2022 UNHCR Fund Allocation Problem and Model

### 4.1 Problem Changes

When we contacted UNHCR again several years later, we learned that problem drift had made our model obsolete. After a number of discussions that spanned several months, we identified the following main changes to the problem.

**Data format.** Instead of the data being provided as a table where each row was an item representing both funds and projects, it is now provided as two separate tables (that we will refer to as *entities*), one for funds and one for projects.

**Attributes.** Some attributes changed names (e.g., fund income was now called *Balance*), others disappeared (e.g., funds no longer had budget as attribute, since they were always empty), and new ones appeared. The complete list of attributes for funds is: *Unique ID, Level, Domain, Region, Sub-Region, Country, Cost Center, Pillar, Situation, Account Status, Account Type, Balance*. Projects have the same attributes except that instead of *Balance*, they have a *Requirement* attribute that is equivalent to previously subtracting income from budget. Also, while previously the smallest geographical operational area provided in the sample data was the value of attribute *ABC*, countries are now further divided into smaller geographical areas defined by the attribute *Cost Centres*. More importantly, funds and

projects are now given priority levels via two different attributes. One is *Account Type*, which can take values OL (*operational level*) and AOL (*above operational level*), the latter having lesser priority [20]. The other is *Domain* which can take several values such as, in decreasing priority order, Headquarters (for leadership, management, policy guidance, etc), Global (for global programmes undertaken at the headquarters but of direct benefit to field operations), and Field (for field operations, which still takes 88% of the funding according to the 2022 Global Report [19]).

**Constraints.** Are now defined in terms of these new attributes. In particular, the constraints for Cost Centre and Domain are similar to those for other attributes.

**Objective.** While the main objective remained unchanged, it now also prioritised the allocation of funds based on their Account Type and Domain, as well as funds carried over from previous years and those with the highest number of constraints.

## 4.2 Model Changes

**Parameters.** Changes to the parameters of the model were mainly caused by UNHCR's decision to split ITEM into two entities, FUNDS and PROJECTS, and change their associated attributes. We modelled this similarly to before but using a different set of indexed arrays per entity, as follows:

```
set of int: PROJECTS = 1..numProjects;
set of int: FUNDS = 1..numFunds;

array[PROJECTS] of LEVEL: p_level;
array[PROJECTS] of DOMAIN: p_domain;
array[PROJECTS] of REGION: p_region;
array[PROJECTS] of ACCOUNT_TYPE: p_account_type;
array[PROJECTS] of int: p_requirement ;

array[FUNDS] of LEVEL: f_level;
array[FUNDS] of DOMAIN: f_domain;
array[FUNDS] of REGION: f_region;
array[FUNDS] of ACCOUNT_TYPE: f_account_type;
array[FUNDS] of int: f_balance;
```

**Variables.** The main decision variables are almost identical to those in the first model, except for the index sets now being known to be funds for the first dimension and projects for the second, and the maximum value being represented by fund parameter `f_balance`.

```
array[FUNDS,PROJECTS] of var 0..max(f_balance) : movement;
```

Most auxiliary variables also undergo very superficial changes, such as `out[i]` and `iin[i]`:

```
array[FUNDS] of var int : out =
  [ sum(p in PROJECTS)(movement[f,p]) | f in FUNDS ];
array[PROJECTS] of int : iin =
  [ sum(f in FUNDS)(movement[f,p]) | p in PROJECTS ];
```

**Constraints.** Some constraints encoding the same rules only require relatively superficial changes. For example, the constraint that earmarks funds based on region looks very similar:

## 37:8 Addressing Problem Drift in UNHCR Fund Allocation

```
constraint forall(f in FUNDS, p in PROJECTS
  where f_region[f] != p_region[p] /\ f_level[f] == L3)
  (movement[f,p] = 0);
```

if one knows L3 is equivalent to the old `Region`. Others change more, e.g., ensuring funds are not overspent is much simpler now that funds and projects are clearly separated:

```
constraint forall(f in FUNDS)(out[f] <= f_balance[f]);
```

In addition, we need new constraints to correctly earmark funds based on new attributes, such as cost centre, which look very similar to those implemented for Phase 1 attributes:

```
constraint forall(f in FUNDS, p in PROJECTS
  where f_cost_centre[f] != p_cost_centre[p] /\ f_level[f] == L6)
  (movement[f,p] = 0);
```

**Objective.** As described above, the old objective maximising the overall fund allocation was now extended to prioritise carry-over funds from the previous round, tightly earmarked funds (funds with the tightest restrictions), and funds allocated according to their Domain and Account Type. The new objective function was implemented as a weighted sum of these different aspects in order to achieve the required prioritisation.

### 4.3 Implementation and Adoption

We implemented the changes to the model as described above over the course of several months, in consultation with the UNHCR domain expert. Compared to the in-house algorithm, the new model is faster (it solves the sample data in under 3 minutes) and more effective in terms of total allocation and priority order. We were also able to experiment with extensions of the model, such as achieving fairness and balance in the allocation, which is easier to do using optimisation technology compared to greedy algorithms.

Unfortunately, the new model was not adopted by UNHCR. However, as a result of our work with them, they proposed the use of optimisation techniques with the new re-implementation of their enterprise resource planning (ERP) system. The viability of this is currently under investigation.

## 5 Phase 3: Addressing Problem Drift via Interactive Modelling DSLs

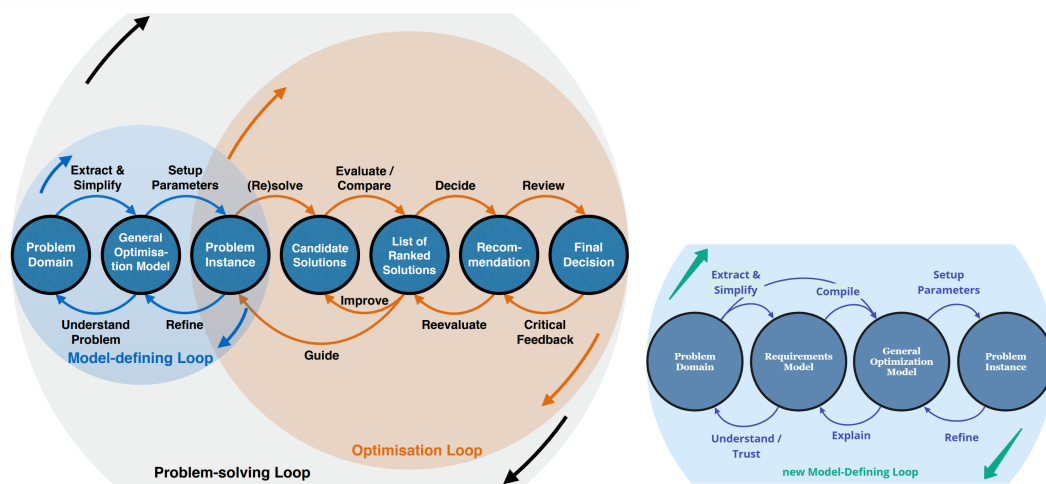
The transition from Phase 1 to Phase 2 of the UNHCR project encountered the usual challenges that exist for optimisation models even after they are successfully integrated into real-world workflows, which is an already challenging task. First, the problem had *drifted* between the two phases and the end users, i.e., the domain experts, did not have sufficient in-house expertise to adjust the implementation to tackle the drift. As a result, they continued to use their original method which was less effective but much more familiar. And second, due to subsequent personnel changes in UNHCR, Phase 2 required not just updating the implemented model, but on-boarding a new domain expert who had no reason to be invested in a new method other than our promises of a better future. While we were lucky enough for our new domain expert to trust us and become invested in developing a better approach, it is easier for organisations to simply continue using their old systems and move on.

In this section, we will present our *initial exploration* of how domain experts could be integrated better into the modelling process. Our proposal is based on our experience with Phases 1 and 2 of the UNHCR problem. Our goal is to develop a new *framework* for integrating domain experts into the modelling process in such a way they can (a) understand and verify how their user requirements have been mapped to the model, thus increasing the model's *transparency*; and (b) modify the model to cope with some level of problem drift, thus increasing the model's *flexibility*. Additionally, such an approach may help organisations obtain a better optimisation model, achieve higher levels of trust in the model, and develop more in-house expertise to manage the life cycle of the model. The rest of this section discusses this framework.

## 5.1 Related Work

The field of *Interactive Optimisation* has a long tradition of introducing domain experts into the development process early on. Meignan et al. [12] present a comprehensive overview of interactive optimisation approaches from operations research studies where domain experts are involved via a User Interface (UI) to guide the *solving* process. It also presents the high level components of an interactive optimisation system that includes a *preference model*, that is, an intermediate model between the UI and the optimisation model that captures user preferences when guiding the solver towards the preferred solution. This approach can be useful for users to find their preferred solutions, particularly for multi-objective optimisation problems with a large set of solutions. However, it can be achieved without domain experts understanding much about the model or being able to modify anything except its search and/or objective function.

Liu et al. [10] introduce a theoretical framework for interactive optimisation problem solving, called the *Problem Solving Loop* (left of Figure 2) and based on the *Sense Making Loop* [16] from the field of Visual Analytics [9]. The Problem Solving Loop aims at directly engaging the domain experts in the optimisation process. In doing this, it captures the high-level user goals and tasks in two major sub-loops: the *model-defining loop* and the *optimisation loop*, which correspond to the modelling design phase and the decision making phase, respectively.



■ **Figure 2** Problem solving loop presented by Liu et al. [10] and updated Model Defining Loop.



However, Liu et al. focus mostly on the latter, which covers the utilisation of the model by the domain experts from the moment they execute a *Problem Instance* of the *General Optimisation Model*, to the moment they make a final decision based on a *List of Ranked Solutions* (see left of Figure 2). Their model-defining loop does capture the fact that, in practice, problems are not converted to models in a singular step; rather, this involves multiple iterations of discussion among different stakeholders. However, they mostly ignore this sub-loop because they argue that its tasks, which lead to the creation of the *General Optimisation Model* and its *Problem Instance*, are usually done by modelling experts. While this is certainly the case currently, it does not mean it is appropriate, as we discuss below.

## 5.2 An Updated Model-Defining Loop

Our experience in several real-world projects indicates that to increase the model's transparency and flexibility, we must deeply involve domain experts in the model-defining loop. Further, we need to do so by giving them a formal role and a formal language with which to communicate not only with the modelling experts but with the model itself. This will enable domain experts to verify the modelling experts' perception of the problem, as well as to directly refine and modify the model, both as part of its initial definition and its ongoing maintenance. Thus, it should help address problem drift. To this end, we focus on extending this loop with the addition of a *Requirements Model* as depicted on the right of Figure 2. The Requirements Model captures user requirements in a *high-level, expressive intermediate formal language* at the early stages of the project. This language will have to be different for each application area, in order to allow both the modelling experts and the domain experts to express the parts of the model they are responsible for. Thus, we propose for the Requirements Model to be expressed in a *Domain Specific Language* (DSL) [13, 21, 8]. DSLs offer powerful expressivity while being tailored to a specific application domain, and have proven successful in many different application areas [6].

We call the DSLs that are used for the Requirements Model *Modelling DSLs, or MDSLs*. Our proposal is for the Requirements Model, expressed in an application-specific MDSL, to be compiled into a General Optimisation Model, which can then be instantiated and solved in the usual way. It is the responsibility of the MDSL designers to develop a compiler (often referred to as a generator in the DSL literature [15]) that generates efficient and correct code for the General Optimisation Model.

An important design decision is the level of abstraction and complexity required of an MDSL. On the one hand, we could make it as expressive as a General Purpose Modelling Language (GPML) such as MiniZinc, but this would defeat the purpose of simplicity and easy communication with the domain experts. On the other hand, significantly restricting its expressivity could severely limit its usefulness. For this reason, we propose to split the General Optimisation Model into two parts. A *fixed part*, which is implemented by the modelling expert; and a *flexible part*, which is generated (compiled) from an MDSL specification.

In the following, we will discuss which parts of an optimisation model such an MDSL should be able to express, and at what level of abstraction the MDSL should be designed. We will then see a concrete example of an MDSL for the UNHCR fund allocation problem.

### 5.3 MDSL expressivity

When designing an MDSL, it is important to select an appropriate level of expressivity, striking a balance between ease of use (especially for non-experts) and usefulness. While there are guidelines for general DSL design [8, 21, 13], we will focus here on the specific aspects related to Modelling DSLs. Let us look at the basic parts of an optimisation model and discuss how those parts are amenable to being expressed within MDSLs.

#### 5.3.1 Variables

The choice of variables for an optimisation problem is one of the fundamental modelling decisions. It has a profound impact on the way the constraints are modelled and on the performance of solving algorithms – a fact that domain experts would usually not be aware of. Therefore, while it is important to define variables in a way that is easy to understand by the domain experts, we argue that the choice itself should not be left to domain experts. The modelling experts should define the variables, which are then made available to domain experts to be used (but not modified) via the MDSL.

This results in an architecture where the General Optimisation Model in our updated model-defining loop is split into two parts: a *fixed* part and a *flexible* one. The modelling experts design the former, which includes the choice of decision variables and is expressed in a General Purpose Modelling Language. This fixed part of the model cannot be modified through the MDSL, and any updates require input from a modelling expert. In addition, the modelling and domain experts collaborate on the *flexible model part*, which captures the Requirements Model and is defined using an MDSL. The two parts together are compiled into the General Optimisation Model, which can then be instantiated with data and solved.

Figure 3 shows a more detailed view of the new model-defining loop that includes this split into a fixed core and a flexible MDSL model.

#### 5.3.2 Parameters

Parameters capture the core objects and data that the model is concerned with. We can split parameters into two kinds: *entities* and *attributes*. *Entities* define the main objects in a problem. For example, in Phase 1 of the UNHCR model, the main entity was an `ITEM`, while Phase 2 had two entities, `PROJECT` and `FUND`. *Attributes* define the *properties* of the entities. For example, in our Phase 1 model the `ITEM` entity had attributes such as `income` and `budget`, while in Phase 2 we saw `requirement` for `PROJECT` and `balance` for `FUND`.

Entities and their attributes commonly appear in the definitions of the constraints and/or the objective function in the model. For example, the balance of a fund will appear in the rules that govern how that fund can be used. The transition from Phase 1 to Phase 2 in the UNHCR problem saw the addition of new attributes (such as cost centre) and the removal of others (such as budget), which required changes to the corresponding funding rules. An MDSL can accommodate attribute changes by adding support for defining attributes and then using them consistently in the constraints and objective function. To do this we can build on the numerous DSLs commonly used in practice for this purpose, such as UML diagramming languages [2], class diagrams used in the object oriented paradigm and entity-relationship diagramming languages [3] used in database design. Note that this part of the MDSL is not application specific and can be used across many domains.

Accommodating changes in entities is more complex, as user decisions often relate to entities. For example, in the UNHCR problem we had to decide how much funding to move from each *fund* to each *project*. Thus, entity changes often require changes in the variables.

## 37:12 Addressing Problem Drift in UNHCR Fund Allocation

Based on our rationale above for leaving the choice of variables to the modelling experts, we do not propose yet to add support into MDSLs for defining new entities. However, a more abstract version of entity might be needed to avoid simple changes in input format.

### 5.3.3 Constraints

Supporting a close collaboration between modelling and domain experts during the definition and refinement of constraints is key for ensuring transparency. Supporting domain experts in modifying them is key for flexibility. As a result, any MDSL will need to be designed such that constraints can be specified and modified in a way that is natural for the domain experts. The Requirements Model, expressed in the MDSL, will serve as a means of communication and documentation between the modelling and domain experts. Therefore, the main challenge when applying our approach is designing an application-specific MDSL with sufficient expressivity to capture current and likely future constraints, yet sufficient simplicity to be understandable by domain experts. A prototype MDSL for the UNHCR problem is presented in Section 5.4.

### 5.3.4 Objective function

Changes to the objective function are also common, both during the modelling process and during the life cycle of a model [11]. To support this, MDSLs can provide basic arithmetic expressions involving all model variables and parameters that can then be used to define and/or modify the objective. This however is not enough for defining extra variables that may be needed to modify the objective. Since we do not want domain experts to introduce new variables using the MDSL, modelling experts will have to foresee and pre-define any auxiliary variables that may be exposed via the MDSL. For example, if the auxiliary variables for `total_allocation` and `minimum_allocation` are pre-modelled, domain experts will subsequently be able to formulate new objective functions using them.

The combination of basic arithmetic expressions and pre-defined variables also allows MDSLs to support domain experts in experimenting with multi-objective solving by combining different objectives in a weighted sum. These weights can then be changed by domain experts (including setting them to 0 to remove certain terms from the objective). Domain experts can even add new terms into the objective if required.

### 5.3.5 Proposed framework

Given the above discussion, we designed our framework to expand the functionality of the updated model-defining loop suggested in Section 5.2 with a new workflow depicted in Figure 3. As discussed earlier, the Requirements Model is used to *extract and simplify* the change requests and to capture them using an MDSL. Domain experts use it to *understand* how constraints are mapped to the model and to build *trust* in the model. The Requirements Model is then *compiled* to generate GPML code to be added to the ***General Optimisation Model***. An interesting extension would be to support bidirectional compilation, where GPML code can be reflected back into the Requirements Model. This would in turn allow the model to be *explained* in terms of the MDSL in the Requirements Model.

### 5.3.6 Natural vs Formal Languages for MDSLs

The main function of the MDSL is to serve as the interface between the domain expert and the optimisation model. Therefore, it has to strike a balance between being easy to understand by the domain expert, easy to implement, and easy to compile into a general

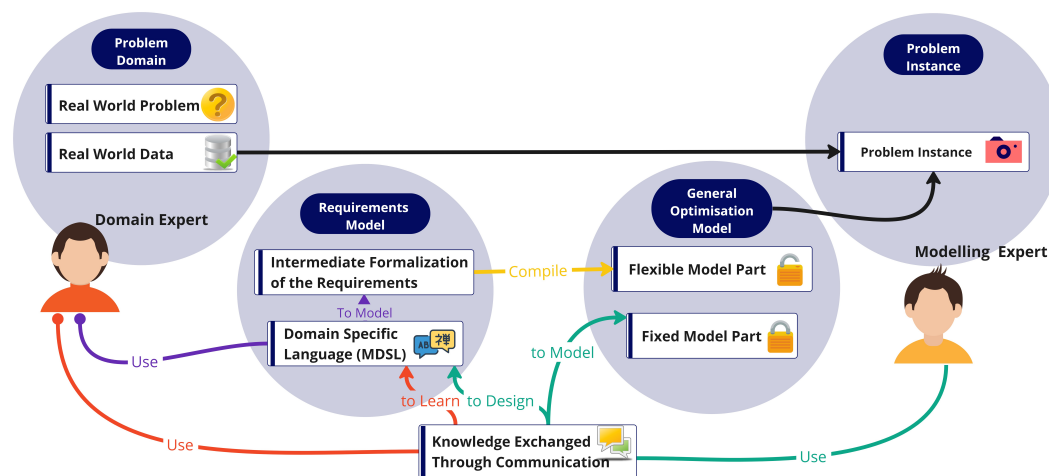
optimisation model. Being easy to understand suggests designing a language closer to the natural language a domain expert would use to describe the problem in a conversation; being easy to implement and to compile suggests one closer to the mathematical model that is going to be the result of the compilation process. Unfortunately, the vague and ambiguous nature of natural language can be problematic for an optimisation model, as its aim is to capture a problem specification precisely and unambiguously.

Let us illustrate this with an example from our UNHCR model. The following are statements from our domain experts, communicating to us their rules for transferring money:

1. Can transfer money from a fund to a project if the pillar of the fund is “all pillars“ or the pillars of the fund and the projects match
2. Can transfer money from a fund to a project if the situation of the fund is “regular situation“ or the situation of the fund and the projects match
3. Can transfer money from a fund to a project if the level of fund is equal to 2 and the domains of the fund and project match
4. Can transfer money from a fund to a project if the level of the fund is equal to 3 and the regions of the fund and project match

These statements have several issues. First, while they are expressed as *Can transfer*, expressing “possibility“ in an optimisation model is not easy. For example, our model’s `movement[i,j]` variable represents whether there is a transfer (`movement[i,j] > 0`) or not (`movement[i,j]=0`). We can represent this rule as *If movement[i,j] > 0, then the pillar of the fund must be “all pillars“, or the pillars of the fund and the projects must match*. While this captures the correct meaning, it may not be immediately intuitive for the domain expert (or anyone else). An alternative would be to state the contrapositive: *If the pillar of the fund is not “all pillars“ and the pillars of the fund and the projects do not match, then we must not transfer funds, so movement[i,j]=0*. This may be more intuitive, but it requires negating all conditions in the rules, which might be difficult to implement.

The second issue is the ambiguity regarding whether the statements are logically connected by “and“ or by “or“, or if they are supposed to be mutually exclusive. This is where the statements in contra-positive form have an advantage, as each of them stands on its own and it is clearer they should be connected by an “and“. However, even then we still have a third source of ambiguity, as we do not know whether the statements are exhaustive or not. In



■ **Figure 3** Further detailed model-defining loop incorporating MDSLs.

other words: if none of the rules holds, should we set `movement[i,j] = 0` or not? While it turns out the answer is “no“, the opposite could have been true, which is more difficult to achieve in an optimisation model.

A final point to note is that rules 3 and 4, when interpreted as mentioned above, in fact contradict each other: Rule 3 would mean that if the level is not 2, or the domains do not match, no movement is possible, while rule 4 states the same for level 3 and the regions. Clearly, the intended interpretation here is that if `movement[i,j] > 0` *and* the level is 2, then the domains must match.

These examples make it clear that, while a language closer to natural language seems convenient for domain experts, it is not precise enough to capture a unique logical meaning. As capturing the constraints in a mathematically precise format is crucial for optimisation modelling, care must be taken for the MDSLs to strike the right balance between being easy to understand *and* mathematically precise. Given the narrow scope of the language, domain experts will require some training to use an MDSL. However, we anticipate this to be only a minor hurdle when compared to learning a complete modelling language such as MiniZinc.

With the current fast advance of AI-based large-language-model discourse systems, it would be interesting to explore how this technology could be used to bridge the gap between precise mathematical MDSL and natural language. However, it is unclear how the inherent ambiguity in natural language formulations could be avoided.

## 5.4 An MDSL for UNHCR fund allocation problem

Our design goal for the MDSL grammar of the new UNHCR problem was to support domain experts in formalising any new required changes to the model in terms of the given set of parameters and variables. Compared to the full MiniZinc language, the MDSL grammar should be significantly simpler, covering only a small sub-set of MiniZinc’s capabilities specific to this problem domain. Most of the constraints in the UNHCR problem are expressed in the form of ad-hoc *rules* that consist of *conditions* and *consequences* in the form of *if-then-else* statements. This level of modelling is close to the actual constraints, while still being relatively easy to understand even for someone without any programming experience.

We first define the *abstract* syntax for the MDSL in the form of the following grammar:

```

<rule>      ::= <logical-expr> | <iteration> | <if-then-else>
<logical-expr> ::= <expr> <compare-op> <expr>
                | "not" <logical-expr>
                | <logical-expr> "and" <logical-expr>
                | <logical-expr> "or" <logical-expr>
<iteration>  ::= "forall" (<identifier>, ...) <logical-expr>
<if-then-else> ::= "if" <logical-expr> "then" <logical-expr> ("else" <logical-expr>)
<compare-op> ::= "=" | "!=" | "<" | "<=" | ">" | ">="
<expr>      ::= <named-expr> | <number> | <expr> <arithmetic-op> <expr>
<named-expr> ::= <identifier> | <identifier> "of" (<identifier>, ...)
                | <identifier> "from" <identifier> "to" <identifier>
<arithmetic-op> ::= "+" | "-" | "*" | "/"

```

The resulting abstract grammar is much simpler than that of MiniZinc. In particular, if-then-else expressions can only have Boolean type and cannot be nested; the grammar only supports one kind of iteration (`forall`); and it restricts the way variables and parameters are addressed. These restrictions were identified as sufficient for UNHCR problem.

While the grammar defines the *kind* of constraints that can be expressed in the MDSL, it does not define the concrete representation of the constraints the domain experts would use. Thus, the grammar can be implemented in several different ways. We now show a concrete text-based language as well as a simple graphical interface below to illustrate the possibilities. However, we do not claim that those are the most intuitive or best suited for this purpose.

A concrete text-based representation of the above abstract MDSL grammar could use syntax that is close to a more natural way of stating constraints, while still being concise and unambiguous. The following example shows a rule as defined in MiniZinc and in a concrete version of the abstract MDSL grammar:

*MiniZinc constraint*

```
constraint forall(f in FUNDS, p in PROJECTS where
  f_level[f] == L4 /\ f_subregion[f] != p_subregion[p]
  (movement[f,p] = 0);
```

*Constraint represented using a concrete version of the abstract MDSL grammar*

```
For Every FUND, PROJECT :
  IF [((level OF FUND )== (L4)) and
      ((subregion OF FUND )!= (subregion OF PROJECT ))]
  THEN
    [(movement FROM FUND TO PROJECT )== (0)]
```

Note how the concrete MDSL grammar simplifies the access to entity attributes, by using notation such as `level OF FUND` instead of having to introduce a new identifier `f` and corresponding expressions `f_level[f]`, as it is done in the MiniZinc model. For two-dimensional arrays like `movement`, it defines an explicit syntax `movement FROM FUND TO PROJECT` that avoids the ambiguity of the direction of movement present in the MiniZinc model. Many other concrete versions of the abstract MDSL grammar are also possible. For example, it would be easy to make the concrete grammar more verbose by writing the iteration as `For Every FUND and PROJECT`. Importantly, the MDSL grammar does not have hard-wired

#### Rule Base

- For Every FUND, PAIR OF PROJECT : IF [((pillar OF FUND )!= (pillar OF PROJECT1 ))and ((pillar OF FUND )!= (all\_pillar))] THEN [( movement FROM FUND TO PROJECT1 )= ( 0 )] ELSE [[edit](#)]
- For Every FUND, PROJECT : IF [((level OF FUND )== (4))and ((subregion OF FUND )!= (subregion OF PROJECT ))] THEN [( movement FROM FUND TO PROJECT )= ( 0 )] ELSE [[edit](#)]

#### Rule Constructor

```
For Every FUND, PROJECT : IF [((level OF FUND )== (4))and ((subregion OF FUND )!= (subregion OF PROJECT ))] THEN [( movement FROM FUND TO PROJECT )= ( 0 )] ELSE []
```

For Every    :

IF

and

==

level

!=

subregion

THEN

=

movement

0

ELSE

■ **Figure 4** User Interface for forming intermediate formalisation.

attribute names. If the set of attributes of an entity is later extended, the MDSL can easily be extended accordingly. Another important choice is that of operator precedences. We opted for full explicit bracketing, in order to avoid ambiguity. However, there are many other approaches, such as indentation-based grouping of expressions that belong together. It is beyond the scope of this paper to explore the advantages and disadvantages of different choices in concrete syntax.

In addition to the text-based representation, a graphical representation of the MDSL specifications could be used. Figure 4 shows a prototype browser-based interface that is still quite close to the text version. It has two main sections: The *Rule Base* section displays the already defined rules, and allows users to turn them on and off as needed. The *Rule Constructor* section allows users to define new rules using drop down list boxes containing various components of the language grammar, such as parameters (entity attributes) and variables. A graphical representation like this (or an integration of the text-based language into a graphical development environment) may have several advantages over pure text-based languages. For instance, precedence between different expressions can be indicated graphically, which may help avoid mistakes in the specification. Furthermore, the graphical interface can restrict the choices to those that are valid in a particular situation.

## 6 Conclusion and Future Work

Many organisations need to solve optimisation problems as part of their core operation, but they still perceive the introduction of technologies like model-based optimisation in their decision support systems as a significant risk. They may appreciate that this technology can solve difficult problems, that it can produce better outcomes than simple algorithms or manual solutions, and that it can incorporate additional factors such as uncertainty, fairness or diversity of solutions. However, the lack of in-house expertise for implementation and maintenance and the perceived black-box nature of the technology are often hurdles that are difficult to overcome. The change of input data formats or requirements after an optimisation solution has been deployed, which we refer to as *problem drift*, adds further risks to this process.

This paper presents the UNHCR fund allocation problem, a combinatorial optimisation problem that deals with the distribution of donated funds to humanitarian projects according to ad-hoc rules set by the donors and operational requirements of UNHCR. While the initial model we implemented in 2016 worked well for the sample data, the problem drift that occurred over the next few years prevented the organisation from fully adopting it, instead continuing to use their in-house algorithm. Phase 2 of our project focused on adapting the initial optimisation model to the changed requirements.

Based on this experience, this paper identifies problem drift in optimisation models and highlights its importance to the community as an area worth investigating. It then proposes to tackle it via an extended framework for model development that incorporates a *Requirements Model* into the Model Defining Loop defined by Liu et al. [10]. The requirements model is based on an application specific Modelling Domain Specific Language (MDSL). We have explored guidelines for the trade-off between general expressivity and ease of use of MDSLs. Our recommendations include that certain parts of a model such as the choice of decision variables need to be left to the optimisation expert, while other parts, such as entity attributes, constraints and objective function, can be exposed in the MDSL. We believe that the introduction of a Requirements Model and MDSL into the modelling process can facilitate collaboration between modelling and domain experts, achieve higher levels of trust with the end users of optimisation technology, as well as equip them with the skills and tools to address certain forms of problem drift in-house. sp An important area for future

work is to evaluate the feasibility, suitability and effectiveness of our proposed framework for addressing problem drift in real-world applications. We acknowledge that the development of an application-specific MDSL together with a model may seem prohibitively expensive. Future work will therefore include exploring the use of template MDSLs for larger problem classes to enable code reuse across applications. Furthermore, we will investigate how to speed up and streamline the MDSL development process using tools such as Language Workbenches [6, 4]. We will also study graphical and hybrid text/graphical MDSLs, which can draw on techniques from research areas such as interactive optimisation and visual analytics. Finally, it would be interesting to explore the use of AI-based dialogue systems such as large language models as a user-facing representation of an MDSL, where the potential ambiguity of natural language becomes a challenging research topic.

---



### References

- 1 A. Biere, M. Heule, and H. van Maaren. *Handbook of Satisfiability*. IOS Press, January 2009.
- 2 Grady Booch, James E. Rumbaugh, and Ivar Jacobson. *The unified modeling language user guide - covers UML 2.0, Second Edition*. Addison Wesley object technology series. Addison-Wesley, 2005.
- 3 Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976. doi:10.1145/320434.320440.
- 4 Sebastian Erdweg, Tijs van der Storm, Markus Völter, Meinte Boersma, Remi Bosman, William R. Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, Gabriël D. P. Konat, Pedro J. Molina, Martin Palatnik, Risto Pohjonen, Eugen Schindler, Klemens Schindler, Riccardo Solmi, Vlad A. Vergu, Eelco Visser, Kevin van der Vlist, Guido H. Wachsmuth, and Jimi van der Woning. The State of the Art in Language Workbenches. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Martin Erwig, Richard F. Paige, and Eric Van Wyk, editors, *Software Language Engineering*, volume 8225, pages 197–217. Springer International Publishing, Cham, 2013. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-319-02654-1\_11.
- 5 Robert Fourer, David M Gay, and Brian W Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.
- 6 Martin Fowler. *Domain-Specific Languages*. Pearson Education, September 2010.
- 7 Alan M. Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel. Essence: A constraint language for specifying combinatorial problems. *Constraints*, 13(3):268–306, September 2008. doi:10.1007/s10601-008-9047-y.
- 8 Paul Hudak. Domain Specific Languages. *Handbook of programming languages*, 3(39-60):23, 1997.
- 9 Youn-ah Kang and John Stasko. Examining the Use of a Visual Analytics System for Sensemaking Tasks: Case Studies with Domain Experts. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2869–2878, December 2012. doi:10.1109/TVCG.2012.224.
- 10 Jie Liu, Tim Dwyer, Kim Marriott, Jeremy Millar, and Annette Haworth. Understanding the Relationship Between Interactive Optimisation and Visual Analytics in the Context of Prostate Brachytherapy. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):319–329, January 2018. doi:10.1109/TVCG.2017.2744418.
- 11 Jie Liu, Tim Dwyer, Guido Tack, Samuel Gratzl, and Kim Marriott. Supporting the Problem-Solving Loop: Designing Highly Interactive Optimisation Systems. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1764–1774, February 2021. doi:10.1109/TVCG.2020.3030364.



- 12 David Meignan, Sigrid Knust, Jean-Marc Frayret, Gilles Pesant, and Nicolas Gaud. A Review and Taxonomy of Interactive Optimization Methods in Operations Research. *ACM Transactions on Interactive Intelligent Systems*, 5(3):1–43, October 2015. doi:10.1145/2808234.
- 13 Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.
- 14 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, Lecture Notes in Computer Science, pages 529–543, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-74970-7\_38.
- 15 Václav Pech. JetBrains MPS: Why Modern Language Workbenches Matter. In Antonio Bucchiarone, Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio, editors, *Domain-Specific Languages in Practice: with JetBrains MPS*, pages 1–22. Springer International Publishing, Cham, 2021. doi:10.1007/978-3-030-73758-0\_1.
- 16 P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of international conference on intelligence analysis*, volume 5. McLean, VA, USA, 2005.
- 17 Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier, August 2006.
- 18 UNHCR Global Appeal 2023. Accessed on 5 May, 2023. URL: <https://reporting.unhcr.org/globalappeal2023>.
- 19 UNHCR Global Report 2022. Accessed on 5 May, 2023. URL: <https://reporting.unhcr.org/global-report-2022>.
- 20 UNHCR operations plan in emergencies. Accessed on 5 May, 2023. URL: <https://emergency.unhcr.org/support-response/planning-and-programming/unhcr-operations-plan-emergencies>.
- 21 Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. *ACM SIGPLAN Notices*, 35(6):26–36, June 2000. doi:10.1145/352029.352035.
- 22 Pascal Van Hentenryck. *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA, 1999.
- 23 Laurence A. Wolsey. *Integer Programming*. John Wiley & Sons, September 2020.

# From Formal Boosted Tree Explanations to Interpretable Rule Sets

Jinqiang Yu  

Department of Data Science and AI, Monash University, Clayton, Australia  
Australian Research Council OPTIMA ITTC, Clayton, Australia

Alexey Ignatiev  

Department of Data Science and AI, Monash University, Clayton, Australia

Peter J. Stuckey  

Department of Data Science and AI, Monash University, Clayton, Australia  
Australian Research Council OPTIMA ITTC, Clayton, Australia

---

## Abstract

The rapid rise of Artificial Intelligence (AI) and Machine Learning (ML) has invoked the need for *explainable AI* (XAI). One of the most prominent approaches to XAI is to train rule-based ML models, e.g. decision trees, lists and sets, that are deemed interpretable due to their transparent nature. Recent years have witnessed a large body of work in the area of constraints- and reasoning-based approaches to the inference of interpretable models, in particular decision sets (DSes). Despite being shown to outperform heuristic approaches in terms of accuracy, most of them suffer from scalability issues and often fail to handle large training data, in which case no solution is offered. Motivated by this limitation and the success of gradient boosted trees, we propose a novel anytime approach to producing DSes that are both accurate and interpretable. The approach makes use of the concept of a generalized formal explanation and builds on the recent advances in formal explainability of gradient boosted trees. Experimental results obtained on a wide range of datasets, demonstrate that our approach produces DSes that more accurate than those of the state-of-the-art algorithms and comparable with them in terms of explanation size.

**2012 ACM Subject Classification** Computing methodologies → Machine learning

**Keywords and phrases** Decision set, interpretable model, gradient boosted tree, BT compilation

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.38

**Supplementary Material** *Software (Source Code)*: <https://github.com/jinqiang-yu/cpl/>  
archived at `swh:1:dir:40fde451a732a518f78caa9ad372a7c267446836`

**Funding** This research was partially funded by the Australian Government through the Australian Research Council Industrial Transformation Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Project ID IC200100009.

## 1 Introduction

Rapid development of Artificial Intelligence (AI) and Machine Learning (ML) have revolutionized all aspects of human lives in recent years [30, 1]. However, decisions made by most widely used ML models are hard for humans to understand hence the interest in the theory and practice of *Explainable AI* (XAI) rises.

One major approach to XAI is to compute *post-hoc* explanations for ML predictions to answer a “*why*” question [34, 44], i.e. why the prediction is made. Although heuristic approaches to post-hoc explanations prevail [34, 44, 43], they suffer from a number of weaknesses [21, 16, 49, 52]. Formal methods [48, 20, 37] provide alternative approaches to explanations that avoid these weaknesses. Another alternative approach to XAI is to compute *interpretable* ML models, i.e. logic-based models, including decision trees [40],



© Jinqiang Yu, Alexey Ignatiev, and Peter J. Stuckey;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 38; pp. 38:1–38:21



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

decision lists [46], and decision sets [29]. These models enable decision makers to obtain succinct explanations from the models directly. In this paper, we focus on the decision set (DS) models.

Decision sets are particularly easy to explain: the rule that fired is an explanation of the decision. This led to an upsurge in interest of decision sets that are both interpretable and accurate. Recent work [50] uses propositional satisfiability (SAT) to generate minimum-size decision sets that are perfectly accurate on the training data, and demonstrates that decision sets that completely agree with the training data outperform others in terms of accuracy. A more scalable maximum satisfiability (MaxSAT) approach [18] to this problem was then proposed. Unfortunately, both of these methods are unable to provide any decision information if a dataset is not completely solved.

Motivated by these works and their limitations, this paper aims at making a bridge between formal post-hoc explainability and interpretable DS models. In particular, the paper focuses on developing a novel anytime approach to computing decision sets that are both interpretable and accurate, by compiling a gradient boosted tree model into a decision set on demand with the use of formal explanations. This is done with the use of the recent approach [17] to compute abductive explanations for gradient boosted trees using maximum satisfiability (MaxSAT). Furthermore, the paper proposes a range of post-hoc model reduction heuristics aiming at enhancing interpretability of the result models, done with MaxSAT and integer linear programming (ILP). The experimental results show that compared with other state-of-the-art methods, decision sets generated by the proposed approach are more accurate, and comparable with the competition in terms of interpretability.

## 2 Preliminaries

**SAT and MaxSAT.** The standard definitions for propositional satisfiability (SAT) and maximum satisfiability (MaxSAT) solving are assumed [3]. A propositional formula  $\phi$  is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses. A *clause* is a disjunction of literals, where a *literal* is either a Boolean variable  $b$  or its negation  $\neg b$ . A *truth* assignment  $\mu$  is a mapping from the set of variables to  $\{0, 1\}$ . A clause is said to be *satisfied* by truth assignment  $\mu$  if one of the literals in the clause is assigned value 1; otherwise, the clause is *falsified*. If all clauses in formula  $\phi$  are satisfied by assignment  $\mu$ ,  $\phi$  is satisfied; otherwise, assignment  $\mu$  falsifies  $\phi$ . A CNF formula  $\phi$  is *unsatisfiable* if there exists no assignment satisfying  $\phi$ .

In the context of unsatisfiable formulas, the MaxSAT problem consists in finding a truth assignment that maximizes the number of satisfied clauses. Hereinafter, we use a variant of MaxSAT called Partial Weighted MaxSAT [3, Chapters 23 and 24]. The formula  $\phi$  in this variant is represented as a conjunction of *hard* clauses  $\mathcal{H}$ , which must be satisfied, and *soft* clauses  $\mathcal{S}$  where each of them is associated with a weight representing a preference to satisfy them, i.e.  $\phi = \mathcal{H} \wedge \mathcal{S}$ . Partial Weighted MaxSAT problems aim at finding a truth assignment  $\mu$  that satisfies all hard clauses and maximizes the total weight of satisfied soft clauses.

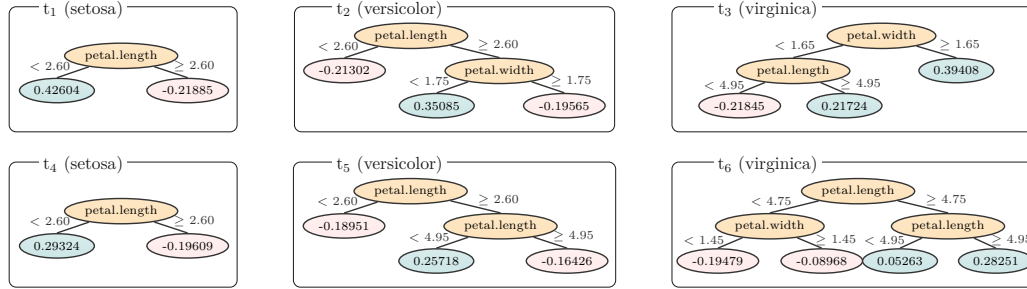
**Classification Problems.** We consider classification problems with a set of classes<sup>1</sup>  $\mathcal{K} = \{1, \dots, k\}$ , and a set of features  $\mathcal{F} = \{1, \dots, m\}$ . The value of each feature  $i \in \mathcal{F}$  is taken from its corresponding (numeric) domain  $D_i$ . As a result, the entire feature space is defined as

---

<sup>1</sup> Non-integer class labels can be mapped to a set  $\{1, \dots, |\mathcal{K}|\}$ .

IF “petal.length” < 2.60	THEN class = “setosa”
IF 2.60 ≤ “petal.length” < 4.95 ∧ “petal.width” < 1.75	THEN class = “versicolor”
IF “petal.length” ≥ 2.60 ∧ “petal.width” ≥ 1.75	THEN class = “virginica”
IF “petal.length” ≥ 4.95	THEN class = “virginica”

(a) Decision set.



(b) BT model [5] consisting of 2 trees per class, each of depth ≤ 2, adopted from [17].

■ **Figure 1** Example DS and BT models computed on the well-known *Iris* classification dataset.

$\mathbb{F} \triangleq \prod_{i=1}^m D_i$ . A concrete point represented by  $\mathbf{v} = (v_1, \dots, v_m) \in \mathbb{F}$ , s.t. each  $v_i$  is a constant value taken by feature  $i \in \mathcal{F}$ , together with its corresponding class  $c \in \mathcal{K}$ , represented by a pair  $(\mathbf{v}, c)$ , indicate a *data instance* or *example*. With a slight abuse of notation and whenever convenient, a data point  $\mathbf{v} \in \mathbb{F}$  is also referred to as an instance. Finally,  $\mathbf{x} = (x_1, \dots, x_m)$  denotes a vector of feature variables  $x_i \in D_i, i \in \mathcal{F}$ , used for reasoning over points in  $\mathbb{F}$ .

A classifier defines a *classification function*  $\tau: \mathbb{F} \rightarrow \mathcal{K}$ . The objective of classification problems is to learn a function  $\tau$  to generalize well on unseen data given a training dataset  $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$ , where each instance  $e_d \in \mathcal{E}$  is a pair of  $(\mathbf{v}_d, c_d)$ . Classification problems are conventionally posed as an optimization problem, i.e. either to minimize the complexity of  $\tau$ , or maximize its accuracy, or both.

**Rules, Decision Sets and Gradient Boosted Trees.** Multiple ways exist to learn classifiers given data  $\mathcal{E}$ . This paper focuses on arguably one of the most interpretable models, i.e. decision sets, trained by *compiling* gradient boosted trees.

A *decision rule* is in the form of “IF antecedent THEN prediction”, where the antecedent is a set of feature literals. Informally, a rule is said to classify an instance  $\mathbf{v} \in \mathbb{F}$  as class  $c \in \mathcal{K}$  if its antecedent is *compatible* with  $\mathbf{v}$  (or *matches*  $\mathbf{v}$ ) and its prediction is  $c$ . A *decision set* (DS) is an unordered set of decision rules  $\mathcal{R}$ . An instance  $(\mathbf{v}, c) \in \mathcal{E}$  is misclassified by a DS if either there exists no rule in  $\mathcal{R}$  matching  $\mathbf{v}$ , or there exists a rule classifying  $\mathbf{v}$  as a class  $c' \in \mathcal{K}$  s.t.  $c' \neq c$ .

A *gradient boosted tree* (BT) is a tree ensemble  $\mathfrak{T}$  defining sets of decision trees  $T_c \in \mathfrak{T}$  for each class  $c \in [|\mathcal{K}|]$ , where  $T_c$  comprises  $N \in \mathbb{N}_{>0}$  trees  $t_{kz+c}, z \in \{0, \dots, N-1\}, k = [|\mathcal{K}|]$ . Given an instance  $\mathbf{v} \in \mathbb{F}$ , its class is obtained by computing the sum of scores assigned by trees for each class  $w(\mathbf{v}, c) = \sum_{t \in T_c} t(\mathbf{v})$  and assigning the class which has the maximum score, i.e.  $\operatorname{argmax}_{c \in [|\mathcal{K}|]} w(\mathbf{v}, c)$ . Whenever convenient,  $\mathbf{n} \in t$  denotes a non-terminal node, where  $t \in \mathfrak{T}$  represents an arbitrary decision tree. Moreover, each such  $\mathbf{n}$  indicates a feature condition in the form of  $x_i < d$ , where feature  $i \in \mathcal{F}$  and *splitting threshold*  $d \in \mathcal{D}_i$ .

■ **Table 1** Several instances extracted from *Iris* dataset.

#	sepal.length	sepal.width	petal.length	petal.width	class
$e_1$	5.1	3.5	1.4	0.2	setosa
$e_2$	7.7	2.6	6.9	2.3	virginica
$e_3$	5.6	2.5	3.9	1.1	versicolor
$e_4$	6.2	2.8	4.8	1.8	virginica
$e_5$	5.6	2.8	4.9	2.0	virginica

► **Example 1.** Figure 1 shows DS and BT models trained on the *Iris* dataset, which has 4 numeric features and 3 classes: “*setosa*”, “*versicolor*”, and “*virginica*”. Observe that instance  $\mathbf{v}_1 \in e_1$  shown in Table 1 is classified as “*setosa*” by the first rule of the DS. In the BT model, each class  $c \in [3]$  is represented by 2 trees  $t_{3z+c}$ ,  $z \in \{0, 1\}$ . Thus, it also classifies  $\mathbf{v}_1$  as “*setosa*”, since the score of this class  $w(\mathbf{v}_1, 1) = t_1 + t_4 = 0.71928$  is higher than the score of “*versicolor*”  $w(\mathbf{v}_1, 2) = t_2 + t_5 = -0.40253$  and the score of “*virginica*”  $w(\mathbf{v}_1, 3) = t_3 + t_6 = -0.41324$ .  $\lrcorner$

**Interpretability and Explanations.** Interpretability is not formally defined as it is considered to be a subjective concept [33]. In this paper interpretability is defined as the overall succinctness of the information offered by an ML model to justify a provided prediction. Moreover, following earlier work [48, 20], we equate explanations for ML models with *abductive explanations* (AXps), which are subset-minimal sets of features sufficient to explain a given prediction. Concretely, given an instance  $\mathbf{v} \in \mathbb{F}$  and a prediction  $c = \tau(\mathbf{v}) \in \mathcal{K}$ , an AXp is a subset-minimal set of features  $\mathcal{X} \subseteq \mathcal{F}$  such that

$$\forall(\mathbf{x} \in \mathbb{F}). \left[ \bigwedge_{i \in \mathcal{X}} (x_i = v_i) \right] \rightarrow (\tau(\mathbf{x}) = c) \quad (1)$$

► **Example 2.** Consider the setup of Example 1. Given instance  $\mathbf{v}_1$ , observe that for any instance with “*petal.length*” = 1.4, the BT is guaranteed to predict “*setosa*” independently of the values of other features, since the weights for “*setosa*” and “*versicolor*” are 0.71928 and  $-0.40253$  respectively as before, and the maximal weight for “*virginica*” is  $0.39408 - 0.08968 = 0.30440$ . Thus, the (only) AXp  $\mathcal{X}$  for the prediction for  $e_1$  made by the BT model is {“*petal.length*”}.  $\lrcorner$

**Explanations in BTs.** Formal reasoning has been recently applied to computing AXps for BT models, with the key difficulty being how to effectively reason about the aggregation over a large number of trees in a BT model. Recent work applied satisfiability modulo theory (SMT) [21] or mixed integer linear programming (MILP) solvers [42, 27] to directly address the linear summations arising in the BT encoding. Hereinafter, we build on the recent MaxSAT approach [17], which maps the aggregation reasoning to a set of MaxSAT queries to avoid a costly encoding of the linear constraints into CNF. Also, [17] demonstrates how a MaxSAT query can be made such that (1) holds if and only if the *optimal* value of the constructed objective function is negative.<sup>2</sup> In general, assuming that each feature  $i \in \mathcal{F}$  is numeric (continuous), the approach orders the set of splitting thresholds  $\{d_{i1}, \dots, d_{ih_i}\}$  in a BT  $\mathfrak{T}$  for each feature  $i$ , where  $h_i$  is the total number of thresholds of feature  $i$  in  $\mathfrak{T}$  and  $d_{ij} \in \mathcal{D}_i$  for  $j \in [h_i]$ . Given an instance  $\mathbf{v} = (v_1, \dots, v_m) \in \mathbb{F}$ , the above approach

<sup>2</sup> The reader is referred to [17] for the details.

associates each value  $v_i$  with a single interval  $I'_i$  from the set of disjoint intervals  $\mathbb{D}_i = \{ I_{i1} \equiv [\min(\mathcal{D}_i), d_{i1}), I_{i2} \equiv [d_{i1}, d_{i2}), \dots, I_{ih_i+1} \equiv [d_{ih_i}, \max(\mathcal{D}_i)] \}$ . Thus, AXp extraction boils down to finding a subset-minimal subset  $\mathcal{X} \in \mathcal{F}$  s.t.

$$\forall(\mathbf{x} \in \mathbb{F}). \left[ \bigwedge_{i \in \mathcal{X}} x_i \in I'_i \right] \rightarrow (\tau(\mathbf{x}) = c) \quad (2)$$

► **Example 3.** Recall Example 2 and assume “*petal.length*” and “*petal.width*” have indices 3 and 4. Note that the sets of splitting thresholds for feature “*petal.length*”  $\{d_{31} = 2.60, d_{32} = 4.75, d_{33} = 4.95\}$  and for feature “*petal.width*”  $\{d_{41} = 1.45, d_{42} = 1.65, d_{43} = 1.75\}$ . Let  $\min(\mathcal{D}_3) = -\infty$  and  $\min(\mathcal{D}_4) = 0.1$ . Then we can associate the values of features 3 and 4 in our instance  $\mathbf{v}_1 \in e_1$  with intervals  $I_{31} \equiv (-\infty, 2.60)$  and  $I_{41} \equiv [0.1, 1.45)$ . Hence by (2), the AXp shown in Example 2 can in fact be seen as a rule  $\langle IF \text{ “petal.length”} < 2.60 \text{ THEN class} = \text{“setosa”} \rangle$ . ◻

### 3 Related Work

Interpretable decision sets are logic-based ML models that can be traced back to the 70s and 80s [39, 15, 4, 45]. To the best of our knowledge, [6] proposed the first approach to decision sets, which were introduced as the variant of decision lists [45, 7]. The first method making use of logic and optimization to synthesize a disjunction of rules that match a given dataset was proposed in [26]. Recent work [29] argued that decision sets are more interpretable than the other logic-based models, i.e. decision lists and decision trees. This work uses smooth local search to generate a set of rules first and heuristically minimizes a linear combination of criteria afterwards, e.g. the size of a rule, their maximum number, overlap or error.

Since then a number of works proposed the use of logic reasoning and optimization procedures to train DS models [22, 36, 12, 50, 18] claiming to significantly outperform the approach of [29] in terms of accuracy and performance. Among those, the works closest to ours are [22, 50, 18]. They proposed SAT-based approaches to computing smallest-size decision sets that *perfectly* agree with the training data by minimizing either the number of rules [22, 18] or the number of literals [50, 18] used in the model. Additionally, [50] is capable of computing *sparse* decisions sets that trade off training accuracy for model size. Despite the dramatic performance increase achieved in [18], all the approaches above suffer from scalability issues.

Post-hoc explainability is one of the major approaches to XAI. Besides a plethora of heuristic sampling-based methods to post-hoc explainability [43, 34, 44], a formal reasoning based approach to computing abductive explanations [48, 20] stands out. AXps can be related with prime implicants of the decision function (hence an alternative name *prime implicant explanations*, *PI-explanations*) associated with ML predictions and are guaranteed to capture the semantics of the ML models in the entire feature space. Although hard to compute in general, AXps were shown to be effectively computable for BT models by an incremental MaxSAT-based approach [17].

Our work aims at making a bridge between interpretable DS models and AXp computation by exploiting the latter for training the former. Given a BT model, it focuses on generating decision rules that agree with the BT. Each rule represents an AXp for the prediction made by the BT model, resulting in a DS model in a way *guided* by the original BT model. The approach is shown to outperform the prior logic-based approaches to DS inference in terms of test accuracy and performance. Note that despite prior attempts to train sparse models guided by tree ensembles [38], to our best knowledge, none of the existing works have applied formal post-hoc explanations to compile interpretable models.

Finally, our approach can be related to the existing line of work on *knowledge distillation* [11, 13], where an interpretable model is trained to approximate a hard-to-interpret black-box model, which is often seen as teacher-to-student knowledge transfer. Note that in contrast to knowledge distillation, our approach is able to *compile* a BT into an *equivalent* DS if we consider the entire feature space, as shown below.

## 4 Decision Sets by Boosted Tree Compilation

Based on [17], this section details a MaxSAT-based approach to compiling a BT into a DS where each rule in the DS is equivalent to a prime implicant of the BT classification function.

### 4.1 Rule Extraction

Recall that an AXp, as defined in (1) and (2), can be seen as an *if-then* rule. Given a hard-to-interpret BT model, the AXp extraction approach of [17] can be modified to compute an interpretable DS consisting of a *set* of AXps for the BT. However, when the features are continuous (numeric), this potential approach suffers from the following issue. Recall that an AXp  $\mathcal{X} \in \mathcal{F}$  indicates a set of *concrete* feature values that are sufficient to explain a prediction  $c = \tau(\mathbf{v})$  for a certain instance  $\mathbf{v} \in \mathbb{F}$ . Although this same AXp can explain other instances compatible with it, its applicability in general is at the mercy of expressivity of the feature literals used in the AXp, i.e. equality literals and succinct interval membership in the case of (1) and (2), respectively. Motivated by this limitation, we propose to compute AXps over the literals intrinsic to the BT model aiming at getting feature intervals that are as general as possible, as detailed below.<sup>3</sup>

In contrast to the work of [17], which associates each feature value  $v_i \in D_i$  with a single *narrowest* interval  $I'_i$  covering the value, we exploit all the splitting points used by the BT for feature  $i$  and identify all of the corresponding literals satisfied by the feature value  $v_i$ . Note that the original MaxSAT encoding [17] introduces a single Boolean variable  $o_{ij}$  for each literal  $x_i < d_{ij}$  with  $d_{ij}$  being a  $j$ 'th threshold used in the BT for feature  $i$ , s.t.  $o_{ij} = 1$  iff  $x_i < d_{ij}$  holds true. This way, each positive  $o_{ij}$  represents an upper bound on the value of  $x_i$  while each negative  $\neg o_{ij}$  represents a lower bound on  $x_i$ .

► **Example 4.** Feature 3 (“*petal.length*”) from Example 3 has 3 thresholds:  $d_{31} = 2.60$ ,  $d_{32} = 4.75$ ,  $d_{33} = 4.95$ . Boolean variables  $o_{31}$ ,  $o_{32}$ , and  $o_{33}$  are set to true iff  $x_3 < 2.60$ ,  $x_3 < 4.75$ , and  $x_3 < 4.95$ , respectively. Let feature 3 take value 3.9 in the instance we want to explain. Observe how we can immediately assign literals  $\neg o_{31}$ ,  $o_{32}$ , and  $o_{33}$  to true. ◻

Next, given an instance  $\mathbf{v} = (v_1, \dots, v_m) \in \mathbb{F}$ , let us construct a complete conjunction  $\bigwedge_{i \in \mathcal{F}, j \in [h_i]} \tilde{o}_{ij}$  of literals  $\tilde{o}_{ij}$  s.t.  $\tilde{o}_{ij}$  is to be replaced by  $o_{ij}$  if  $v_i < d_{ij}$  and replaced by  $\neg o_{ij}$  otherwise. By construction, this conjunction holds true for instance  $\mathbf{v}$ . Now, given this conjunction of literals, we can apply the existing approach of [17] to extract a subset-minimal explanation  $\mathcal{Y} \subseteq \{\tilde{o}_{ij} \mid i \in \mathcal{F}, j \in [h_i]\}$  for instance  $\mathbf{v}$  over literals  $\tilde{o}_{ij}$  s.t.

$$\forall(\mathbf{x} \in \mathbb{F}). \left[ \bigwedge_{l \in \mathcal{Y}} l \right] \rightarrow (\tau(\mathbf{x}) = c) \quad (3)$$

Such an explanation  $\mathcal{Y}$  may (or may not) define either a lower bound on feature  $i$ , an upper bound, or both, aiming to construct the *most general* interval for each feature  $i \in \mathcal{Y}$ . Hence, we informally refer to such explanations as *generalized AXps* or simply *rules* (hereinafter, we use both interchangeably).

<sup>3</sup> An alternative to our approach is *inflation* of abductive explanations, which is discussed in [23, 24]. Given an AXp, it aims at extending the set of values covered by each feature literal in the AXp while the AXp condition (1) still holds.

---

**Algorithm 1** Deletion-based Rule Extraction.
 

---

**Function:** RuleExtract( $\mathfrak{T}, \mathbf{v}, c, \mathcal{E}$ )**Input:**  $\mathfrak{T}$ : BT defining  $\tau(\mathbf{x})$ ,  $\mathbf{v}$ : Instance,  $c$ : Prediction, i.e.  $c = \tau(\mathbf{v})$   $\mathcal{E}$ : Training data**Output:**  $\mathcal{Y}$ : Subset-minimal rule

```

1:  $\langle \mathcal{H}, \mathcal{S} \rangle \leftarrow \text{Encode}(\mathfrak{T})$ 
2:  $\mathcal{Y} \leftarrow \text{Init}(\mathfrak{T}, \mathbf{v})$ 
3:  $\mathcal{Y} \leftarrow \text{Sort}(\mathcal{Y}, \mathcal{E})$ 
4: for  $l \in \mathcal{Y}$  do
5:   if EntCheck( $\langle \mathcal{H}, \mathcal{S} \rangle, c, \mathcal{Y} \setminus \{l\}$ ) then
6:      $\mathcal{Y} \leftarrow \mathcal{Y} \setminus \{l\}$ 
7: return  $\mathcal{Y}$ 

```

---

► **Example 5.** Consider instance  $\mathbf{v}_3$  predicted as “*versicolor*” by the BT (observe that  $v_3 = 3.9$  and  $v_4 = 1.1$ ) and recall the thresholds for features 3 and 4 discussed in Example 3. We can compute a generalized AXp  $\mathcal{Y} = \{\neg o_{31}, o_{33}, o_{43}\}$  representing the second rule of the DS shown in Figure 1a. The original approach of [17] would instead compute an AXp defining the narrowest intervals for features 3 and 4, representing a rule:  $\langle \text{IF } 2.60 \leq \text{“petal.length”} < 4.75 \wedge \text{“petal.width”} < 1.45 \text{ THEN class} = \text{“versicolor”} \rangle$ , which is far less general than  $\mathcal{Y}$ . ◻

A possible rule extraction procedure is outlined in Algorithm 1. (Please ignore line 3 for now; feature sorting is described in Section 4.2). The input BT model  $\mathfrak{T}$  is encoded into MaxSAT by applying the approach of [17]. Given an instance  $\mathbf{v} \in \mathbb{F}$ , the initial set of literals  $\mathcal{Y} = \{\tilde{o}_{ij} \mid i \in \mathcal{F}, j \in [h_i]\}$  is created. Note that any feature  $i \in \mathcal{F}$  unused in the BT  $\mathfrak{T}$  is excluded from  $\mathcal{Y}$ . The rest of the procedure implements the standard deletion-based AXp extraction [20], i.e. it iterates through all literals in  $\mathcal{Y}$  one by one, and checks which of the them can be safely removed such that entailment (3) still holds.

► **Example 6.** Consider our running example model and instance  $\mathbf{v}_2 \in e_2$  from Table 1 predicted as “*virginica*” by the BT  $\mathfrak{T}$ . Given the thresholds for features 3 and 4 in Example 3, set  $\mathcal{Y}$  is initialized to  $\{\neg o_{31}, \neg o_{32}, \neg o_{33}, \neg o_{41}, \neg o_{42}, \neg o_{43}\}$ . The other two features are excluded from  $\mathcal{Y}$  since they are irrelevant to the classification function in  $\mathfrak{T}$ . Applying Algorithm 1 results in extracting a subset-minimal generalized AXp  $\mathcal{Y} = \{\neg o_{33}\}$ , which represents the rule  $\langle \text{IF petal.length} \geq 4.95 \text{ THEN class} = \text{“virginica”} \rangle$ . ◻

► **Remark 7.** Algorithm 1 relies on deciding whether formula (3) holds for each feature in explanation  $\mathcal{Y}$ . Here, this is done by means of a series of incremental core-guided MaxSAT oracle calls [19, 17]. One may wonder whether or not incomplete *anytime* MaxSAT solving [31, 35, 2, 32] can be applied in this setting. Although this may look plausible at first glance, time-restricted anytime MaxSAT algorithms can only *over-approximate* exact MaxSAT solutions while (3) holds *if and only if* the exact value of the objective function is negative. Therefore, an over-approximation of a MaxSAT solution is *never able* to prove the validity of (3) and so none of the features being tested can be discarded in the case of incomplete MaxSAT algorithms, which defies the purpose of Algorithm 1.

## 4.2 Boosted Tree Compilation

As mentioned above, generalized AXps can be seen as general decision rules that can be applied to an enormous number of instances. Therefore, it makes little sense to extract such rules for each instance in the feature space  $\mathbb{F}$ . Instead, one can devise an on-demand



■ **Algorithm 2** Compile a BT into a DS.

---

**Function:**  $\text{Compile}(\mathfrak{T}, \tau, \mathcal{C})$

**Input:**  $\mathfrak{T}$ : BT defining  $\tau(\mathbf{x})$ ,  $\tau$ : Classification function in  $\mathfrak{T}$ ,  $\mathcal{C}$ : Coverage set

**Output:**  $\mathcal{R}$ : Set of Rules

```

1:  $\mathcal{R} \leftarrow \emptyset$ 
2:  $\mathcal{C}_u \leftarrow \mathcal{C}$ 
3: while  $\mathcal{C}_u \neq \emptyset$  do
4:    $\mathbf{v} \leftarrow \text{GetInst}(\mathcal{C}_u)$ 
5:    $\mathcal{Y} \leftarrow \text{RuleExtract}(\mathfrak{T}, \mathbf{v}, c = \tau(\mathbf{v}), \mathcal{C}_u)$ 
6:    $\mathcal{C}_c \leftarrow \text{GetCover}(\mathcal{Y}, \mathcal{C}_u)$ 
7:    $\mathcal{C}_u \leftarrow \mathcal{C}_u \setminus \mathcal{C}_c$ 
8:    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{Y}$ 
9: return  $\mathcal{R}$ 

```

---

*compilation* process, i.e. given a *yet uncovered* instance  $\mathbf{v} \in \mathbb{F}$ , we can apply Algorithm 1 to extract a rule covering  $\mathbf{v}$  (and some other instances). Clearly, *exhaustive* compilation of a BT, i.e. if the target is to cover all the instances in  $\mathbb{F}$  with generalized AXps of the BT, is computationally expensive given that AXp extraction for tree ensembles is hard for  $D^P$  [25]. This can also lead to the large size of the resulting DSes making them hard to interpret. In practice, *local* compilation aiming at capturing the behavior of the BT on the training data only, is sufficient to generate a DS, which is both accurate and interpretable.

The proposed approach to compiling a BT  $\mathfrak{T}$  into a DS  $\mathcal{R}$  is shown in Algorithm 2. We initialize the set  $\mathcal{C}_u$  of currently uncovered instances to be equal to  $\mathcal{C}$ , i.e. the set of examples we wish to cover. The algorithm represents a loop generating rules until the set of computed rules  $\mathcal{R}$  covers all instances in coverage set data  $\mathcal{C}$ , i.e. until there is no uncovered instances in  $\mathcal{C}$ . Each iteration of the algorithm selects an instance  $\mathbf{v}$  from  $\mathcal{C}_u$ . Afterwards, a generalized AXp  $\mathcal{Y}$  for the prediction  $c = \tau(\mathbf{v})$  by the BT  $\mathfrak{T}$  (recall that  $\mathfrak{T}$  is meant to compute classification function  $\tau(\mathbf{x})$ ) is extracted by invoking Algorithm 1. The iteration proceeds by updating the set of rules  $\mathcal{R}$  and the set of uncovered instances  $\mathcal{C}_u$ . The algorithm terminates when all the instances in the coverage set  $\mathcal{C}$  are covered and returns a compiled DS  $\mathcal{R}$ .

► **Proposition 8.** *Let  $\mathfrak{T}$  be a BT and  $\mathcal{R}$  be a DS returned by Algorithm 2 for  $\mathfrak{T}$ . Then  $\mathcal{R} \equiv \mathfrak{T}$  with respect to  $\mathcal{C}$ .*

We consider two usages of the algorithm: for *exhaustive compilation* the coverage set  $\mathcal{C} = \mathbb{F}$  is all possible feature combinations (in practice we model this coverage set implicitly, rather than in its explicit exponential sized form), and for *training set compilation* where  $\mathcal{C} = \mathcal{E}$  is the training set. Based on the properties of prime implicants, Proposition 8 states that as a generalized AXp  $\mathcal{Y} \in \mathcal{R}$  is a formal explanation for a prediction made by BT  $\mathfrak{T}$ , a compiled DS captures the semantics of the original model  $\mathfrak{T}$  on *coverage set*  $\mathcal{C}$ , assuming everything else is a *don't care*. Furthermore, if the process is applied subject to coverage set  $\mathcal{C} = \mathbb{F}$ , i.e. when we target the entire feature space  $\mathbb{F}$ , then  $\mathcal{R}$  and  $\mathfrak{T}$  behave identically, i.e. they compute the same classification function  $\tau(\mathbf{x})$ .

► **Corollary 9.** *Let Algorithm 2 return a DS  $\mathcal{R}$  for a BT  $\mathfrak{T}$ . Then there is no instance in feature space  $\mathbb{F}$  covered by two distinct rules  $\mathcal{Y}_1, \mathcal{Y}_2 \in \mathcal{R}$  predicting inconsistent classes  $c_1 \neq c_2$ .*

As each generalized AXp for  $\mathfrak{T}$  represents a prime implicant of the decision function  $\tau(\mathbf{x})$  computed over literals  $\tilde{o}_{ij}$ , the above corollary claims that there are no overlapping rules in the result DS  $\mathcal{R}$ . This contrasts with other modern approaches to DS inference, where rule overlap is known to be a problem [29, 22]. Note that this approach still suffers from another common issue of DS models: namely, if DS  $\mathcal{R}$  is computed for the training data  $\mathcal{E}$ , there may still be instances in  $\mathbb{F}$  uncovered by  $\mathcal{R}$ .

► **Example 10.** Consider the running example BT model shown in Figure 1b. Its compiled DS representation computed by Algorithm 2 is shown in Figure 1a. Observe that there is no rule overlap in the DS computed. In fact, as the DS is computed by taking into account feature space  $\mathbb{F}$ , it computes the same classification function as the original BT model. ◻

**Feature Sorting.** Intuitively, how general and hence how applicable a rule is depends on how frequently the features used in it appear in the training data  $\mathcal{E}$  labeled with the target class. Thus, a simple heuristic to apply when extracting a rule for prediction  $c = \tau(\mathbf{v})$  is to sort the initial state of  $\mathcal{Y} = \{\tilde{o}_{ij} \mid i \in \mathcal{F}, j \in [h_i]\}$  based on how frequently the corresponding literals  $\tilde{o}_{ij}$  apply in examples  $\mathcal{E}$  labeled with  $c$ . This feature sorting represented by line 3 in Algorithm 1 in practice (according to our experiments) results in significantly more general rules and so overall smaller DSes.

**Anytime Property.** Most widely used reasoning-based algorithms to infer DSes provide a solution only if the computation is completed; otherwise, no decision set is reported. In contrast to these, the proposed approach is an *anytime* algorithm, i.e. it can return a *valid* DS  $\mathcal{R}$  even though the compilation process is interrupted before all the coverage set instances  $\mathcal{C}$  are covered. Furthermore, it can generate a more comprehensive DS  $\mathcal{R}$ , which covers more instances as it keeps going, i.e. after we have covered  $\mathcal{C} \subseteq \mathbb{F}$  we can continue running the algorithm for the (unseen) instances of  $\mathbb{F}$ .

### 4.3 Post-Hoc Model Reduction

The compiled DS  $\mathcal{R}$  can be large (in terms of either the number of rules or the total number of literals) since each generalized AXp  $\mathcal{Y} \in \mathcal{R}$  may need a significant number of literals to explain a prediction made by BT  $\mathfrak{T}$ , or/and many rules are required to explain all instances of  $\mathcal{C}$ . Once the target DS is obtained, we can apply post-hoc heuristic methods for reducing its size and so making it more interpretable. The methods below are in a way inspired by the optimization problems studied in [18, 50]. Although these ideas are applicable to any DS inference method once the result model is devised, they do not look necessary for standard DS inference algorithms as they minimize the model while training. On the contrary, no minimization is applied in the rule enumeration process described above and so post-hoc model reduction plays a vital role in our approach to reduce the size of final DS models.

**Reducing the Number of Rules.** Given a set of rules  $\mathcal{R}$ , we can compute a minimum subset  $\mathcal{R}^* \subseteq \mathcal{R}$  that is still equivalent to the BT  $\mathfrak{T}$  wrt. the coverage set  $\mathcal{C}$  using discrete optimization, e.g. integer-linear programming (ILP). Concretely, the approach aims at selecting the smallest-size subset  $\mathcal{R}^* \subseteq \mathcal{R}$  that covers all instances in  $\mathcal{C}$ , where  $\mathcal{R}$  is the compiled DS from  $\mathfrak{T}$ . Here, the size of  $\mathcal{R}^*$  is measured as the total number of literals used. This can be done by solving the following *set cover problem* [28]. Namely, for each rule  $\mathcal{Y}_j \in \mathcal{R}$ , we introduce a Boolean variable  $u_j$  such that  $u_j = 1$  iff  $\mathcal{Y}_j$  is included in  $\mathcal{R}^*$ .

Additionally, a Boolean variable  $y_{ij}$  is used to indicate that  $\mathcal{Y}_j$  covers  $e_i \in \mathcal{C}$ . As a result, the weighted set cover problem for minimizing the total number of literals used is as follows:

$$\text{minimize} \quad \sum_{j=1}^{|\mathcal{R}|} (|\mathcal{Y}_j| + 1) \cdot u_j \quad (4)$$

$$\text{subject to} \quad \forall_{i \in [n]} \sum_{j=1}^{|\mathcal{R}|} y_{ij} \cdot u_j \geq 1 \quad (5)$$

**Reducing the Number of Literals.** Additionally, one can minimize the total number of literals used in the rules of  $\mathcal{R}$ . Given a rule  $\mathcal{Y} \in \mathcal{R}$ , this can be done either lexicographically by maximizing rule accuracy followed by size minimization, or by optimizing both, or trading off misclassifications for rule size – in either case, a single MaxSAT call per rule to minimize can be made. The intuition is that if a rule  $\mathcal{Y}$  misclassifies  $k$  instances then its optimized version  $\mathcal{Y}^* \subseteq \mathcal{Y}$  should not result in many more misclassifications on training data  $\mathcal{E}$ . Recall that a rule misclassifies an instance  $\mathbf{v}_k \in \mathcal{C}$  if it matches  $\mathbf{v}_k$  but assigns it to a wrong class.

Inspired by [18], we introduce a Boolean variable  $p_k$ , which is true iff rule  $\mathcal{Y}$  covers  $\mathbf{v}_k$  – this holds if  $\mathcal{Y}$  does not use any literals incompatible with  $\mathbf{v}_k$ . If  $\mathcal{Y}_{\mathbf{v}_k} = \{\tilde{o}_{ij} \mid i \in \mathcal{F}, j \in [h_i]\}$  are all the literals compatible with  $\mathbf{v}_k$  then this can be modeled with constraints

$$\forall_{k \in [|\mathcal{C}|]} p_k \leftrightarrow \bigwedge_{l \in \mathcal{Y} \setminus \mathcal{Y}_{\mathbf{v}_k}} \neg l \quad (6)$$

Furthermore, let rule  $\mathcal{Y}$  predict  $c \in \mathcal{K}$  and let  $\mathcal{C}_\ominus \subseteq \mathcal{C}$  contain all instances labeled with any other class. Thus, we can apply the objective below when minimizing rule  $\mathcal{Y}$ :

$$\sum_{l \in \mathcal{Y}} l + \sum_{k \in [|\mathcal{C}_\ominus|]} W \cdot p_k \quad (7)$$

If  $W$  is large enough, say  $|\mathcal{C}| + 1$ , this lexicographically minimizes misclassifications and then literals. If  $W$  is small, e.g.  $1/\lambda \cdot |\mathcal{C}|$ , this trades off  $\lambda \cdot |\mathcal{C}|$  misclassifications for one literal.

## 5 Experimental Results

This section compares the proposed approach with the state-of-the-art DS learning algorithms on a variety of publicly available datasets in terms of accuracy, scalability, model and explanation size. The experiments are performed on an Intel Xeon 8260 CPU running Ubuntu 20.04.2 LTS, with the time limit of 3600s and the memory limit of 8GByte. Our experiments contain two parts, namely, exhaustive BT compilation and training-set BT compilation.

**Prototype implementation.** A prototype of the compilation-based approach to generating DSes was developed as a set of Python scripts using  $\mathcal{C} = \mathcal{E}$ , hereinafter referred to as *cpl*. The implementation of BT compilation exploits [17] and, therefore, makes use of the RC2 MaxSAT solver [19].<sup>4</sup> The BTs to be compiled are computed by XGBoost [5]; the number of trees per class in a BT model is 50 and the maximum depth of each tree is 3. Post-hoc

<sup>4</sup> Real weights in the objective function are not conventionally supported by MaxSAT solvers; the only other solver to support real weights besides RC2 is LMHS [47].

literal reduction is done again with RC2 [19]. Let  $cpl_l$  denote the implementation applying lexicographic optimization while  $cpl_{l_{\lambda_1}}$  trades off model accuracy for the number of literals used, with  $\lambda_1 = 0.005$ . Let  $cpl_r$  denote the implementation with post-hoc rule reduction applied using the Gurobi ILP solver [14]. The configuration with both post-hoc lexicographic optimization and rule reduction is denoted  $cpl_{lr}$ . Finally, the proposed approach applying exhaustive compilation  $\mathcal{C} = \mathbb{F}$  is referred to as  $cpl_f$ .

**Competition.** Our approach is compared against: *twostg* a two-stage MaxSAT approach [18] for DSes perfectly accurate on the training data; *opt* another MaxSAT approach [50] for perfectly accurate DSes;  $sp_{\lambda_1}$  a sparse alternative to *opt* by the same authors (with  $\lambda_1 = 0.005$ ) optimizing like  $cpl_{l_{\lambda_1}}$ ; *iml<sub>1</sub>* and *iml<sub>16</sub>* using MaxSAT-based IMLI [12] to minimize the number of literals given a predefined number of rules (we use 1 or 16); *ids* a state-of-the-art approach [29] based on smooth local search;<sup>5</sup> *ripper* a popular heuristic DS algorithm RIPPER [8]; and CN2 (referred to as *cn2*) another heuristic algorithm [7, 6].<sup>6</sup>

**Datasets.** For the evaluation, 59 publicly available datasets from UCI Machine Learning Repository [9] and Penn Machine Learning Benchmarks [41] are considered. We apply 5-fold cross validation, resulting in 295 pairs of training and test (unseen) data. For the sake of a fair comparison, the datasets used are preprocessed so that each original feature  $i \in \mathcal{F}$  is replaced with a number of non-intersecting feature intervals  $x_i < d_{ij}$  defined by the XGBoost model (see Section 2). This guarantees that all competitors tackle the same problem instances.

## 5.1 Exhaustive BT Compilation

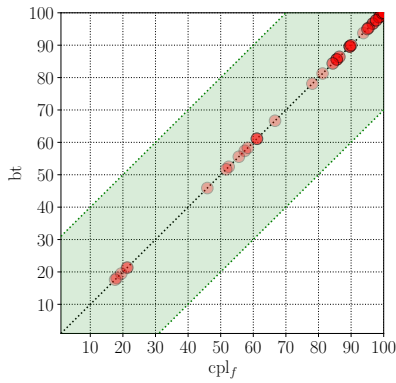
The first experiment compares exhaustive compilation, where  $\mathcal{C} = \mathbb{F}$  is the entire feature space. This is impractical except for 6 small benchmarks.

**Results.** Here we compare  $cpl_f$  with the competition in terms of accuracy, the total number of literals used and explanation size. We present the results as cactus plots showing the number of datasets that e.g. reach a certain accuracy, or finish in a certain runtime, for each method. These experimental results are shown in Figures 2 and 3 as well as the average results across folds are described in Table 2 where only the results of the datasets *completely* solved by compared competitors are presented. Note that  $cpl_f$  is nowhere near as scalable as the approaches described in the later experiments, but it is the *most accurate* approach to creating DSes we are aware of.

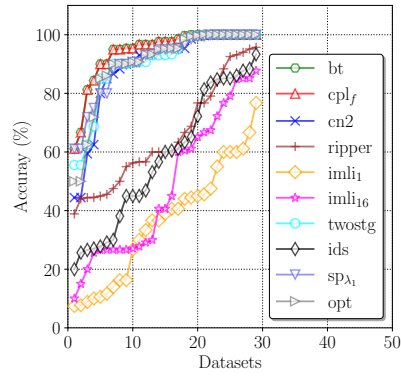
**Test accuracy.** An instance is considered misclassified if either there exists a rule of a wrong class that covers it, or it is not covered by any rule of the correct class. Thus, the test accuracy in this paper is calculated as  $\frac{n-g}{n}$ , where  $n$  is the total number of instances in the test data and  $g$  is the total number of misclassified instances. If an approach fails to train a model within the time limit, we assume its accuracy to be 0% for this dataset.

<sup>5</sup> Since the original implementation performs poorly [22], here we consider the new implementation of IDS [10], which is claimed to be orders of magnitude faster than the original implementation.

<sup>6</sup> Note that since RIPPER and IMLI compute a single class only given the training data, both of these competitors are augmented with a default rule predicting a class (1) different from the target class and (2) represented by the majority of training instances. Other algorithms, including our approach, incorporate a default rule that assigns a class based on the majority class in the training instances.

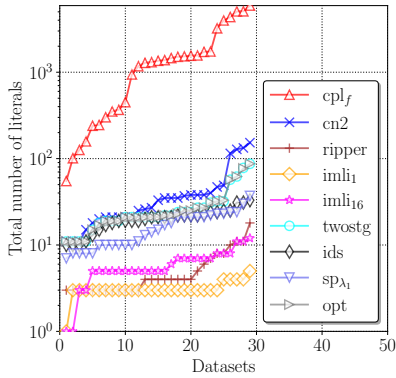


(a) Comparison with BT.

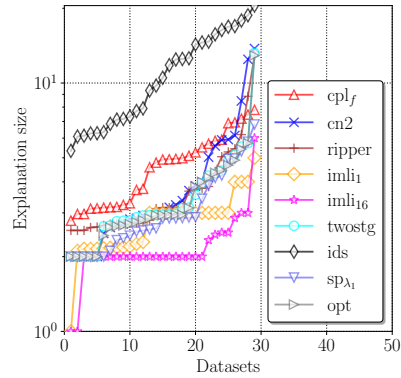


(b) Comparison with the others.

■ **Figure 2** Accuracy of exhaustive compilation. The *standard* interpretation of cactus plots is assumed, i.e. a plot sorts the datapoints for each method by the  $y$ -axis value, and then shows them in increasing order independently of other methods. Thus, the order of datasets/folds differs for different methods. Also, the order of datasets for the same method differs in different subplots.



(a) Number of literals used.



(b) Explanation size.

■ **Figure 3** Succinctness of exhaustive compilation.

As can be seen in Figure 2b and Table 2, the best accuracy is achieved by BTs and  $cpl_f$ . In fact, these models share the same accuracy (this is also confirmed in Figure 2a), which should not come as a surprise given that  $cpl_f$  replicates the behavior of the BT in the entire feature space  $\mathbb{F}$  (see Proposition 8).

**Model Complexity.** In general, complexity of a DS model can be measured by the total number of literals used in this DS. The total number of literals used in DS models is compared in Figure 3a and Table 2. Though the accuracy of DSEs trained by  $cpl_f$  outperforms the other competitors, these models are significantly larger, which is no surprise given that  $cpl_f$  computes many more rules with no post-hoc reduction applied.

**Explanation size.** Explanation size is defined as the number of literals required to explain an instance.<sup>7</sup> This is arguably more important than the model size, since it defines “how hard” it is to understand an individual explanation. A small DS model tends to provide

<sup>7</sup> See [51] for details.

■ **Table 2** Accuracy, number of literals used, and explanation size across folds.

Approach	Dataset					
	cardiotocography	hayes-roth	iris	new-thyroid	orbit	zoo
	Accuracy (%)					
<b>bt</b>	<b>100.0</b>	<b>84.38</b>	<b>96.0</b>	<b>96.74</b>	<b>99.66</b>	<b>96.0</b>
<i>cpl<sub>f</sub></i>	<b>100.0</b>	<b>84.38</b>	<b>96.0</b>	<b>96.74</b>	<b>99.66</b>	<b>96.0</b>
<i>sp<sub>λ<sub>1</sub></sub></i>	100.0	73.44	94.0	91.63	99.43	89.05
<i>opt</i>	100.0	70.63	93.33	91.63	99.54	93.05
<i>twostg</i>	100.0	71.25	92.67	92.09	99.54	91.1
<i>cn2</i>	100.0	62.5	92.67	93.02	99.54	89.1
<i>ripper</i>	45.3	66.25	57.33	80.93	94.11	60.33
<i>ids</i>	27.23	43.75	58.67	76.28	85.29	40.62
<i>imli<sub>16</sub></i>	27.23	38.75	25.34	69.77	70.55	43.33
<i>imli<sub>1</sub></i>	45.3	39.37	32.67	26.98	8.93	60.33
	Number of literals used					
<i>cpl<sub>f</sub></i>	3120.0	76.0	214.0	3614.2	729.8	1422
<i>sp<sub>λ<sub>1</sub></sub></i>	21.0	33.5	9.0	15.4	10.0	23.2
<i>opt</i>	21.0	63.6	19.4	23.0	11.8	30.0
<i>twostg</i>	21.0	64.2	19.8	22.6	11.8	29.8
<i>cn2</i>	21.0	116.2	27.2	36.6	13.2	40.8
<i>ripper</i>	<b>3.0</b>	12.8	5.0	8.2	4.0	<b>3.0</b>
<i>ids</i>	21.0	21.6	19.8	20.0	25.0	14.2
<i>imli<sub>16</sub></i>	5.0	<b>2.2</b>	7.4	7.4	6.4	5.0
<i>imli<sub>1</sub></i>	<b>3.0</b>	<b>2.2</b>	<b>3.0</b>	<b>4.2</b>	<b>3.0</b>	<b>3.0</b>
	Explanation size					
<i>cpl<sub>f</sub></i>	7.26	3.76	3.02	4.9	3.18	5.4
<i>sp<sub>λ<sub>1</sub></sub></i>	<b>2.0</b>	6.31	2.45	4.13	2.86	3.64
<i>opt</i>	<b>2.0</b>	5.41	2.76	4.3	2.94	2.96
<i>twostg</i>	<b>2.0</b>	5.4	2.87	4.23	2.94	3.33
<i>cn2</i>	<b>2.0</b>	6.94	3.02	4.47	3.02	4.05
<i>ripper</i>	2.73	10.15	4.3	4.3	3.15	2.59
<i>ids</i>	16.08	18.23	13.06	7.74	6.23	9.28
<i>imli<sub>16</sub></i>	<b>2.0</b>	<b>2.2</b>	<b>2.1</b>	<b>1.97</b>	<b>2.8</b>	2.46
<i>imli<sub>1</sub></i>	2.18	<b>2.2</b>	3.0	4.0	3.0	<b>2.2</b>

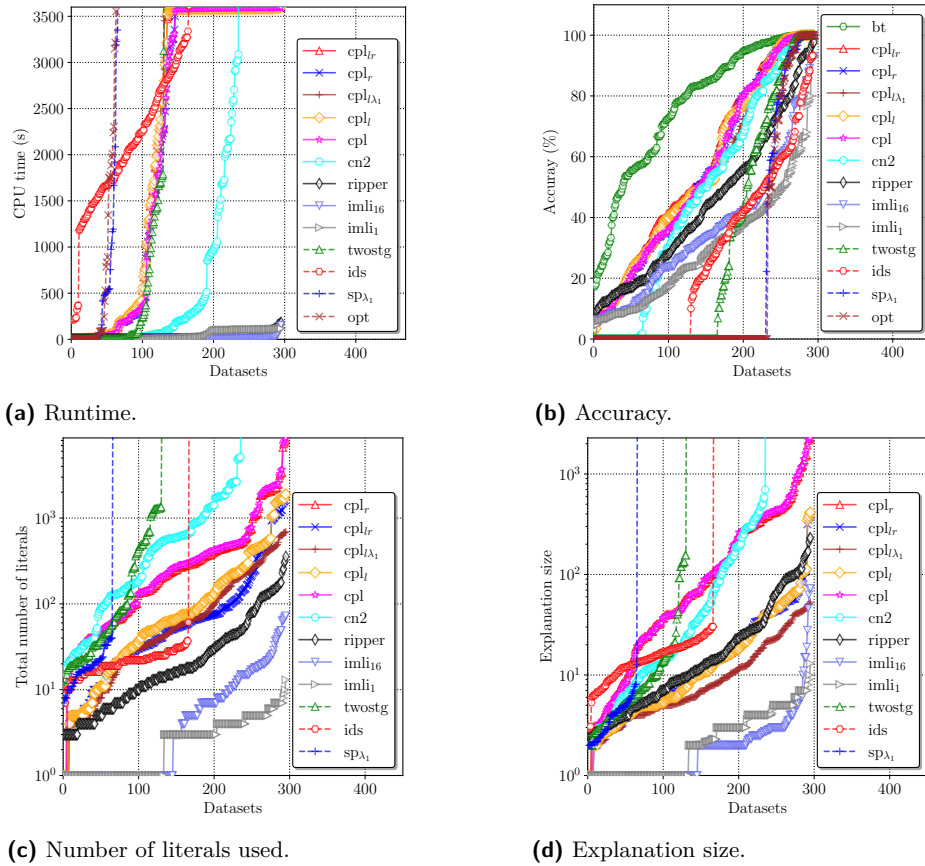
compact explanations but it is not always accurate. As can be seen in Figure 3b and Table 2 and similar to the total number of literals used in DSes, *cpl<sub>f</sub>* requires more literals to explain an instance than all competitors except *ids*.

A crucial observation to make here is that we test explanation size for each of the test instances available. Although test data are meant to extrapolate the overall unseen data, such approximation of the unseen feature space is not ideal. As a result, there may be numerous instances in  $\mathbb{F}$  *uncovered* by all the approaches but *cpl<sub>f</sub>*, in which case it will be the *only* approach providing a user not only with a prediction but also with a succinct explanation of the prediction made.

## 5.2 BT Compilation Targeting Training Data

Compilation to cover the training set  $\mathcal{C} = \mathcal{E}$  is much more efficient, and the main usage we expect of our algorithms.

**Scalability.** Figure 4a depicts scalability of all selected algorithms on the 295 considered datasets. Note that runtime of our approach includes BT training time. The best performance is demonstrated by the proposed implementation, i.e. *cpl* and *cpl<sub>\*</sub>*,  $*$   $\in \{l, r, lr, l\lambda_1\}$ , where all selected datasets are solved within the time limit. This is not surprising since the approach is an anytime algorithm that can always return a valid DS. As for other competitors, the heuristic method *ripper* and the MaxSAT approaches *imli<sub>1</sub>* as well as *imli<sub>16</sub>* also solve all considered datasets. Next is the heuristic algorithm *cn2*, where 235 datasets are solved



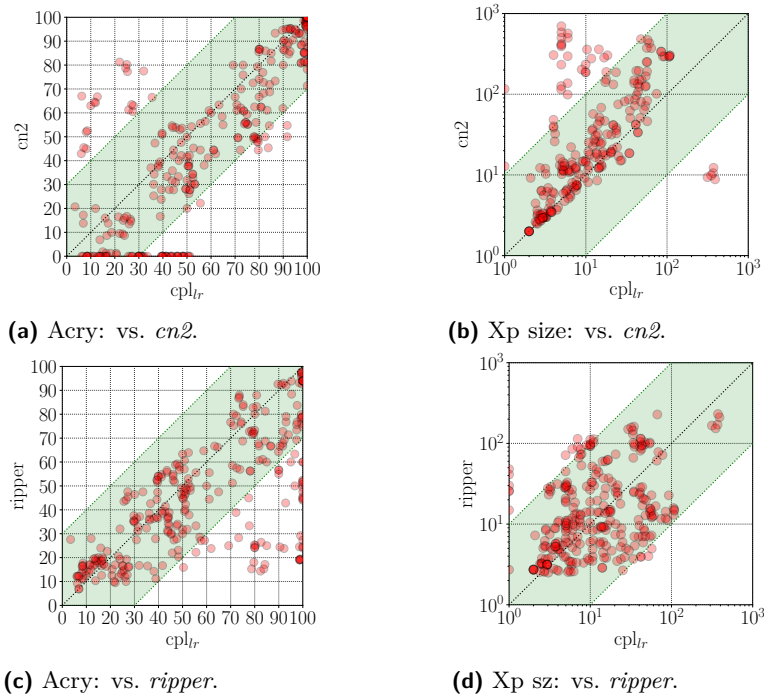
■ **Figure 4** Summary of experimental results when the competitors aim at training a DS given training data  $\mathcal{E}$  (i.e.  $\mathcal{C} = \mathcal{E}$ ).

within the 3600s time limit. Followed by *ids*, which solves 166 considered datasets. The two-stage MaxSAT approach *twostg* successfully addresses 130 datasets, while the other MaxSAT algorithm for perfect decision sets *opt* and its sparse alternative *sp <sub>$\lambda_1$</sub>*  solve 65 and 63 datasets respectively.

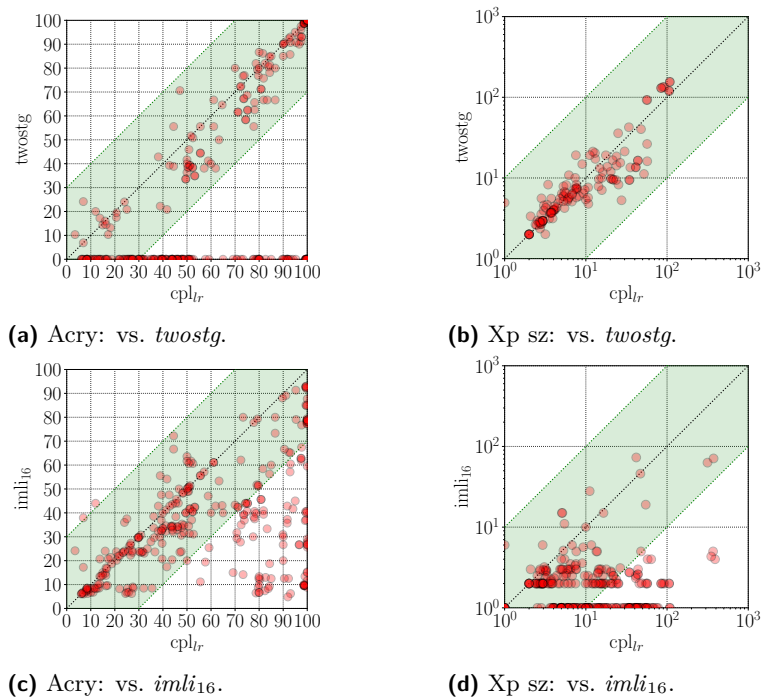
**Test Accuracy.** The accuracy among the selected approaches is shown in Figure 4b. The average accuracy among all selected datasets for BTs is 77.34%, beating all DS approaches. The highest accuracy among DSEs is achieved by all the configurations of the proposed approach, i.e. *cpl* and *cpl<sub>\*</sub>*, where the average accuracy ranges from 54.01% (*cpl <sub>$\lambda_1$</sub>* ) to 57.49% (*cpl<sub>lr</sub>*).<sup>8</sup> Unsurprisingly, the accuracy in *cpl <sub>$\lambda_1$</sub>*  is lower than the other configurations since *cpl <sub>$\lambda_1$</sub>*  trades off training accuracy on the number of literals in the computation process.

Next most accurate are the heuristic methods *cn2* (48.03%) followed by *ripper* (44.81%). The average accuracy of *imli<sub>16</sub>* and *imli<sub>1</sub>* is 35.47% and 29.7% respectively, while the average accuracy of *twostg* is 29.6% and *ids* is 26.78. Finally, the worst accuracy is demonstrated by *sp <sub>$\lambda_1$</sub>*  and *opt* (18.84% and 18.27% on average respectively) as these tools fail to provide prediction information for many datasets within the time limit. We will omit further discussion of *sp* and *opt <sub>$\lambda_1$</sub>*  since they solve so few datasets.

<sup>8</sup> Note that most datasets we used represent non-binary classification. Also, DSEs are not to be compared with BTs. As Figure 4b shows (and as our work aims to demonstrate), our approach outperforms the state-of-the-art DS inference methods in terms of accuracy.



■ **Figure 5** Comparison of *cpl<sub>r</sub>* vs. *cn2* and *ripper* in terms of accuracy and explanation size.



■ **Figure 6** *cpl<sub>r</sub>* vs. *imli<sub>16</sub>* and *twostg* in terms of accuracy and explanation size.



**Model Complexity.** Figure 4c illustrates the comparison among selected approaches regarding the total number of literals used in each DS solution. The average number of literals are in order: *imli*<sub>1</sub> (2.77), *imli*<sub>16</sub> (8.26), *ids* (21.14), *ripper* (38.47), *cpl*<sub>l $\lambda$ 1</sub> (118.47), *cpl*<sub>l $r$</sub>  (157.53), *cpl*<sub>l</sub> (213.27), *twostg* (265.98), *cpl*<sub>r</sub> (584.39), *cpl* (620.82), *cn2* (700.49). Clearly, rule reduction and literal reduction can significantly reduce the size of the model without significantly affecting accuracy. Note how our approaches while significantly larger than the least accurate competitors, are significantly smaller than the most accurate competitor *cn2*.

**Explanation Size.** Figure 4d shows the explanation size for each competitor. The average explanation sizes are in order: *imli*<sub>1</sub> (2.61), *imli*<sub>16</sub> (3.00), *cpl*<sub>l $\lambda$ 1</sub> (12.14), *ids* (15.28), *twostg* (17.5), *cpl*<sub>l $r$</sub>  (25.34), *cpl*<sub>l</sub> (26.18), *ripper* (29.08), *cn2* (81.93), *cpl*<sub>r</sub> (234.46), *cpl* (240.88). Figure 4d demonstrates that post-hoc literal reduction not only helps decrease the number of literals required to explain DS models, but also enables DSes to remain accurate, whereas rule reduction does not contribute to smaller explanations. With literal reduction applied our approaches are very competitive in terms of explanation size.

**Detailed Comparison.** While cactus plots allow us to compare many methods over a large suite of benchmarks, they do not allow direct comparison on individual benchmarks. We provide a detailed comparison of *cpl*<sub>l $r$</sub>  versus other decision set inference approaches in Figures 5 and 6, including *cn2*, *ripper*, *twostg*, and *imli*<sub>16</sub>.<sup>9</sup> The scatter plots depicting explanation size are obtained for the datasets solvable by both competitors. Note that *cpl*<sub>l $r$</sub>  can generate more accurate DSes than the competitors. Also observe that the explanation size of DSes computed by *cpl*<sub>l $r$</sub>  is smaller than *cn2* and comparable with *twostg*. Although the explanation size of DSes in *cpl*<sub>l $r$</sub>  is larger than *ripper* and *imli*<sub>16</sub>, the two approaches are less interpretable as they compute DSes representing only one class.

**Summary.** The experimental results were performed on various datasets, demonstrating that our approach computes DSes that outperform the state-of-the-art competitors in terms of accuracy and yield comparable explanation size to them.

## 6 Conclusions

This paper introduced a novel *anytime* approach to generating decision sets by means of on-demand extraction of generalized abductive explanations for boosted tree models. It can be used for exhaustive compilation of a BT model wrt. the entire feature space, or target a set of training instances. Augmented by a number of post-hoc model reduction techniques, the approach is shown to compute decision sets that are more accurate than decision sets computed by the state-of-the-art algorithms and comparable with them in terms of explanation size.

As the proposed approach targets generating a decision set by compiling a BT, a natural line of future work is to extend the proposed approach to compile BTs into the other interpretable models, i.e. decision trees and decision lists, making use of AXp extraction for BTs. Additionally, another future work is to apply AXp extraction to compile other accurate black box models, e.g. neural networks, into decision sets.

<sup>9</sup> The average results across the folds are given in the appendix.

---

**References**

---

- 1 ACM. Fathers of the deep learning revolution receive ACM A.M. Turing award. <http://tiny.cc/9plzpz>, 2018.
- 2 Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. Core-boosted linear search for incomplete MaxSAT. In *CPAIOR*, pages 39–56, 2019.
- 3 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2021.
- 4 Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
- 5 Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *KDD*, pages 785–794, 2016.
- 6 Peter Clark and Robin Boswell. Rule induction with CN2: some recent improvements. In *EWSL*, pages 151–163, 1991.
- 7 Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.
- 8 William W. Cohen. Fast effective rule induction. In *ICML*, pages 115–123, 1995.
- 9 Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL: <http://archive.ics.uci.edu/ml>.
- 10 Jiri Filip and Tomas Kliegr. Pyids-python implementation of interpretable decision sets algorithm by lakkaraaju et al, 2016. In *RuleML+ RR*, 2019.
- 11 Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. In *CEx@AI\*IA*, volume 2071 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.
- 12 Bishwamittra Ghosh and Kuldeep S. Meel. IMLI: an incremental framework for MaxSAT-based learning of interpretable classification rules. In *AIES*, pages 203–210. ACM, 2019.
- 13 Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *Int. J. Comput. Vis.*, 129(6):1789–1819, 2021.
- 14 Gurobi Optimization. Gurobi optimizer reference manual, 2022. URL: <http://www.gurobi.com/>.
- 15 Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Inf. Process. Lett.*, 5(1):15–17, 1976.
- 16 Alexey Ignatiev. Towards trustworthy explainable AI. In *IJCAI*, pages 5154–5158, 2020. doi:10.24963/ijcai.2020/726.
- 17 Alexey Ignatiev, Yacine Izza, Peter J. Stuckey, and João Marques-Silva. Using MaxSAT for efficient explanations of tree ensembles. In *AAAI*, pages 3776–3785, 2022.
- 18 Alexey Ignatiev, Edward Lam, Peter J. Stuckey, and Joao Marques-Silva. A scalable two stage approach to computing optimal decision sets. In *AAAI*, pages 3806–3814, 2021.
- 19 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. RC2: an efficient MaxSAT solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019.
- 20 Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. In *AAAI*, pages 1511–1519, 2019. doi:10.1609/aaai.v33i01.33011511.
- 21 Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. On validating, repairing and refining heuristic ML explanations. *CoRR*, abs/1907.02509, 2019. arXiv:1907.02509.
- 22 Alexey Ignatiev, Filipe Pereira, Nina Narodytska, and João Marques-Silva. A SAT-based approach to learn explainable decision sets. In *IJCAR*, pages 627–645, 2018.
- 23 Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. On tackling explanation redundancy in decision trees. *J. Artif. Intell. Res.*, 75:261–321, 2022.
- 24 Yacine Izza, Alexey Ignatiev, Peter J. Stuckey, and Joao Marques-Silva. Delivering inflated explanations. *CoRR*, abs/2306.15272, 2023.
- 25 Yacine Izza and Joao Marques-Silva. On explaining random forests with SAT. In *IJCAI*, July 2021.

- 26 Anil P. Kamath, Narendra Karmarkar, K. G. Ramakrishnan, and Mauricio G. C. Resende. A continuous approach to inductive inference. *Math. Program.*, 57:215–238, 1992.
- 27 Kentaro Kanamori, Takuya Takagi, Ken Kobayashi, Yuichi Ike, Kento Uemura, and Hiroki Arimura. Ordered counterfactual explanation by mixed-integer linear optimization. In *AAAI*, pages 11564–11574. AAAI Press, 2021.
- 28 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.
- 29 Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *KDD*, pages 1675–1684, 2016.
- 30 Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.
- 31 Zhendong Lei and Shaowei Cai. Solving (weighted) partial maxsat by dynamic local search for SAT. In *IJCAI*, pages 1346–1352, 2018.
- 32 Zhendong Lei and Shaowei Cai. Nudist: An efficient local search algorithm for (weighted) partial maxsat. *Comput. J.*, 63(9):1321–1337, 2020.
- 33 Zachary C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, 2018. doi:10.1145/3233231.
- 34 Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NeurIPS*, pages 4765–4774, 2017. URL: <https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html>.
- 35 Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. *Artif. Intell.*, 243:26–44, 2017.
- 36 Dmitry Malioutov and Kuldeep S. Meel. MLIC: A MaxSAT-based framework for learning interpretable classification rules. In *CP*, pages 312–327, 2018.
- 37 Joao Marques-Silva and Alexey Ignatiev. Delivering trustworthy AI through formal XAI. In *AAAI*, pages 12342–12350, 2022.
- 38 Hayden McTavish, Chudi Zhong, Reto Achermann, Ilias Karimalis, Jacques Chen, Cynthia Rudin, and Margo I. Seltzer. Fast sparse decision tree optimization via reference ensembles. In *AAAI*, pages 9604–9613, 2022.
- 39 Ryszard S Michalski. On the quasi-minimal solution of the general covering problem. In *International Symposium on Information Processing*, pages 125–128, 1969.
- 40 Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and João Marques-Silva. Learning optimal decision trees with SAT. In *IJCAI*, pages 1362–1368, 2018.
- 41 Randal S. Olson, William G. La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Min.*, 10(1):36:1–36:13, 2017.
- 42 Axel Parmentier and Thibaut Vidal. Optimal counterfactual explanations in tree ensembles. In *ICML*, volume 139 of *PMLR*, pages 8422–8431, 2021.
- 43 Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *KDD*, pages 1135–1144, 2016. doi:10.1145/2939672.2939778.
- 44 Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI*, pages 1527–1535, 2018. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16982>.
- 45 Ronald L. Rivest. Learning decision lists. *Mach. Learn.*, 2(3):229–246, 1987.
- 46 Cynthia Rudin and Seyda Ertekin. Learning customized and optimized lists of rules with mathematical programming. *Mathematical Programming Computation*, 10:659–702, 2018.
- 47 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In *SAT*, pages 539–546, 2016.
- 48 Andy Shih, Arthur Choi, and Adnan Darwiche. A symbolic approach to explaining bayesian network classifiers. In *IJCAI*, pages 5103–5111, 2018. doi:10.24963/ijcai.2018/708.

- 49 Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling LIME and SHAP: adversarial attacks on post hoc explanation methods. In *AIES*, pages 180–186, 2020. doi:10.1145/3375627.3375830.
- 50 Jinqiang Yu, Alexey Ignatiev, Peter J. Stuckey, and Pierre Le Bodic. Computing optimal decision sets with SAT. In *CP*, pages 952–970, 2020.
- 51 Jinqiang Yu, Alexey Ignatiev, Peter J Stuckey, and Pierre Le Bodic. Learning optimal decision sets and lists with sat. *JAIR*, 72:1251–1279, 2021.
- 52 Jinqiang Yu, Alexey Ignatiev, Peter J. Stuckey, Nina Narodytska, and Joao Marques-Silva. Eliminating the impossible, whatever remains must be true: On extracting and applying background knowledge in the context of formal explanations. In *AAAI*, 2023.

**A** Summaries of Results Across Folds

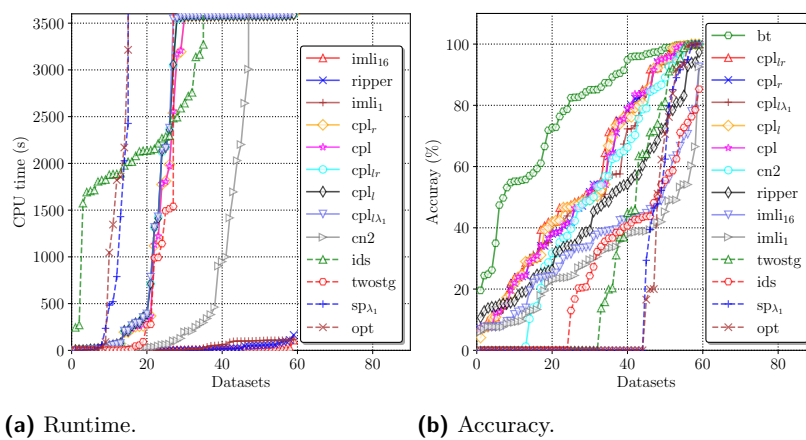


Figure 7 Experimental results of runtime and accuracy across folds.

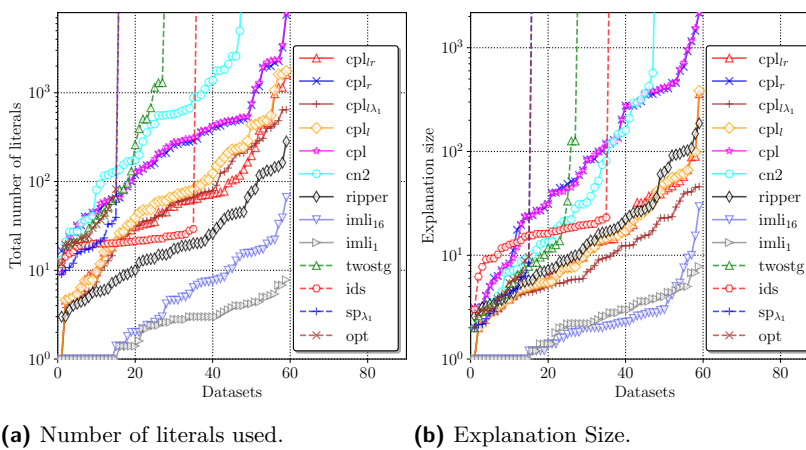
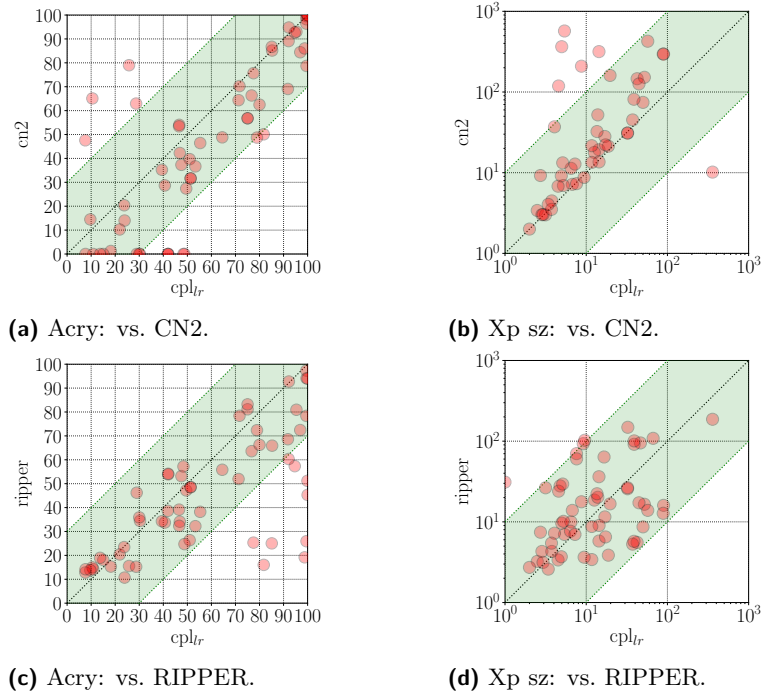


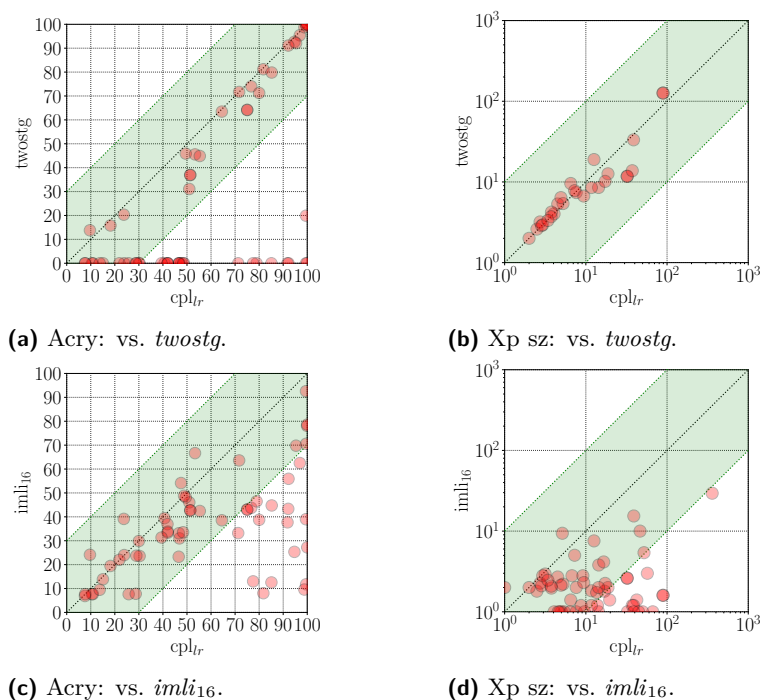
Figure 8 Experimental results of model complexity and explanation size across folds.

Figures 7 and 8 illustrate the average experimental results across folds regarding scalability, accuracy, model complexity, and explanation size. Since 5-fold cross validation is used, these results for each dataset are obtained from the average of 5 pairs of training and test data. Here, observations similar to those described in Section 5 can be made, i.e. the best

scalability and accuracy among selected DS competitors are both demonstrated by  $cpl$  and  $cpl_*$ ,  $*$   $\in \{l, r, lr, l\lambda_1\}$ , while  $iml_{i_1}$  and  $iml_{i_{16}}$  show the smallest model complexity and explanation size.



■ **Figure 9**  $cpl_{lr}$  vs. CN2 and RIPPER across folds in terms of accuracy and explanation size.



■ **Figure 10** *cpl<sub>lr</sub>* vs. *imli<sub>16</sub>* and *twostg* Across Folds in terms of accuracy and explanation size.

## B Detailed Comparisons Across Folds

In this appendix, we provide a detailed comparison of *cpl<sub>lr</sub>* versus other decision set inference approaches across folds.


Figure 9 and Figure 10 detail the comparisons of *cpl<sub>lr</sub>* with CN2, RIPPER, *imli<sub>16</sub>* and *twostg* in terms of average accuracy and explanation size across folds. As can be seen in Figure 9a, the accuracy of DSes generated by *cpl<sub>lr</sub>* is higher than the accuracy of CN2, where the average accuracy is 57.49% and 48.03%, respectively. Additionally, Figure 9b demonstrates that the explanation size of DSes produced by CN2 (81.93 on average) can be two orders of magnitude larger than the explanation size of *cpl<sub>lr</sub>* (25.88 on average).

Figure 9c illustrates that the average accuracy in RIPPER is 44.81%, which is 12.68% lower than the accuracy in *cpl<sub>lr</sub>*. Although Figure 9d depicts that RIPPER is comparable with *cpl<sub>lr</sub>* regarding explanation size (29.08 and 25.34 on average respectively), RIPPER is less interpretable as it computes DSes representing only one class.

As can be observed in Figure 10a, the accuracy of *twostg* (29.67% on average) is 27.82% lower than the accuracy in *cpl<sub>lr</sub>* while Figure 10b illustrates that the explanation size is comparable between the two approaches. Finally, Figure 10c demonstrates that the accuracy of *imli<sub>16</sub>* is 22.02% lower than the accuracy of *cpl<sub>lr</sub>* on average. However, as can be seen in Figure 10d, the explanation size of *imli<sub>16</sub>* is smaller than the explanation size of *cpl<sub>lr</sub>* but *imli<sub>16</sub>* generates DSes targeting only a single class, which significantly diminishes the interpretability of computed DSes.



# Searching for Smallest Universal Graphs and Tournaments with SAT

Tianwei Zhang ✉ 🏠 

Algorithms and Complexity Group, TU Wien, Austria

Stefan Szeider ✉ 🏠 

Algorithms and Complexity Group, TU Wien, Austria

---

## Abstract

A graph is induced  $k$ -universal if it contains all graphs of order  $k$  as an induced subgraph. For over half a century, the question of determining smallest  $k$ -universal graphs has been studied. A related question asks for a smallest  $k$ -universal tournament containing all tournaments of order  $k$ .


This paper proposes and compares SAT-based methods for answering these questions exactly for small values of  $k$ . Our methods scale to values for which a generate-and-test approach isn't feasible; for instance, we show that an induced 7-universal graph has more than 16 vertices, whereas the number of all connected graphs on 16 vertices, modulo isomorphism, is a number with 23 decimal digits. Our methods include static and dynamic symmetry breaking and lazy encodings, employing external subgraph isomorphism testing.

**2012 ACM Subject Classification** Mathematics of computing → Extremal graph theory; Software and its engineering → Constraint and logic languages; Hardware → Theorem proving and SAT solving; Mathematics of computing → Graph enumeration

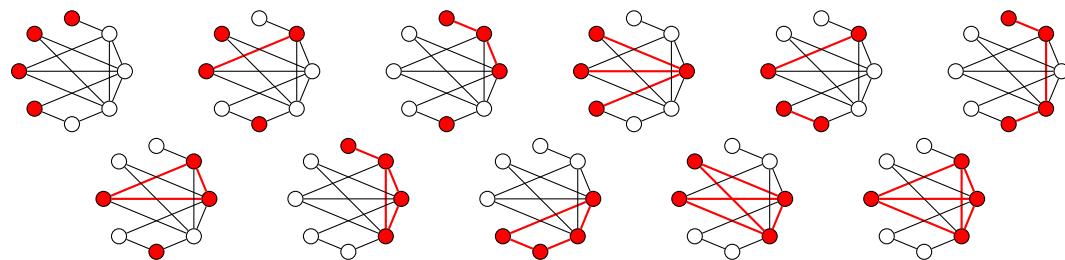
**Keywords and phrases** Constrained-based combinatorics, synthesis problems, symmetry breaking, SAT solving, subgraph isomorphism, tournament, directed graphs

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.39

**Supplementary Material** *Software:* <https://doi.org/10.5281/zenodo.8147732>

**Funding** The project leading to this publication has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101034440, and was supported by the Vienna Science and Technology Fund (WWTF) within the project ICT19-065. 

**Acknowledgements** This work was carried out in part while the second author visited the Simons Institute for the Theory of Computing, University of Berkeley, within the program *Extended Reunion: Satisfiability*. The authors thank Ciaran McCreesh for advising them to use and modify the Glasgow subgraph solver.



■ **Figure 1** A smallest induced 4-universal graph and how all 11 graphs of order 4 embeds into it.



© Tianwei Zhang and Stefan Szeider;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 39; pp. 39:1–39:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany





■ **Figure 2** A smallest 4-universal tournament and how all 4 tournaments of order 4 embeds into it.

## 1 Introduction

A graph  $G$  is *induced  $k$ -universal* if it contains all  $k$ -vertex graphs as induced subgraphs; in other words, any  $k$ -vertex graph can be obtained from  $G$  by deleting some of  $G$ 's vertices. This notion of a universal graph, which extends naturally to directed graphs and tournaments (orientations of complete graphs) was introduced by Rado in 1964 [25] and has since then attracted much attention in combinatorics. Figures 1 and 2 provide examples of a universal graph and a universal tournament, respectively. The fundamental question is to determine for each integer  $k \geq 1$  the smallest  $n$  such that an induced  $k$ -universal graph with  $n$  vertices exists; this number is denoted  $f(k) = n$ . In 1965, Moon [21] obtained the bounds  $2^{(k-1)/2} \leq f(k) \leq O(k \cdot 2^{k/2})$ ; Alstrup et al. [2] recently improved the upper bound to  $f(k) \leq 16 \cdot 2^{k/2}$ . The tight asymptotic upper bound of  $f(k) \leq (1 + o(1))2^{(k-1)/2}$  is due to Alon [1].

In contrast to these general bounds and asymptotic results, very little is known about exact  $f(k)$  values, even for small  $k$ . Trimble [26] carried out a brute-force search, expanding an approach suggested by Preen [24], and could determine the values of  $f(k)$  for  $k \leq 6$  and the interval  $16 \leq f(7) \leq 18$ . The brute-force search enumerates all candidate  $n$ -vertex graphs up to isomorphism and tests for each of them whether it contains all the  $k$ -vertex graphs as induced subgraphs. We refer to the  $k$ -vertex graphs that need to be checked as induced subgraphs as *pattern graphs*. Tools exist for the isomorph-free enumeration of all connected  $n$ -vertex graphs (we can assume that a candidate graph is connected), but the number of such graphs exceeds eleven million for  $n = 10$ , and one billion for  $n = 11$  [23, A001349]. Thus, the feasibility of the brute-force approach quickly hits a rigid boundary. In particular, we cannot tighten the gap  $16 \leq f(7) \leq 18$  in this way, as enumerating all graphs with 16 vertices is far out of reach, although we can easily enumerate all the 7-vertex pattern graphs; the exact number is 1044 [23, A000088].

This paper proposes new methods that allow us to break this boundary. The idea is to formulate the problem as a *synthesis problem* for the universal graph rather than the easier problem of testing a candidate graph for being universal. This way, we can avoid the bottleneck of enumerating all candidate graphs. We formulate the synthesis problem in propositional logic to harness the power of solvers for the propositional satisfiability problem (SAT) [8]. Suppose the sought-for universal graph  $G$  has the numbers  $1, \dots, n$  as its vertices. For each pair  $1 \leq i < j \leq n$ , we introduce a propositional variable  $e_{i,j}$  (an *edge variable*) whose truth value determines whether there is an edge between vertices  $i$  and  $j$  in  $G$ .

We propose and compare various strategies and techniques of encoding the property of  $G$  being induced  $k$ -universal. Each of the encodings must combine two fundamental properties: (i) to ensure that the found graph is indeed induced  $k$ -universal and (ii) to break symmetries to minimize the enormous search space. We have two fundamentally different approaches for properties (i) and (ii).

Our first approach for ensuring that the found graph is indeed induced  $k$ -universal uses a *direct SAT encoding*. For the relevant cases, the number of pattern graphs is reasonably small, and we can enumerate them up to isomorphism. For each pattern graph, we obtain

a formula which constrains the edge variables in such a way that the sought-for universal graph contains an induced subgraph that is isomorphic to the pattern graph. A conjunction over the formulas, one for each pattern graph, yields the SAT encoding.

Our second approach for ensuring universality utilizes the *Glasgow subgraph solver (GSS)* [19], a powerful constraint-based tool for testing subgraphs. However, during the SAT solver's search we don't have a candidate graph available for testing whether it accommodates all the required induced subgraphs. A partial truth assignment instead sets a subset of the edge variables to true or false, with some edge variables remaining undecided. Thus, a partial truth assignment gives rise to a *partially defined graph*. Our objective is to find out already for a partially defined graph whether it could be extended to a fully defined induced  $k$ -universal graph. Indeed, we could achieve this by accessing GSS's internal functionality.

Our first approach to breaking symmetries utilizes the *SAT modulo Symmetries (SMS)* framework [17]. Also SMS works on partially defined graphs as represented by a partial truth assignment to edge variables. Whenever the solver determines an edge, the partially defined graph represented by the current partial truth assignment is sent to an external propagator, which determines whether the partially defined graph can be extended to a fully defined graph that is canonical (i.e., a unique graph within its isomorphism class). If not, a clause is sent back to the solver.

Our second approach to symmetry breaking is based on the concept of *templates* which serve two purposes: symmetry breaking and search space partition. In an offline phase, we fix a small collection of highly symmetric pattern graphs and compute all possible ways these pattern graphs can interact as induced subgraphs in the universal graph, up to isomorphism. Each possible interaction gives rise to a template. By hardcoding a template, we get a formula that is satisfiable if and only if there exists an induced  $k$ -universal graph on  $n$  vertices in which the fixed collection of pattern graphs interact as prescribed in the template.  $f(k) \leq n$  if and only if for at least one of the templates, the corresponding formula is satisfiable. In addition to the symmetry breaking aspect, the templates approach allows us to parallelize the search, running SAT calls for various templates independently.

We also evaluate our SAT-based approach to determining smallest  $k$ -universal tournaments. A tournament is a directed graph that can be obtained from a complete undirected graph by orienting its edges. The name tournament originates from such a directed graph's interpretation as the outcome of a round-robin tournament in which every player encounters every other player exactly once, and in which no draws occur. Tournaments are a central combinatorial object whose properties have been intensively studied. The notion of universality applies to tournaments in a natural way: we say that a tournament is  $k$ -universal if it contains each tournament on  $k$ -vertices as a subgraph. Let  $t(k)$  denote the smallest number of vertices a  $k$ -universal tournament can have. The study of  $k$ -universal tournaments goes back more than 50 years to Moon's book (entirely dedicated to tournaments [22]) where he determined the bounds  $2^{(k-1)/2} \leq t(k) \leq O(k \cdot 2^{k/2})$ . The improvements on the bounds parallel the ones for induced  $k$ -universal graphs:  $f(k) \leq 16 \cdot 2^{\lceil k/2 \rceil}$  by Alstrup et al. [2] and  $f(k) \leq (1 + o(1))2^{(k-1)/2}$  by Alon [1].

The situation regarding exact values for  $t(k)$  is even worse than for  $f(k)$ . While the number of tournaments up to isomorphism is for all  $k \leq 19$  [23, A000568], very little can be found on the value of  $t(k)$  in the literature. The only obvious values are  $t(1) = 1$  and  $t(2) = 2$ . In Moon's book [22], it is left as an exercise to the reader to determine the value of  $t(3)$  and  $t(4)$ , indicating that these numbers might be easily deduced.

We implemented our SAT-based approaches to obtain new results on smallest induced  $k$ -universal graphs and smallest  $k$ -universal tournaments. In particular, we could improve the known lower bound for the order of an induced 7-universal graph by showing that no induced

■ **Table 1** Overview of results along the order  $k$  of the pattern graphs or tournaments.  $\mathcal{G}(k)/\mathcal{T}(k)$  give the number of pattern graphs/tournaments of order  $k$ ,  $f(k)/t(k)$  give the order of a smallest (induced)  $k$ -universal graph or tournament, and  $F(k)/T(k)$  their number modulo isomorphism, respectively. New results are indicated in bold face.

$k$	Graphs			Tournaments		
	$\mathcal{G}(k)$	$f(k)$	$F(k)$	$\mathcal{T}(k)$	$t(k)$	$T(k)$
1	1	1	1	1	1	1
2	2	3	2	1	2	1
3	4	5	5	2	4	<b>3</b>
4	11	8	438	4	5	<b>1</b>
5	34	10	22	12	<b>8</b>	<b>1643</b>
6	156	14	> <b>36294</b>	56	<b>10</b>	<b>1088</b>
7	1044	[ <b>17,18</b> ]	–	456	[ <b>13,15</b> ]	–

7-universal graph with 16 vertices exists, leaving the possibilities 17 and 18 for the smallest solution, and we could determine the exact order of smallest  $k$ -universal tournaments up to  $k = 6$  and leaving the possibilities 13,14, and 15 for the smallest solution for  $k = 7$ . We also determined the precise number of optimal solutions for  $k$ -universal tournaments for  $k \leq 6$ , up to isomorphism, which is of independent interest for combinatorial research. Our main results are summarized in Table 1, smallest  $k$ -universal tournaments that we identified are shown in Figures 6 and 7 in the appendix.

## 2 Preliminaries

We denote the set  $\{1, \dots, n\}$  by  $[n]$  and the set  $\{m, m + 1, \dots, n\}$  by  $[m, n]$  for  $m \leq n$ .

### 2.1 Graphs

We consider simple undirected graphs  $G$ , denoting the vertex set and the edge set of  $G$  by  $V(G)$  and  $E(G)$ , respectively. The *order* of a graph is the number  $|V(G)|$  of its vertices. An edge between vertices  $u, v \in V(G)$  is denoted  $uv$  or equivalently  $vu$ . A graph  $G'$  is a *subgraph* of a graph  $G$  if  $V(G') \subseteq V(G)$  and  $E(G') \subseteq E(G)$ . A subgraph  $G'$  of  $G$  is *induced* if for any  $u, v \in V(G')$ ,  $uv \in E(G)$  implies  $uv \in E(G')$ . Thus, an induced subgraph is determined by its set of vertices.

The *neighborhood* of a vertex  $u \in V(G)$  is denoted  $N_G(u)$ , i.e.,  $N_G(u) := \{x \in G \mid ux \in E(G)\}$ . Two vertices  $v, v' \in G$  are *twins* if they have the exact same neighbors excluding each other, i.e.,  $N_G(u) \setminus \{v\} = N_G(v) \setminus \{u\}$ .

The *complement graph*  $\overline{G}$  of a graph  $G$  has  $V(\overline{G}) = V(G)$  and  $E(\overline{G}) := \{uv \mid u \neq v \in V(G), uv \notin E(G)\}$ .

We denote the complete graph of order  $k$  by  $K_k$ , the complete bipartite graph by  $K_{l,r}$ .

A *monomorphism* from  $H$  to  $G$  is an injective mapping  $\phi : V(H) \rightarrow V(G)$  such that for all  $u, v \in V(H)$  we have  $\phi(u)\phi(v) \in E(G)$  if  $uv \in E(H)$ . Such a monomorphism  $\phi$  is *faithful* if for all  $u, v \in V(H)$  we have  $uv \in E(H)$  if  $\phi(u)\phi(v) \in E(G)$  [11]. We say  $H$  *embeds into*  $G$  if there is a faithful monomorphism from  $H$  to  $G$ . A *bijective* faithful monomorphism is an *isomorphism*. We refer to  $H$  as the *pattern graph* and  $G$  as the *target graph*. For a monomorphism  $\phi$  from  $H$  to  $G$  we put  $\phi(V(H)) := \{\phi(v) \mid v \in V(H)\}$ .

Let  $\mathcal{G}(k)$  denote the class of all graphs of order  $k$  up to isomorphism. A graph  $G$  is *induced  $k$ -universal* if all  $H \in \mathcal{G}(k)$  embed into  $G$ .  $f(k)$  is the smallest order of an induced  $k$ -universal graph, and  $F(k)$  is the number of non-isomorphic  $k$ -universal graphs of order  $f(k)$ .

## 2.2 Partially Defined Graphs

The notion of partially defined graphs was introduced in the context of SMS [17], for capturing the combinatorial object represented by a partial truth assignment on the edge variables. In this paper, we utilize this concept for two further purposes: to check with an external subgraph solver whether the current branch of the search has the potential of success (Section 4.5) and for the formulation of our template method for symmetry breaking and search partition (Section 4.6).

A *partially defined graph* is a graph where some edges are undefined in the sense that their presence in the graph is open. Formally, a partially defined graph  $G$  is a graph whose edge set  $E(G)$  is split into disjoint sets  $E_d(G)$  and  $E_u(G)$ , the sets of defined and undefined edges, respectively.  $G$  is *fully defined* if  $E_u(G) = \emptyset$ . For a partially defined graph  $G$ ,  $\mathcal{X}(G)$  denotes the set of all fully defined graphs  $G$  can be extended to, i.e.,  $G' \in \mathcal{X}(G)$  if and only if  $V(G') = V(G)$ ,  $E_d(G) \subseteq E(G') \subseteq E(G)$ .

We extend the notion of a faithful monomorphism and an isomorphism to partially defined graphs. Let  $H, G$  be partially defined graphs. An injective mapping  $\phi : V(H) \rightarrow V(G)$  is a *faithful monomorphism* from  $H$  to  $G$  if for all  $u, v \in V(H)$  it holds that

1.  $uv \in E_d(H)$  if and only if  $\phi(u)\phi(v) \in E_d(G)$ , and
2.  $uv \in E_u(H)$  if and only if  $\phi(u)\phi(v) \in E_u(G)$ .

If there is a faithful monomorphism from  $H$  to  $G$  we say that  $H$  embeds into  $G$ . We say that  $H$  and  $G$  are *isomorphic* if there is a *bijective* faithful monomorphism from  $H$  to  $G$ .

## 2.3 Directed Graphs and Tournaments

We denote a directed graph (or digraph)  $D$  by its vertex set  $V(D)$  and arc set  $A(D) \subseteq V(D) \times V(D)$ . A digraph  $D$  is an *oriented graph* if it has no directed digon  $\{(u, v), (v, u)\}$ . (Induced) subdigraphs, isomorphisms, and faithful monomorphisms are defined for digraphs analogously to graphs. A *tournament* is an oriented graph with a maximal number of arcs, i.e., adding any further arc creates a digon. A tournament is *transitive* if it contains no directed cycles. A tournament  $D'$  is a subtournament of tournament  $D$  if it is a subdigraph of  $D$ . If a tournament  $H$  is isomorphic to a subtournament of  $D$ , then we say that  $H$  embeds into  $D$ . Hence  $H$  embeds into  $D$  if and only if there exists a faithful monomorphism  $\phi : V(H) \rightarrow V(D)$ .

Let  $\mathcal{T}(k)$  denote the class of all tournaments of order  $k$  up to isomorphism. A tournament  $D$  is  *$k$ -universal* if all  $H \in \mathcal{T}(k)$  embed into  $D$ . Let  $T(k)$  denote the number of non-isomorphic tournaments of order  $t(k)$  that are induced  $k$ -universal tournaments.

## 2.4 SAT

We consider propositional formulas in conjunctive normal form (CNF). A CNF formula is a conjunction of *clauses*, each clause is a disjunction of *literals*, a literal is a propositional variable  $x$  or its negation  $\neg x$ . A propositional formula  $F$  is *satisfiable* if there exists a mapping  $\tau$  that assigns each variable a truth value  $\in \{0, 1\}$  such that each clause contains a literal  $x$  with  $\tau(x) = 1$  or a literal  $\neg x$  with  $\tau(x) = 0$ . A truth assignment is *partial* for a CNF

formula if it is defined only for a subset of the formula's variables. SAT solvers are tools that decide whether a given CNF formula is satisfiable or not [8]. Today's most powerful complete SAT solvers follow the conflict driven clause learning (CDCL) paradigm. Modern CDCL SAT solvers like Cadical [4] can certify the unsatisfiability of a CNF formula by producing DRAT proofs (deletion, reverse asymmetric tautology) that can be independently checked and verified [12, 27].

### 3 A Lower Bound for Induced $k$ Universal Graphs

Trimble [26] showed that there is no graph of order less than  $2k + 2$  that is universal for  $\{K_k, \overline{K_k}, K_{3,3}, \overline{K_{3,3}}\}$ , for all  $k \geq 6$ . We extend this argument to show the following more general proposition, which gives a strictly tighter lower bound for  $k \geq 8$ .

► **Proposition 1.**  $2k + 2\lfloor \frac{k}{2} \rfloor - 4 \leq f(k)$  for all  $k \geq 2$ .

**Proof.** Suppose  $G$  is an induced  $k$ -universal graph. Graphs  $K_k$  and  $\overline{K_k}$  are both elements of  $\mathcal{G}(k)$ , and therefore there exist faithful monomorphisms  $\phi_K : V(K_k) \rightarrow V(G)$  and  $\phi_I : V(\overline{K_k}) \rightarrow V(G)$ . Let  $S_1 := \{\phi_K(v) \mid v \in V(K_k)\}$ ,  $S_2 := \{\phi_I(v) \mid v \in V(\overline{K_k})\}$  and  $S_3 := V(G)/(S_1 \cup S_2)$ . Clearly, the subgraphs induced by  $S_1$  and  $S_2$  are a clique and an independent set, respectively, and therefore may overlap by no more than one vertex. This implies that  $|S_1 \cup S_2| \geq 2k - 1$  and  $|S_3| \leq n - 2k + 1$ . Define  $H := K_{\lfloor \frac{k}{2} \rfloor, \lfloor \frac{k}{2} \rfloor + (k \bmod 2)}$ . Let  $G_1$  be an induced subgraph of  $G$  that is isomorphic to  $H$ . Since there are no edges between the two cliques of  $G_1$ , it must be the case that at least one of the two cliques of  $G_1$  does not intersect  $S_1$ . Denote this clique by  $C$ . Since  $S_2$  is an independent set in  $G$ ,  $C$  can only intersect  $S_2$  by at most one vertex. Since  $C$  has at least  $\lfloor \frac{k}{2} \rfloor$  vertices, it follows that  $C$  intersects  $S_3$  by at least  $\lfloor \frac{k}{2} \rfloor - 1$  vertices. In other words, the subgraph induced by  $S_3$  contains a clique  $D$  of size  $\lfloor \frac{k}{2} \rfloor - 1$ .

Now consider the graph  $\overline{H}$ . Since  $\overline{H}$  is an induced subgraph of  $G$ , it follows that  $H = \overline{\overline{H}}$  is an induced subgraph of  $\overline{G}$ . We can repeat the argument of the previous paragraph with the roles of  $S_1$  and  $S_2$  reversed to show that the subgraph induced by  $S_3$  contains an independent set of size  $\lfloor \frac{k}{2} \rfloor - 1$ . Since this independent set can overlap with the clique  $D$  in at most one vertex, it follows that the minimal size of  $S_3$  is  $2(\lfloor \frac{k}{2} \rfloor - 1) - 1$ . In other words, for any  $k$ -universal graph of order  $n$  where  $S_1 \cap S_2 \neq \emptyset$ ,  $n - 2k + 1 \geq 2(\lfloor \frac{k}{2} \rfloor - 1) - 1$ , which implies that  $f(k) \geq 2k + 2\lfloor \frac{k}{2} \rfloor - 4$ . Hence, the proposition is shown. ◀

Proposition 1 gives us  $f(2) \geq 2$ ,  $f(3) \geq 4$ ,  $f(4) \geq 8$ ,  $f(5) \geq 10$ ,  $f(6) \geq 14$  and  $f(7) \geq 16$ . Note that for  $k = 4, 5, 6$ , the lower bound given by the proposition is also the exact value of  $f(k)$ . Even though it does not give a tighter lower bound for  $f(7)$ , it does improve on the existing knowledge for  $k \geq 8$ .

► **Corollary 2.** Let  $G$  be a  $k$ -universal graph. If there exist faithful monomorphisms  $\phi_K : V(K_k) \rightarrow V(G)$  and  $\phi_I : V(\overline{K_k}) \rightarrow V(G)$  with  $\phi_K(V(K_k)) \cap \phi_I(V(\overline{K_k})) = \emptyset$ , then the order of  $G$  is at least  $2k + 2\lfloor \frac{k}{2} \rfloor - 3$ .

### 4 Encodings for Induced $k$ universal Graphs

Throughout this section, we fix an integer  $n$  and consider a potential universal graph  $G$  with  $V(G) = [n]$  whose edges are represented by *edge variables*  $e_{u,v}$ ,  $1 \leq u < v \leq n$ .

## 4.1 Encoding Universality

First, we define a CNF formula  $M(H, n)$  that ensures that  $(m_{u,v})_{u \in V(H), v \in V(G)}$  encodes an injective mapping from  $V(H)$  to  $V(G)$ .

$$M(H, n) := \bigwedge_{u \in V(H)} \left( \bigvee_{x \in [n]} m_{u,x} \right) \wedge \bigwedge_{\substack{u \neq v \in V(H) \\ x \in [n]}} (\neg m_{u,x} \vee \neg m_{v,x}) \wedge \bigwedge_{\substack{u \in V(H) \\ x \neq y \in [n]}} (\neg m_{u,x} \vee \neg m_{u,y})$$

Now, we specify a CNF formula  $F(H, n)$  which is satisfiable if and only if  $H$  embeds into  $G$ .  $F(H, n)$  encodes the existence of a faithful monomorphism  $\phi$  from  $H$  to  $G$  in terms of variables  $m_{u,v}$ ,  $u \in V(H)$ ,  $v \in V(G)$ , which are true if and only if  $\phi(u) = v$ .

$$F(H, n) := M(H, n) \wedge \bigwedge_{\substack{uv \in E(H) \\ 1 \leq x < y \leq n}} (\neg m_{u,x} \vee \neg m_{u,y} \vee e_{x,y}) \wedge \bigwedge_{\substack{u \neq v \in V(H) \\ \text{s.t. } uv \notin E(H) \\ 1 \leq x < y \leq n}} (\neg m_{u,x} \vee \neg m_{v,y} \vee \neg e_{x,y}).$$

The second and third conjunct ensure that the mapping preserves edges and non-edges, respectively. The formula

$$U(k, n) = \bigwedge_{H \in \mathcal{G}(k)} F(H, n)$$

is satisfiable if and only if there exists a  $k$ -universal graph on  $n$  vertices. From a satisfying assignment we can read off a  $k$ -universal graph.

## 4.2 Symmetry-breaking Based on Twins

A basic symmetry-breaking can already be achieved by observing the symmetries of a pattern graph  $H \in \mathcal{G}(k)$ . Specifically, if  $v, v' \in V(H)$  are twins and  $\phi$  is a faithful monomorphism from  $H$  to  $G$ , then  $\phi' : V(H) \rightarrow V(G)$  defined as

$$\phi'(x) := \begin{cases} \phi(v') & \text{if } x = v, \\ \phi(v) & \text{if } x = v', \\ \phi(x) & \text{otherwise,} \end{cases}$$

is also a faithful monomorphism. This means that we can always require that the faithful monomorphism encoded by  $(m_{u,v})_{u \in V(H), v \in V(G)}$  preserves the order of vertices (seen as integers) for twins. The following CNF formula encodes this additional requirement.

$$\text{Twin}(k) := \bigwedge_{\substack{H \in \mathcal{G}(k) \\ u < v \in V(H) \\ \text{s.t. } u, v \text{ are twins} \\ 1 \leq x < y \leq n}} \neg m_{v,x} \vee \neg m_{u,y}$$

## 4.3 Embedding $K_k$ and $\overline{K_k}$

From preliminary experiments we observed that fixing (or ‘‘hardcoding’’) some edges/non-edges of the target graph can drastically speed up the solving process. Trimble [26] hardcodes  $K_k$  and  $\overline{K_k}$  in his experiments, and we agree that it is a good place to start. Since the two subgraphs can have at most one overlapping vertex, we have the following two formulas, each hardcoding one of the two possibilities. We assume  $n \geq 2k$ .

$$\text{KI}_{\text{disjoint}}(k) := \bigwedge_{1 \leq x < y \leq k} e_{x,y} \wedge \neg e_{x+k,y+k}, \quad \text{KI}_{\text{overlap}}(k) := \bigwedge_{1 \leq x < y \leq k} e_{x,y} \wedge \neg e_{x+k-1,y+k-1}.$$

In Section 4.6, we will extend this idea by hardcoding more pattern graphs.

To sum up, the following two encodings are shared by all methods.

$$U_d(k, n) := U(k, n) \wedge \text{Twin}(k) \wedge \text{KI}_{\text{disjoint}}(k),$$

$$U_o(k, n) := U(k, n) \wedge \text{Twin}(k) \wedge \text{KI}_{\text{overlap}}(k).$$

Note that  $U(k, n)$  is satisfiable if and only if  $U_d(k, n)$  or  $U_o(k, n)$  is satisfiable.

#### 4.4 SAT Modulo Symmetries (SMS)

If a SAT solver checks the satisfiability of  $U_d(k, n)$  or  $U_o(k, n)$ , it does not break symmetries and considers isomorphic copies of graphs implicitly represented by the edge variables, making the approach impractical. Fortunately, the SAT modulo Symmetries (SMS) framework [17] offers a solution.

For two (fully defined) graphs  $G_1, G_2$ , let  $G_1 \preceq G_2$  if and only the adjacency matrix of  $G_1$  is lexicographically smaller or equal to the adjacency matrix of  $G_2$  (we consider the matrix as the string obtained by concatenating the rows of the matrix).

SMS works as follows. Whenever the SAT solver decides on an edge variable  $e_{u,v}$ , the partially defined graph  $G$  represented by the current truth assignment on the edge variables is sent to an external propagator. With a certain pre-determined probability the propagator checks a necessary condition for  $\mathcal{X}(G)$  containing a  $\preceq$ -minimal graph. If the condition is not met, then this branch of the search can be terminated: a clause that excludes  $G$  is learned and sent back to the SAT solver.

In our context, while the satisfiability of  $U(k, n)$  and  $\text{Twin}(k)$  are unaffected by our choice of the  $\preceq$ -minimal graph as the representative target graph for each isomorphic class,  $\text{KI}_{\text{disjoint}}(k)$  and  $\text{KI}_{\text{overlap}}(k)$  might cause an otherwise satisfying assignment to fail the minimality check.

The solution to this is to take advantage of the following functionality of SMS. When SMS checks the  $\preceq$ -minimality of a partially defined graph  $G$ , one can specify a partition  $P$  of the vertex set such that  $\preceq$ -minimality is only checked among all partially defined graphs that can be obtained from  $G$  by permuting vertices within an equivalence class of  $P$ . For  $U_d(k, n)$ , we give SMS the partition  $\{[k], [k+1, 2k], [2k+1, n]\}$ , and for  $U_o(k, n)$ , we give the partition  $\{[k-1], [k, k], [k+1, 2k-1], [2k, n]\}$ . The disadvantage of this adaption is that we are not able to trim down search branches as efficiently, but the advantage is that we can enjoy the speed-up brought by hardcoding a part of the target graph.

#### 4.5 External Subgraph Isomorphism Testing

Next we lay out a lazy encoding strategy where we check during the SAT solver's run whether the partially defined graph represented by the current partial assignment to the edge variables can be extended to a  $k$ -universal graph. We utilize the *Glasgow subgraph solver (GSS)* [19] to achieve this. The GSS is a state-of-the-art solver for the subgraph isomorphism problem based on constraint programming. Given a pattern graph  $H$  and a target graph  $G$ , the subgraph isomorphism problem asks whether there exists a monomorphism from  $H$  to  $G$ . One of the key concepts underlying the GSS's functionality is to generate auxiliary graphs  $H_1, \dots, H_m$  with  $V(H) = V(H_i), 1 \leq i \leq m$ , and  $G_1, \dots, G_m$  with  $V(G) = V(G_i), 1 \leq i \leq m$ , such that a mapping from  $V(G)$  to  $V(H)$  is a (faithful) monomorphisms from  $H$  to  $G$  if and only if it is a monomorphisms from  $H_i$  to  $G_i$ , for every  $1 \leq i \leq m$ . Now the solver searches for a mapping from  $V(G)$  to  $V(H)$  that is a monomorphism from  $H_i$  to  $G_i$ , for every  $1 \leq i \leq m$ . For instance, the GSS can decide whether  $H$  is isomorphic to an *induced* subgraph of  $G$  by setting  $H_1 = H, G_1 = G, H_2 = \overline{H}$ , and  $G_2 = \overline{G}$ .

We utilize this functionality of the GSS to check whether a partially defined graph can be extended to an inducted  $k$ -universal graph.

For a partially defined graph  $G$  we define two fully defined graphs  $G_{-0}$  and  $G_{-1}$  by setting  $V(G_{-0}) = V(G_{-1}) = V(G)$ ,

$$E(G_{-0}) = \{uv \mid u \neq v \in V(G), uv \in E_d(G) \cup E_u(G)\}, \text{ and}$$

$$E(G_{-1}) = \{uv \mid u \neq v \in V(G), uv \notin E_d(G)\}.$$

► **Proposition 3.** *Let  $H$  be a graph and  $G$  a partially defined graph  $G$ .  $H$  embeds into some graph in  $\mathcal{X}(G)$  if and only if there is a mapping from  $V(H)$  to  $V(G)$  that is a monomorphism both from  $H$  to  $G_{-0}$  and from  $\bar{H}$  to  $G_{-1}$ .*

Thus, we can use the GSS with a minimal change to its interface to check whether  $H$  embeds into some graph in  $\mathcal{X}(G)$ . We integrate the subgraph solver into the external propagator of the SMS framework. Whenever the SAT solver decides on an edge variable  $e_{u,v}$ , the partially defined graph  $G$  represented by the current truth assignment on the edge variables is sent to the propagator. The propagator checks whether all graphs from a set of randomly selected pattern graphs of a certain size are embeddable into  $G$ . If the check fails, this branch of the search can be terminated: a clause that excludes  $G$  is learned and sent back to the SAT solver. We set up three parameters for this process to control the usage of the subgraph solver: *Sample size* specifies the number of randomly selected pattern graphs. *Threshold* specifies the minimal number of determined edges/non-edges that is required to present in  $G$  for the embedability check to be performed. Finally, *frequency* specifies the likelihood the embedability check is performed and is indicated as an integer  $f$ . Everytime the SAT solver decides on an edge variable  $e_{u,v}$ , if the threshold is met, then there is a  $1/f$  chance the embedability check will be performed.

## 4.6 Templates

Next we propose a different approach, based on the concept of a template, which extends the simple method of hardcoding edges or non-edges for speeding up the solving process discussed in Section 4.3.

As it turned out, templates can be defined as partially defined graphs, but with a different purpose than the partially defined graphs used in SMS. Fix a small collection of graphs from  $\mathcal{G}(k)$ . We want each template to provide a distinct way in terms of how this small collection of graphs can be embedded at simultaneously in the universal graph. Ideally, these templates should allow us to hardcode as many edges/non-edges into the universal graph as possible for the benefit of speed-up. At the same time, there should not be too many of them, as we set a separate SAT instance of each template and need to show that all of them are unsatisfiable to establish a lower bound. In the end, we also want them to be easy to generate.

Let  $\mathcal{H} \subseteq \mathcal{G}(k)$ . An  $(n, \mathcal{H})$ -*template* is a partially defined graph  $G$  such that there exists a family of faithful monomorphism  $(\delta_H : V(H) \rightarrow V(G))_{H \in \mathcal{H}}$  with the additional condition that for all  $x, y \in V(G)$ , if there is no  $H \in \mathcal{H}$  such that  $x, y \in \delta_H(V(H))$ , then  $xy \in E_u(G)$ . Let  $\mathcal{H}^{(n)}$  denote the set of all  $(n, \mathcal{H})$ -templates modulo isomorphism.

► **Proposition 4.** *Let  $k < n$  integers and  $\mathcal{H} \subseteq \mathcal{G}(k)$ . Then  $f(k) \leq n$  if and only if  $\mathcal{X}(G)$  contains a  $k$ -universal graph for some  $G \in \mathcal{H}^{(n)}$ .*

According to this proposition, we can limit the search for an induced  $k$ -universal graph to the graphs in  $\mathcal{X}(G)$  for some  $G \in \mathcal{H}^{(n)}$ . It remains to generate  $\mathcal{H}^{(n)}$  for suitable sets  $\mathcal{H}$  for which



$\mathcal{H}^{(n)}$  remains reasonably small. We accomplish that by representing templates (i.e., partially defined graphs) as directed graphs and utilizing SMS for directed graphs as introduced by Kirchweger et al. [16], as follows.

Let us define a mapping  $\mathcal{D}$  from the set of partially defined undirected graphs to the set of (fully defined) directed graphs: for each partially defined graph  $G$ , we put  $V(\mathcal{D}(G)) := V(G)$  and  $A(\mathcal{D}(G)) := \{uv \in E_d(G) \mid u > v\} \cup E_u(G)$ . We also define the reverse mapping  $\mathcal{D}'$  from directed graphs to partially defined undirected graphs: for each directed graph  $D$ , we put  $V(\mathcal{D}'(D)) := V(D)$ ,  $E_d(\mathcal{D}'(D)) := \{uv \mid uv \in A(D), vu \notin A(D)\} \cup \{uv \mid vu \in A(D), uv \notin A(D)\}$  and  $E_u(\mathcal{D}'(D)) := \{uv \mid uv, vu \in A(D)\}$ . For any partially defined graph  $G$  we have that  $\mathcal{D}'(\mathcal{D}(G))$  and  $G$  are isomorphic.

We encode into CNF the condition for a directed graph  $D$  to be  $\mathcal{D}(G)$  for some template  $G$  of  $\mathcal{H}^{(n)}$ , use SMS for digraphs to enumerate all such  $D$  modulo isomorphism, and then apply  $\mathcal{D}'$  to recover the corresponding  $G$ . We encode the condition for a directed graph  $D$  to be  $\mathcal{D}(G)$  for some template  $G$  of  $\mathcal{H}$  as follows.

$$D(\mathcal{H}, n) := \bigwedge_{H \in \mathcal{H}} \left( M(H, n) \wedge \bigwedge_{\substack{uv \in E(H) \\ 1 \leq x < y \leq n}} (m_{u,x} \wedge m_{v,y}) \rightarrow (\neg a_{x,y} \wedge a_{y,x}) \wedge \bigwedge_{\substack{uv \notin E(H) \\ 1 \leq x < y \leq n}} (m_{u,x} \wedge m_{v,y}) \rightarrow (\neg a_{x,y} \wedge \neg a_{y,x}) \right) \\ \wedge \bigwedge_{1 \leq x < y \leq n} \left( (a_{x,y} \wedge a_{y,x}) \vee \bigvee_{H \in \mathcal{H}} \left( \bigvee_{i \in V(H)} m_{i,x} \wedge \bigvee_{i \in V(H)} m_{i,y} \right) \right).$$

The first three conjuncts in  $D(\mathcal{H}, n)$  enforce that  $H$  is an induced subgraph of  $G$ , and the last conjunct ensures that for all pairs  $u, v$  of  $V(G)$  such that  $u, v$  are not both in the range of  $\delta_H$  for all  $H \in \mathcal{H}$ , where  $\delta_H$  refers to the faithful monomorphism from  $H$  to  $G$ , we have  $uv \in U(G)$ .

By trial and error, we determined that an ideal set of pattern graphs to generate templates for the task of determining  $f(7)$  is  $\mathcal{H}_4 := \{K_7, \overline{K}_7, K_{3,4}, \overline{K}_{3,4}\}$ . Running SMS on  $D(\mathcal{H}_4, 16)$  and  $D(\mathcal{H}_4, 17)$  gives us 350 and 2772 solutions, respectively. We recover the templates from these solutions by applying  $\mathcal{D}'$ . Then, we filter out isomorphic templates using Nauty with the method explained in the Nauty and Traces User's Guide [20], which converts detecting isomorphisms between graphs with edge-colorings to that between graphs with vertex-colorings. This leaves us with 40 and 359 templates, respectively. Hence we have established the following result.

► **Proposition 5.**  $|\mathcal{H}_4^{(16)}| = 40$  and  $|\mathcal{H}_4^{(17)}| = 359$ .

## 5 Encodings for $k$ Universal Tournaments

### 5.1 Basic Encoding

For a fixed integer  $n$ , we consider a potential  $k$ -universal tournament  $G$  with  $V(G) = [n]$  whose arcs are represented by *arc variables*  $a_{u,v}$ ,  $u \neq v \in [n]$ . The truth value of  $a_{u,v}$  indicates whether there is an arc from  $u$  to  $v$ .

First, we specify a CNF formula  $F'(H, n)$  which is satisfiable if and only if  $H$  embeds into  $G$ , i.e., if  $H$  is isomorphic to a subtournament of  $G$ . The following CNF formula encodes the existence of a faithful monomorphism  $\phi$  from  $H$  to  $G$ , in a similar way  $F(H, n)$  encodes

the universality condition for universal graphs.

$$F'(H, n) := M(H, n) \wedge \bigwedge_{x \neq y \in [n]} ((\neg a_{x,y} \vee \neg a_{y,x}) \wedge (a_{x,y} \vee a_{y,x})) \wedge \bigwedge_{\substack{uv \in A(H) \\ 1 \leq x < y \leq n}} (\neg m_{u,x} \vee \neg m_{v,y} \vee a_{x,y}).$$

$M(H, n)$  ensure that  $(m_{u,v})_{u \in V(H), v \in V(G)}$  encodes an injective mapping from  $V(H)$  to  $V(G)$ . The second conjunct ensures that there is exactly one arc between any two distinct vertices. The third conjunct ensures that the mapping preserves arcs. The formula

$$U'(k, n) = \bigwedge_{H \in \mathcal{T}(k)} F'(H, n)$$

is satisfiable if and only if there exists a  $k$ -universal tournament on  $n$  vertices. From a satisfying assignment we can read off a  $k$ -universal tournament.

## 5.2 Approaches to Universal Tournaments

For the SMS approach, we use a recent extension of the framework to directed graphs [16].

For the SAT approach, there are two more features that we add to the basic encoding. The first feature is to hardcode an embedding of the transitive tournament onto  $[k]$ .

$$\text{TT}(k) = \bigwedge_{1 \leq i < j \leq k} a_{i,j}$$

The second feature is to utilize the perfect symmetry-breaking clauses from Lohn et al.'s work [18] who defined a series of CNF formulas  $I(t)$ ,  $1 \leq t \leq 8$ , which they call *isolators*. These isolators restrict arc variables to achieve a perfect symmetry-breaking for tournaments in the sense that the satisfying assignments to  $I(t)$  are in 1-1 correspondence with the tournaments of order  $t$  modulo isomorphism. Since the size of  $I(t)$  grows quickly in  $t$ , we only utilize isolators for  $t \leq 6$ .

Given  $l \leq r$ , we write  $I(l, r)$  to denote the isolator resulting from considering  $[l, r]$  instead of  $[r - l + 1]$  to be the underlying vertex set. To obtain the overall encoding  $\text{TTI}(k, n)$ , we start with  $\text{TT}(k)$  and see how many vertices there are in  $[k + 1, n]$ . If  $n - k \leq 6$ , then we add  $I(k + 1, n)$  to the encoding and finish. If  $n - k > 6$ , then we add  $I(k + 1, k + 6)$  to the encoding and repeat this process with  $[k + 7, n]$ . For example,  $\text{TTI}(6, 10) = \text{TT}(6) \wedge I(7, 10)$  and  $\text{TTI}(7, 16) = \text{TT}(7) \wedge I(8, 13) \wedge I(14, 16)$ .

## 6 Certificates

There is a long tradition of computer-assisted proofs in mathematics; a famous example is the proof of the Four-Color Theorem by Appel and Haken [3]. There are mathematical statements whose proof seems to require extensive brute-force search, which is impossible for a human to check [13]. For trusting mathematical statements established by such computational methods, certifying the computational result with a formal proof that can be independently checked is highly desirable. Proof checking should ideally be accomplished through a simple algorithm whose correctness can be trusted or even fully verified. Most of the SAT-based methods described in our paper can be independently verified. In particular, the runs of a CDCL SAT solver on our encodings can produce a DRAT proof, for which a verified proof checker DRAT-trim is available [12, 27]. This approach extends to SMS as follows [15]. During the SMS run, we collect all the generated symmetry-breaking clauses and add them to the clauses generated from the encoding. We can run a plain CDCL SAT solver on this extended set of clauses that generates a DRAT proof. The validity of the symmetry-breaking clauses can also be checked offline with a simple tool.

One could extend the our current verification tool chain in various ways. One can formally verify SAT encodings themselves [5, 6, 9] to provide additional trust in the obtained results. To certify the correctness of auxiliary computations provided by the Glasgow subgraph solver, one can utilize a Cutting Planes proof logging format [10]. What remains is the usage of Nauty [20] for generating pattern graphs and occasionally for filtering isomorphic copies, which does not support independent certification. The former usage, the generation of pattern graphs, can easily be replaced by SMS. For the latter, if an isomorphism between two objects has been found, it is easy to verify whether this is indeed a correct isomorphism. On the other hand, if Nauty fails to identify two objects as being isomorphic, this would only make our overall approach less efficient but would not alter the final results.

## 7 Experiments

### 7.1 Configurations

Based on the techniques discussed in Sections 4 and 5, we consider several configurations that we test in our experiments. We use a naming convention for configurations, where a name starting with UG indicates a configuration for universal graphs and a name starting with UT indicates a configuration for universal tournaments. All UG configurations contain the following basic encoding: the encoding for universality (Section 4.1), the symmetry breaking for twins (Section 4.2), and the hardcoded  $K_k, \overline{K_k}$  (Section 4.3). We distinguish between the use of SMS and the use of SAT without SMS (plain SAT).

---

<b>UG-SAT</b>	basic encoding with plain SAT.
<b>UG-SMS</b>	basic encoding with SMS.
<b>UG-SMS-GSS</b>	basic encoding with SMS extended with the Glasgow Subgraph Solver.
<b>UG-SAT-Temp</b>	basic encoding together with templates from Section 4.6 with plain SAT.
<b>UT-SMS</b>	encoding for universality from Section 5.1 with SMS.
<b>UT-SAT-TTI</b>	encoding for universality with fixed transitive tournament and isolators from Section 5.2 and plain SAT.

---

Theoretically, it is also possible to run SMS extended with GSS without the basic encoding. However, the preliminary experiments we conduct show that this is impractical given the drastically increased running time. Hence, we do not use this method in our main experiments.

### 7.2 Setup

The setup for the experiments is as follows. We generate  $\mathcal{G}(k)$  and  $\mathcal{T}(k)$  with Nauty 2.8.6 [20]. For configurations based on SMS, we use a recent SMS version that invoke a recent version of the Cadical SAT solver through the new IPASIR-UP interface [7]. For configurations that do not use SMS we employ Cadical as provided by the PySAT package [14]. We made minor changes to GSS [19] to access its internal functionality. All non-standard tools can be found in the supplementary material.

The experiments are run on a Sun Grid Engine (SGE) with Ubuntu 18.04.6 LTS. The architecture of the nodes in the SGE are among the following: 2× Intel Xeon E5540 with 2.53 GHz Quad Core, 2× Intel Xeon E5649 with 2.53 GHz 6-core, 2× Intel Xeon E5-2630 v2 with 2.60GHz 6-core, 2× Intel Xeon E5-2640 v4 with 2.40GHz 10-core and 2× AMD EPYC 7402 with 2.80GHz 24-core.

## 7.3 Universal Graphs

### 7.3.1 Verifying $f(k)$ and $F(k)$ for $1 \leq k \leq 5$

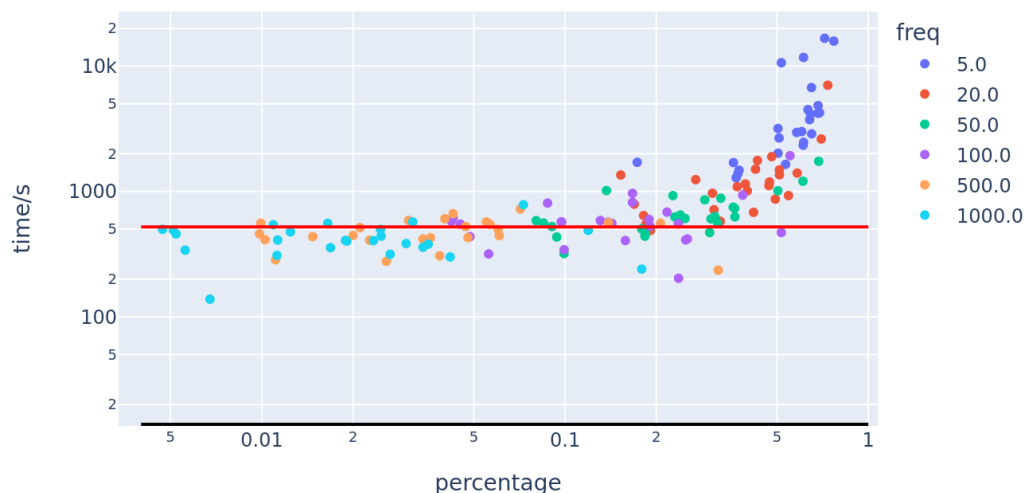
We run the configurations UG-SMS and UG-SAT-Temp to verify values  $f(k)$  and  $F(k)$  for  $1 \leq k \leq 5$  as obtained by Trimble [26]. To verify the values of  $f(k)$ , we run both configurations on  $n = f(k) - 1$  and  $n = f(k)$  to see whether the former gives a negative answer (unsat) and the latter a positive (sat). To verify the values of  $F(k)$ , we solve incrementally, i.e., every time a universal graph is discovered, a clause that prohibits the generation of this particular graph is added to the solver and the solver continues. For UG-SMS, we do not hardcode  $K_k$  and  $\overline{K}_k$  and start with  $([n])$  as the initial partition. This way, the minimality check is enough to filter out non-isomorphic solutions. For UG-SAT-Temp, we detect and exclude isomorphic copies among the found universal graphs using Nauty. It turns out, that both configurations can verify each of the vaules within a couple of seconds.

### 7.3.2 Comparing the Efficiency of Computing $f(6)$

The case of  $k = 6$  serves as a good benchmark for testing the efficiency of different methods, since it is neither too hard nor too easy. The value  $f(6) = 14$  was determined by Trimble [26]. Hence, we test various configurations on  $k = 6, n = 13$  and  $k = 6, n = 14$ . In the experiments described below, we run all instances 5 times and take the average running time.

For the case of  $k = 6, n = 13$ , we only test UG-SAT and UG-SMS since this case is computationally easy. The configurations use  $0.61s$  and  $1.39s$ , respectively.

For the case of  $k = 6, n = 14$ , we test UG-SAT, UG-SMS, and UG-SMS-GSS. The first two configurations use  $13.9s$  and  $521s$ , respectively. With UG-SMS-GSS, we try all possible combinations of the following parameter values:  $threshold \in \{40, 50, 60, 70, 80\}$ ,  $frequency \in \{5, 20, 50, 100, 500, 1000\}$ , and  $sample\ size \in \{5, 20, 60, 100, 156\}$ . We record the total time used in solving and the time used by the embedability check.



■ **Figure 3** Data points for UG-SMS-GSS on  $k = 6, n = 14$ , grouped according to the value of *frequency*. We cast the data points in a two-dimensional space where the *x*-axis represents the percentage of time taken by the embedability check, and the *y*-axis represents the total time used. We draw a red horizontal line standing for the case of SMS, and a black horizontal line standing for the case of pure SAT.

■ **Table 2** Running times for UG-SMS-GSS on the unsatisfiable case with  $k = 7$  and  $n = 16$ .

case	sample size	frequency	threshold	total time/h	time used in embedability check/h
1	1044	500	100	73.63	0.12
2	1044	1000	80	71.34	3.40
3	1044	500	70	82.36	11.63
4	1044	1000	110	76.34	0

To understand the parameters' influence on the running time, and how the three configurations compare with each other, we present our findings in Figure 3. We first observe that UG-SAT uses significantly less time than UG-SMS or UG-SMS-GSS. Second, the introduction of the embedability check slightly decreases the running time in some cases and drastically increases the running time in others. Third, the solving time positively correlates with the percentage of the time used by the embedability check. In general, UG-SAT is clearly the most efficient among the configurations and the benefit of UG-SMS-GSS compared to UG-SMS is not obvious.

In Figure 3, data points of different *frequency* are shown in different colors. We see that data points of the same color form loose clusters. A further check into the distribution in terms of sample size and threshold within each group does not yield any meaningful pattern. We also tried grouping the same data points according to *sample size* and *threshold*, but did not observe any obvious patterns. This indicates that the *frequency* parameter might be the only useful one among the three in terms of tuning for efficiency, and giving it a reasonably large value (which corresponds to the infrequent invocation of the embedability check) is preferred.

We also tried to determine the value of  $F(6)$  incrementally with UG-SAT. After a few days we terminated the program. At that state, it had generated 36294 non-isomorphic 6-universal graphs of order 14.

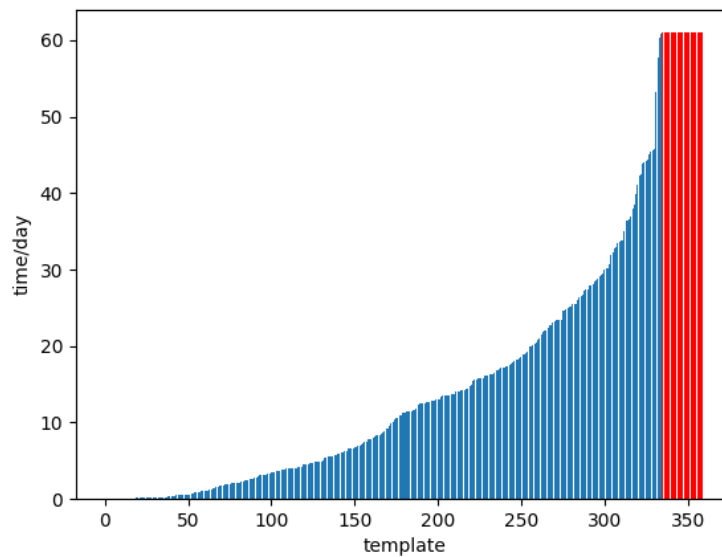
### 7.3.3 $f(7) > 16$

To show  $f(7) > 16$ , we use UG-SAT, UG-SMS, UG-SMS-GSS, and UG-SAT-Temp. All four configurations conclude that a 7-universal graph with 16 vertices does not exist. UG-SAT takes 39 hours and UG-SMS takes 68 hours. For UG-SMS-GSS, we narrow down the choices of the possible values for the parameters based on previous experiments, and test all combinations with *sample size* = 1044, *frequency*  $\in \{500, 1000\}$  and *threshold*  $\{70, 80, 90, 100, 110\}$ . Only 4 test cases out of 50 terminate within 97 hours. Table 2 provides some details. For UG-SAT-Temp, there are  $|\mathcal{H}_4^{(16)}| = 40$  templates to consider. The easiest template takes 0.4s, the hardest 38 minutes. The total time spent on all 40 templates is 3.6 hours, making the average time spent on each template 322.4s.

Since UG-SAT-Temp seems to be the most effective among the considered configurations, we also tried using it to determine whether there exists a 7-universal graph of order 17. Here we have  $|\mathcal{H}_4^{(17)}| = 359$  templates to consider. Unfortunately, this approach does not seem to be able to bring the problem down to the reach of modern SAT solvers. By the time of the writing, 25 out of the 359 cases still have not terminated. The 334 cases that have terminated spent 4237 CPU days, all returning with a negative answer. Figure 4 shows the time spent on each template in an ascending order. Even though we have no definitive result, the facts that (i) most cases could be shown unsatisfiable, and (ii) the SAT solver was not able to find a counter example on the remaining cases even after a long runtime, provide some evidence that a 7-universal graph of order 17 might not exist.

■ **Table 3** Overview of running times in seconds for finding one or all solutions up to isomorphism.

$k$	$\mathcal{T}(k)$	$t(k)$	$T(k)$	$t(k) - 1$		$t(k)$			
						UT-SAT-TTI		UT-SMS	
				UT-SAT-TTI	UT-SMS	one	all	one	all
5	12	8	1643	0.03	0.17	0.00	48.28	0.06	3.27
6	56	10	1088	6.41	30.68	11.83	N/a	105.36	47033.06



■ **Figure 4** Time spent on each template ordered by their value. The blue bars represent the time spent by templates that have terminated, and the red bars show the time spent on templates that have not terminated by the time of writing.

## 7.4 Universal Tournaments

### 7.4.1 $t(k)$ and $T(k)$ for $1 \leq k \leq 6$

We run the configurations UT-SMS and UT-SAT-TTI to calculate  $t(k)$  and  $T(k)$  for  $1 \leq k \leq 6$ . To calculate the value of  $f(k)$ , we start with the lower bound

$$t(k) \geq \max(\{k, t(k-1), \min\{x \mid C_k^x \geq |\mathcal{T}(k)|\}\})$$

where  $C_k^x$  denotes the number of  $k$ -combinations from a given set of  $x$  elements, i.e.,  $C_k^x = \frac{x!}{k!(x-k)!}$ . We test  $\text{TTI}(k, n)$  while increasing  $n$  until we find a universal tournament. To calculate  $T(k)$ , we modify the two methods similarly as we did for verifying  $F(k)$ . Table 3 shows the time spent on the case  $n = t(k) - 1$  and  $n = t(k)$ . The time consumed in all applicable cases for  $0 \leq k \leq 4$  are under 0.005s, so we omit them in the table. The process of finding out all non-isomorphic 6-universal tournaments of order 10 with UT-SAT-TTI does not terminate within a couple of days.

### 7.4.2 $13 \leq t(7) \leq 15$

An obvious lower bound for  $t(7)$  is 11 since  $C_7^{10} = 120 < 456 = |\mathcal{T}(7)|$ . Neither UT-SMS nor UT-SAT-TTI gives a result on  $k = 7, n = 11$  within half an hour. We generate separate SAT instances where we hardcode each of the four non-isomorphic tournaments of order 4 onto  $[8, 11] \subseteq V(G)$  in addition to TTI(7, 11). The four instances terminate with a negative answer in 14.55s, 458.32s, 1399.09s, and 1212.89s, respectively. This way, we have shown that  $t(7) > 11$ . To show that  $t(7) > 12$ , we generate and test the following SAT instance for each  $\vec{a} = (a_i)_{i \in [7]}, a_i \in [4]$ ,

$$\text{TTI}(7, 12) \wedge \bigvee_{i \in [7]} \psi_{a_i}(i)$$

where

$$\begin{aligned} \psi_1(i) &:= |\{j \in [8, 12] \mid e_{i,j} = 1\}| \leq 1, \quad \psi_2(i) := |\{j \in [8, 12] \mid e_{i,j} = 1\}| = 2 \\ \psi_3(i) &:= |\{j \in [8, 12] \mid e_{i,j} = 1\}| = 3, \quad \text{and } \psi_4(i) := |\{j \in [8, 12] \mid e_{i,j} = 1\}| \geq 4. \end{aligned}$$

This gives us 16384 instances in total and all of them return with a negative answer. Thus,  $t(7) > 12$ . The sum of the time spent is around 2002 CPU days, making the average running time for each case 2.9 hours. More than half of the cases terminate within 40 minutes, and the hardest case takes almost 4 days.

The upper bound  $t(7) \leq 15$  is obtained with UT-SAT-TTI, which finds a 7-universal tournament of order 15 within 57.95s. As for the cases of TTI(7, 13) and TTI(7, 14), UT-SAT-TTI does not terminate within a few days, and it is not practical to generate separate SAT instances in the way we did for TTI(7, 12), given the large number of instances and the estimated time per case from our preliminary experimentation.

## 8 Conclusion

We have proposed several methods to compute the smallest order of induced  $k$ -universal graphs and tournaments as a synthesis problem that does not require the enumeration of all potential candidates. Our approaches make use of state-of-the-art CDCL SAT solvers. For the SAT encoding, we proposed a direct encoding and a lazy encoding that utilizes the Glasgow subgraph solver (GSS); for symmetry breaking, we proposed using the SAT modulo Symmetries (SMS) framework and a new templates approach.

We tested and compared these methods, which let us understand their strengths and weaknesses. The template approach was the fastest for unsatisfiable instances; SMS was particularly strong for enumerating all solutions; the GSS was able to speed up the SMS approach. We could improve the known lower bound and show that no induced 7-universal graph with 16 vertices exists, leaving possibilities 17 and 18 for the smallest solution.

We could determine the exact order of smallest  $k$ -universal tournaments up to  $k = 6$  and the interval [13,15] for  $k = 7$ . We determined the precise number of optimal solutions, which is of independent interest for combinatorial research.

As future work, determining the exact order of a smallest induced 7-universal graph might be within reach by refining our methods and additional cubing. It would be interesting to see whether it is possible to combine SMS with the templates approach, as this might give a significant boost to possibly allow us to attack the  $k$ -universality problems for even larger values of  $k$ .

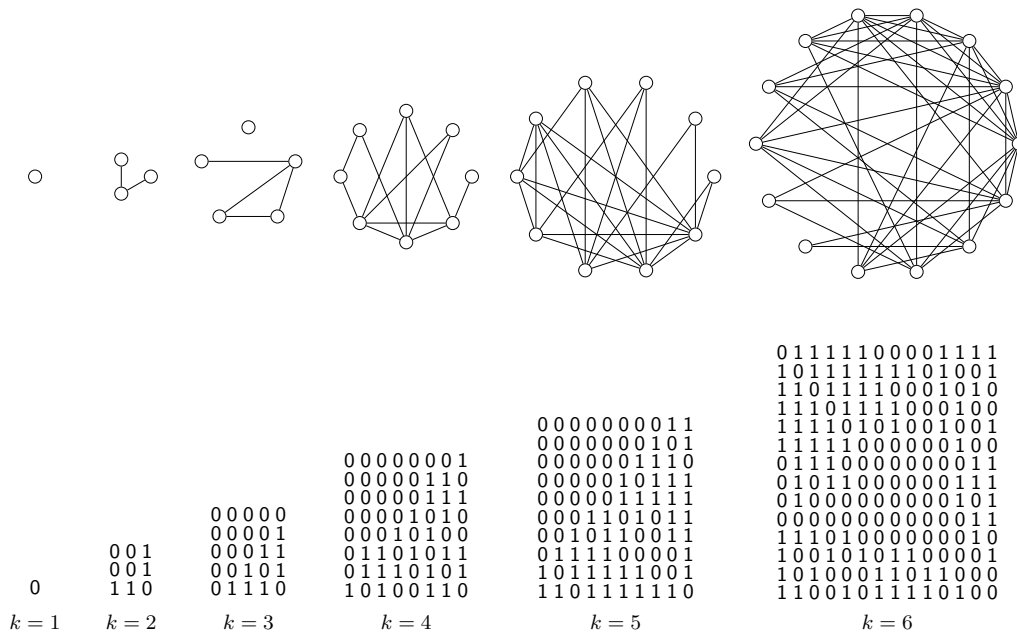
## References

- 1 Noga Alon. Asymptotically optimal induced universal graphs. *Geom. Funct. Anal.*, 27(1):1–32, 2017. doi:10.1007/s00039-017-0396-9.
- 2 Stephen Alstrup, Haim Kaplan, Mikkel Thorup, and Uri Zwick. Adjacency labeling schemes and induced-universal graphs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 625–634. ACM, 2015. doi:10.1145/2746539.2746545.
- 3 Kenneth Appel and Wolfgang Haken. The solution of the four-color-map problem. *Sci. Amer.*, 237(4):108–121, 152, 1977. doi:10.1038/scientificamerican1077-108.
- 4 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020. URL: [https://tuhat.helsinki.fi/ws/files/142452772/sc2020\\_proceedings.pdf](https://tuhat.helsinki.fi/ws/files/142452772/sc2020_proceedings.pdf).
- 5 Cayden R. Codel. Verifying SAT encodings in lean. Master’s thesis, Computer Science Department School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, May 2022. URL: <http://reports-archive.adm.cs.cmu.edu/anon/2022/CMU-CS-22-106.pdf>.
- 6 Luís Cruz-Filipe, João Marques-Silva, and Peter Schneider-Kamp. Formally verifying the solution to the boolean Pythagorean triples problem. *Journal of Automated Reasoning*, 63(3):695–722, 2019. doi:10.1007/s10817-018-9490-4.
- 7 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. IPASIR-UP: User propagators for CDCL. In Meena Mahajan and Friedrich Slivovsky, editors, *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), July 04-08, 2023, Alghero, Italy*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- 8 Johannes K. Fichte, Daniel Le Berre, Markus Hecher, and Stefan Szeider. The silent (r)evolution of SAT. *Communications of the ACM*, 66(6):64–72, June 2023. doi:10.1145/3560469.
- 9 Sofia Giljegård and Johan Wennerbeck. Puzzle solving with proof—writing a verified SAT encoding chain in HOL4. Master’s thesis, Chalmers University of Technology and University of Gothenburg, 2021. URL: <https://hdl.handle.net/20.500.12380/304104>.
- 10 Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1134–1140. ijcai.org, 2020. doi:10.24963/ijcai.2020/158.
- 11 Geña Hahn and Claude Tardif. Graph homomorphisms: structure and symmetry. In Geña Hahn and Gert Sabidussi, editors, *Graph Symmetry*, pages 107–166. Kluwer Academic Publishers, Dordrecht, 1997.
- 12 Marijn Heule, Warren A. Hunt Jr., Matt Kaufmann, and Nathan Wetzler. Efficient, verified checking of propositional proofs. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *Lecture Notes in Computer Science*, pages 269–284. Springer Verlag, 2017. doi:10.1007/978-3-319-66107-0\_18.
- 13 Marijn J. H. Heule and Oliver Kullmann. The science of brute force. *Communications of the ACM*, 60(8):70–79, 2017. doi:10.1145/3107239.
- 14 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018. doi:10.1007/978-3-319-94144-8\_26.
- 15 Markus Kirchweger, Manfred Scheucher, and Stefan Szeider. A SAT attack on Rota’s Basis Conjecture. In *Theory and Applications of Satisfiability Testing - SAT 2022 - 25th International Conference, Haifa, Israel, August 2-5, 2022, Proceedings*, 2022. doi:10.4230/LIPIcs.SAT.2022.4.

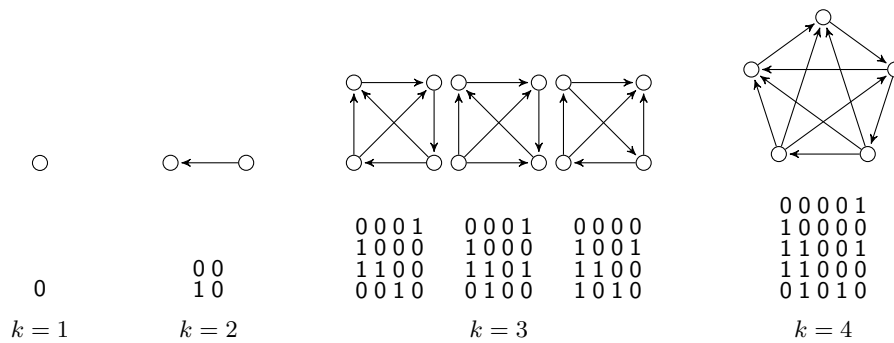


- 16 Markus Kirchweger, Manfred Scheucher, and Stefan Szeider. SAT-based generation of planar graphs. In Meena Mahajan and Friedrich Slivovsky, editors, *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), July 04-08, 2023, Alghero, Italy*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- 17 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 39:1–39:17. Dagstuhl, 2021. doi:10.4230/LIPIcs.CP.2021.34.
- 18 Evan Lohn, Chris Lambert, and Marijn J. H. Heule. Compact symmetry breaking for tournaments. In Alberto Griggio and Neha Rungta, editors, *22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022*, pages 179–188. IEEE, 2022. doi:10.34727/2022/isbn.978-3-85448-053-2\_24.
- 19 Ciaran McCreesh, Patrick Prosser, and James Trimble. The Glasgow subgraph solver: Using constraint programming to tackle hard subgraph isomorphism problem variants. In Fabio Gadducci and Timo Kehrer, editors, *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings*, volume 12150 of *Lecture Notes in Computer Science*, pages 316–324. Springer, 2020. doi:10.1007/978-3-030-51372-6\_19.
- 20 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symbolic Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.
- 21 Jhon W. Moon. On minimal  $n$ -universal graphs. *Proc. Glasgow Math. Assoc.*, 7:32–33 (1965), 1965. doi:10.1017/S2040618500035139.
- 22 John W. Moon. *Topics on tournaments*. Holt, Rinehart and Winston, New York-Montreal, Que.-London, 1968.
- 23 OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences. Published electronically at <http://oeis.org>.
- 24 James Preen. What is the smallest graph that contains all non-isomorphic 4-node and 5-node connected graphs as induced subgraphs? online, October 2011. Mathematics Stack Exchange, <https://math.stackexchange.com/q/75513>, last edited 2020-04-25.
- 25 Richard Rado. Universal graphs and universal functions. *Acta Arithmetica*, 9(4):331–340, 1964. URL: <http://eudml.org/doc/207488>.
- 26 James Trimble. Induced universal graphs for families of small graphs. *CoRR*, abs/2109.00075, 2021. doi:10.48550/arXiv.2109.00075.
- 27 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing – SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer Verlag, 2014. doi:10.1007/978-3-319-09284-3\_31.

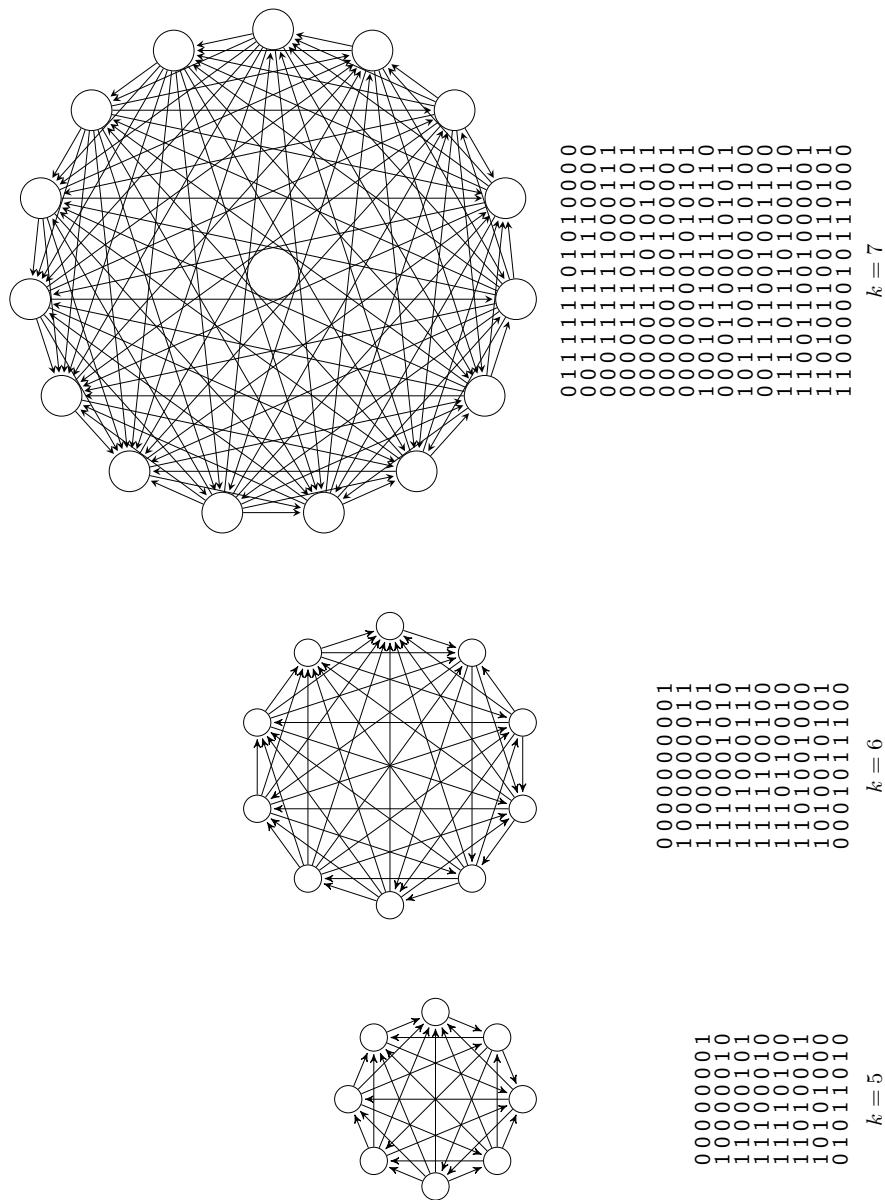
**A** Appendix



**Figure 5** Smallest induced  $k$ -universal graphs and their incidence matrices.



**Figure 6** All smallest  $k$ -universal tournaments for  $k = 1, 2, 3, 4$ , and their incidence matrices.



**Figure 7** The three displayed tournaments are examples of  $k$ -universal tournaments of order 8, 10, and 15 for  $k = 5$ ,  $k = 6$ , and  $k = 7$ , respectively. We found many more with this property. We also show their incidence matrices. For  $k = 5, 6$  we know that there are no  $k$ -universal tournaments of smaller order, for  $k = 7$  we cannot rule out that there is one of order 14.

# FastMapSVM for Predicting CSP Satisfiability

Kexin Zheng<sup>1</sup> ✉

University of Southern California, Los Angeles, CA, USA

Ang Li<sup>1</sup> ✉

University of Southern California, Los Angeles, CA, USA

Han Zhang ✉

University of Southern California, Los Angeles, CA, USA

T. K. Satish Kumar ✉

University of Southern California, Los Angeles, CA, USA

---

## Abstract

Recognizing the satisfiability of Constraint Satisfaction Problems (CSPs) is NP-hard. Although several Machine Learning (ML) approaches have attempted this task by casting it as a binary classification problem, they have had only limited success for a variety of challenging reasons. First, the NP-hardness of the task does not make it amenable to straightforward approaches. Second, CSPs come in various forms and sizes while many ML algorithms impose the same form and size on their training and test instances. Third, the representation of a CSP instance is not unique since the variables and their domain values are unordered. In this paper, we propose FastMapSVM, a recently developed ML framework that leverages a distance function between pairs of objects. We define a novel distance function between two CSP instances using maxflow computations. This distance function is well defined for CSPs of different sizes. It is also invariant to the ordering on the variables and their domain values. Therefore, our framework has broader applicability compared to other approaches. We discuss various representational and combinatorial advantages of FastMapSVM. Through experiments, we also show that it outperforms other state-of-the-art ML approaches.

**2012 ACM Subject Classification** Computing methodologies → Machine learning

**Keywords and phrases** Constraint Satisfaction Problems, Machine Learning, FastMapSVM

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.40

**Funding** This work at the University of Southern California is supported by DARPA under grant number HR001120C0157 and by NSF under grant number 2112533. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the sponsoring organizations, agencies, or the U.S. Government.

## 1 Introduction

Constraints constitute a very natural and general means for formulating regularities in the real world. A fundamental combinatorial structure used for reasoning with constraints is that of the Constraint Satisfaction Problem (CSP). The CSP formally models a set of variables, their corresponding domains, and a collection of constraints between subsets of the variables. Each constraint restricts the set of allowed combinations of values of the participating variables. A solution of a given CSP instance is an assignment of values to all the variables from their respective domains such that all the constraints are satisfied. Technologies for efficiently solving CSPs bear immediate and important implications on how fast we can solve computational problems that arise in several other areas of research, including computer vision, spatial and temporal reasoning, model-based diagnosis, planning and scheduling, and language understanding.

---

<sup>1</sup> Corresponding Author



© Kexin Zheng, Ang Li, Han Zhang, and T. K. Satish Kumar;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 40; pp. 40:1–40:17

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Unfortunately, solving CSPs is NP-hard since they generalize the Satisfiability (SAT) problem. Although many technologies have been developed for solving CSPs in practice [4], they do not sufficiently harness the power of Machine Learning (ML) techniques. While there have been a lot of attempts to apply ML techniques to CSPs, none of these attempts have yielded spectacular results: They do not consistently produce high-quality outcomes. Examples of ML approaches used in the CSP domain include the application of Support Vector Machines (SVMs) [2], linear regression [22], decision tree learning [7, 6], clustering [15, 8],  $k$ -nearest neighbors [13], and others [9]. However, most of these approaches have had limited success for a variety of challenging reasons.

First, from a complexity theory perspective, the NP-hardness of the task does not make it amenable to straightforward ML approaches. For example, since a neural network (NN) is essentially a continuous differentiable form of a circuit, it is not straightforward to make NNs effective in the CSP domain. Second, CSPs come in various forms and sizes while many ML algorithms use an architectural framework that imposes the same form and size on their training and test instances. For example, an NN may have a fixed input layer that it uses for the training and test instances alike. Third, the representation of a CSP instance is not unique since the variables and their domain values are unordered. This poses a significant combinatorial challenge for ML algorithms since they have to learn the permutation invariance with respect to orderings on the variables and their domain values. For example, an NN may pose the overhead of having to be trained on all permutations of the same CSP instance to become effective.

In this paper, we consider the problem of predicting the satisfiability of CSP instances using ML. In ML terminology, this is essentially a binary classification problem defined on CSPs with the two possible classification labels “satisfiable” and “unsatisfiable”. This classification problem is a cornerstone task for addressing the combinatorics of CSPs. It also serves as a stepping stone for the task of solving CSPs. In fact, any ML framework expected to be viable for solving CSPs should likely first demonstrate its success on solving the aforementioned classification problem on CSPs.

We propose to solve the above classification problem on binary CSPs<sup>2</sup> using a recently developed ML framework called FastMapSVM [19]. While most ML algorithms learn to identify characteristic features of *individual* objects in a class, FastMapSVM leverages a domain-specific distance function on *pairs* of objects. It does this by combining the strengths of FastMap [5] and SVMs. In its first stage, FastMapSVM invokes FastMap, an efficient linear-time algorithm that maps complex objects to points in a Euclidean space, while preserving pairwise distances between them. In its second stage, it invokes SVMs and kernel methods for learning to classify the points in this Euclidean space.

FastMapSVM has demonstrated success on classifying complex objects such as seismograms in Earthquake Science [19]. It offers several advantages over ML algorithms that reason about individual objects instead of pairs of objects. First, FastMapSVM enables domain experts to incorporate their domain knowledge using a distance function. This avoids relying on complex ML models to infer the underlying structure in the data entirely. Second, because the distance function encapsulates domain knowledge, FastMapSVM naturally facilitates interpretability and explainability. In fact, it even provides a perspicuous visualization of the objects and the classification boundaries between them. Third, FastMapSVM uses significantly smaller amounts of time and data for model training compared to other ML algorithms. Fourth, it extends the applicability of SVMs and kernel methods to domains with complex objects.

---

<sup>2</sup> Binary CSPs have at most two variables per constraint but are allowed to have non-Boolean variables. Binary CSPs are representationally as powerful as general CSPs with bounded arity of the constraints.

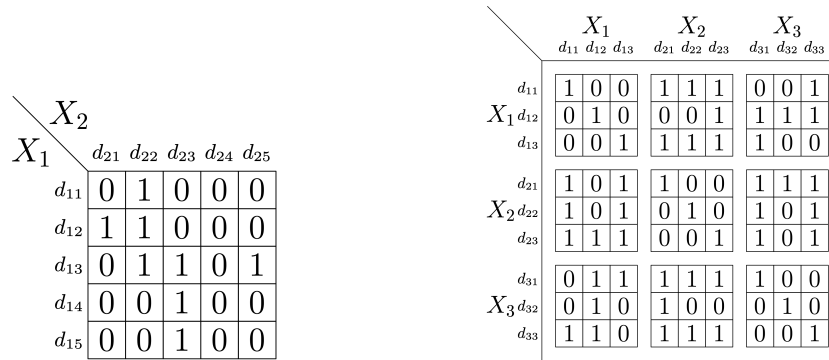


Figure 1 The left panel shows the (0, 1)-matrix representation of a single constraint  $C(X_1, X_2)$ . The right panel shows the (0, 1)-matrix representation of an entire binary CSP instance.

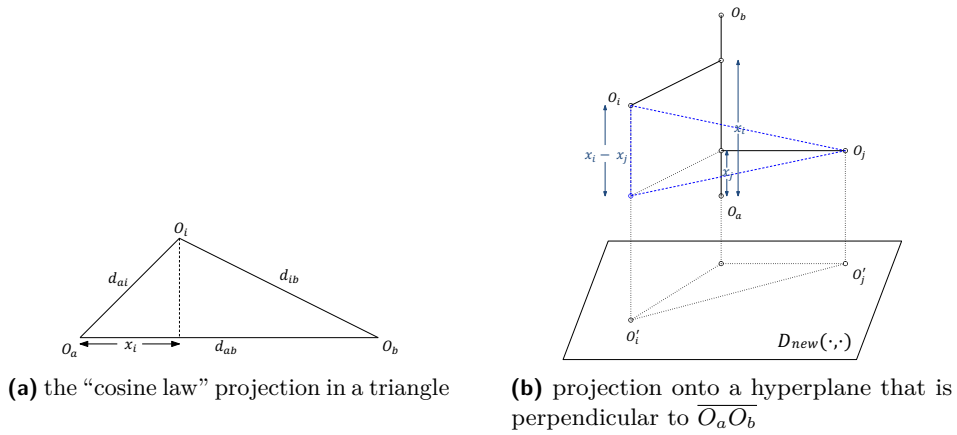
In applying FastMapSVM to the CSP domain, we define a novel distance function between two CSP instances. This distance function uses maxflow computations and is well defined for CSP instances of different sizes. It is also invariant to the ordering on the variables and their domain values in the CSP instances. Therefore, FastMapSVM has broader applicability compared to other ML approaches in the CSP domain. Moreover, since it uses the intelligence of SVMs, kernel methods, and maxflow computations, it is able to significantly outperform competing ML approaches. It is also able to outperform procedures that invest polynomial time in establishing local consistency—such as arc-consistency—to discover unsatisfiable CSP instances. This demonstrates that a trained FastMapSVM model acquires an intelligence beyond that of prominent polynomial-time procedures.<sup>3</sup> We discuss various other representational and combinatorial advantages of FastMapSVM and, through experiments, we also demonstrate its superior performance.

## 2 Preliminaries and Definitions

A CSP instance is defined by a triplet  $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ , where  $\mathcal{X} = \{X_1, X_2 \dots X_N\}$  is a set of variables and  $\mathcal{C} = \{C_1, C_2 \dots C_M\}$  is a set of constraints on subsets of them. Each variable  $X_i$  is associated with a finite discrete-valued domain  $D_i \in \mathcal{D}$ , and each constraint  $C_i$  is a pair  $\langle S_i, R_i \rangle$  defined on a subset of variables  $S_i \subseteq \mathcal{X}$ , called the *scope* of  $C_i$ .  $|S_i|$  is referred to as the *arity* of the constraint.  $R_i \subseteq D_{S_i}$  ( $D_{S_i} = \times_{X_j \in S_i} D_j$ ) denotes all compatible tuples of  $D_{S_i}$  allowed by the constraint. The absence of a constraint on a certain subset of the variables is equivalent to a constraint on the same subset of the variables that allows all combinations of values to them. A *solution* of a CSP instance is an assignment of values to all the variables from their respective domains such that all the constraints are satisfied. A binary CSP instance has at most two variables per constraint. Binary CSPs are representationally as powerful as general CSPs with bounded arity of the constraints [4].

A binary CSP is *arc-consistent* if and only if for all variables  $X_i$  and  $X_j$ , and for every instantiation of  $X_i$ , there exists an instantiation of  $X_j$  such that the direct constraint between them is satisfied. Similarly, a binary CSP is *path-consistent* if and only if for all variables

<sup>3</sup> This is an important hallmark of an ML algorithm. In [20], a deep NN model is presented to recognize the satisfiability of CSP instances with Boolean variables and binary constraints. However, this class of CSP instances is equivalent to 2-SAT and can be solved in polynomial time, diminishing the advantages of an ML framework over polynomial-time reasoning.



■ **Figure 2** The figure, borrowed from [3], illustrates how coordinates are computed and recursion is carried out in FastMap.

$X_i$ ,  $X_j$  and  $X_k$ , and for every instantiation of  $X_i$  and  $X_j$  that satisfies the direct constraint between them, there exists an instantiation of  $X_k$  such that the direct constraints between  $X_i$  and  $X_k$  and between  $X_j$  and  $X_k$  are also satisfied.

For a given binary CSP instance, we can build a matrix representation for it using a simple mechanism. First, we assume that the domain values of each variable are ordered in some way. (We can simply use the order in which the domain values of each of the variables are specified.) Under such an ordering, we can represent each binary constraint as a 2-dimensional matrix with all its entries set to either 1 or 0 based on whether the corresponding combination of values to the participating variables is allowed or not by that constraint. The left panel of Figure 1 shows the  $(0, 1)$ -matrix representation of a binary constraint between two variables  $X_1$  and  $X_2$  with domain sizes of 5 each. The combination of values ( $X_1 \leftarrow d_{12}, X_2 \leftarrow d_{21}$ ) is an allowed combination, and the corresponding entry in the matrix is therefore set to 1. However, the combination of values ( $X_1 \leftarrow d_{14}, X_2 \leftarrow d_{22}$ ) is a disallowed combination, and the corresponding entry is therefore set to 0. In general,  $d_{ip}$  denotes the  $p$ -th domain value of  $X_i$  assuming an index ordering on the domain values of  $X_i$ .

The  $(0, 1)$ -matrix representation of an entire binary CSP instance can be constructed simply by stacking up the matrix representations for the individual constraints into a bigger “block” matrix. The right panel of Figure 1 illustrates how a binary CSP instance on 3 variables  $X_1$ ,  $X_2$ , and  $X_3$  can be represented as a “mega-matrix” with 3 sets of rows and 3 sets of columns. Each block-entry inside this mega-matrix is the matrix representation of the direct constraint between the corresponding row and column variables. Therefore, the matrix representation of an entire binary CSP instance has  $\sum_{i=1}^N |D_i|$  rows and  $\sum_{i=1}^N |D_i|$  columns.

### 3 FastMap and FastMapSVM

In this section, we describe FastMap and FastMapSVM to set up the groundwork for our approach. Both these rely on a domain-specific distance function  $D(\cdot, \cdot)$  on pairs of objects.  $D(\cdot, \cdot)$  is required to be a non-negative symmetric function.

### 3.1 FastMap

FastMap [5] is a Data Mining algorithm that embeds complex objects—like audio signals, seismograms, DNA sequences, electrocardiograms, or magnetic-resonance images—into a  $K$ -dimensional Euclidean space, for a user-specified value of  $K$  and a user-supplied distance function  $D(\cdot, \cdot)$  on pairs of objects. The Euclidean distance between any two objects in the embedding approximates the domain-specific distance between them. Therefore, similar objects, as quantified by  $D(\cdot, \cdot)$ , map to nearby points in Euclidean space while dissimilar objects map to distant points. Although FastMap preserves  $O(N^2)$  pairwise distances between  $N$  objects, it generates the embedding in only  $O(KN)$  time. Because of its efficiency, FastMap has already found numerous real-world applications, including in Data Mining [5], shortest-path computations [3], solving combinatorial optimization problems on graphs [12], and community detection and block modeling [11].

FastMap embeds a collection of complex objects in an artificially created Euclidean space that enables geometric interpretations, algebraic manipulations, and downstream ML algorithms. It gets as input a collection of complex objects  $\mathcal{O}$ , where  $D(O_i, O_j)$  represents the domain-specific distance between objects  $O_i, O_j \in \mathcal{O}$ . It generates a Euclidean embedding that assigns a  $K$ -dimensional point  $\mathbf{p}_i \in \mathbb{R}^K$  to each object  $O_i$ . A good Euclidean embedding is one in which the Euclidean distance  $\|\mathbf{p}_j - \mathbf{p}_i\|_2$  between any two points  $\mathbf{p}_i$  and  $\mathbf{p}_j$  closely approximates  $D(O_i, O_j)$ .

In the first iteration, FastMap heuristically identifies the farthest pair of objects  $O_a$  and  $O_b$  in linear time. Once  $O_a$  and  $O_b$  are determined, every other object  $O_i$  defines a triangle with sides of lengths  $d_{ai} = D(O_a, O_i)$ ,  $d_{ab} = D(O_a, O_b)$ , and  $d_{ib} = D(O_i, O_b)$ , as illustrated in Figure 2a. The lengths of the sides of the triangle define its entire geometry, and the projection of  $O_i$  onto the line  $\overline{O_a O_b}$  is given by

$$x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2) / (2d_{ab}). \quad (1)$$

FastMap sets the first coordinate of  $\mathbf{p}_i$ , the embedding of  $O_i$ , equal to  $x_i$ . In the subsequent  $K - 1$  iterations, the same procedure is followed for computing the remaining  $K - 1$  coordinates of each object; however, the distance function is adapted for each iteration. For example, for the first iteration, the coordinates of  $O_a$  and  $O_b$  are 0 and  $d_{ab}$ , respectively. Because these coordinates fully explain the true distance between these two objects, from the second iteration onward, the rest of  $\mathbf{p}_a$  and  $\mathbf{p}_b$ 's coordinates should be identical. Intuitively, this means that the second iteration should mimic the first one on a hyperplane that is perpendicular to the line  $\overline{O_a O_b}$ . Figure 2b illustrates this. Although the hyperplane is never explicitly constructed, it conceptually implies that the distance function for the second iteration should be changed for all  $i$  and  $j$  in the following way:

$$D_{new}(O'_i, O'_j)^2 = D(O_i, O_j)^2 - (x_i - x_j)^2, \quad (2)$$

in which  $O'_i$  and  $O'_j$  are the projections of  $O_i$  and  $O_j$ , respectively, onto this hyperplane, and  $D_{new}(\cdot, \cdot)$  is the new distance function. The distance function is recursively updated according to Equation 2 at the beginning of each of the  $K - 1$  iterations that follow the first.

In each of the  $K$  iterations, FastMap heuristically finds the farthest pair of objects according to the distance function defined for that iteration. These objects are called pivots and are stored as reference objects. There are very few, that is,  $\leq 2K$ , reference objects. Technically, finding the farthest pair of objects in any iteration takes  $O(N^2)$  time. However, FastMap uses a linear-time “pivot changing” heuristic [5] to efficiently and effectively identify a pair of objects  $O_a$  and  $O_b$  that is very often the farthest pair. It does this by initially



choosing a random object  $O_b$  and then choosing  $O_a$  to be the farthest object away from  $O_b$ . It then reassigns  $O_b$  to be the farthest object away from  $O_a$ , reassigns  $O_a$  to be the farthest object away from  $O_b$ , and so on, until convergence or a maximum of  $C$  iterations, for a small constant  $C \leq 10$ .

### 3.2 FastMapSVM

FastMapSVM [19] elegantly combines the strengths of FastMap and SVMs. In the first phase, it invokes FastMap for efficiently mapping complex objects to points in a Euclidean space, while preserving pairwise distances between them. In the second phase, it invokes SVMs and kernel methods for learning to classify the points in this Euclidean space. FastMapSVM has several advantages over other methods.

First, FastMapSVM leverages domain-specific knowledge via a distance function. There are many real-world domains in which feature selection for *individual* objects is hard. While a domain expert can occasionally identify and incorporate domain-specific features of the objects to be classified, doing so becomes increasingly hard with increasing complexity of the objects. Therefore, many existing ML algorithms for classification find it hard to leverage domain knowledge when used off the shelf. However, in many real-world domains with complex objects, a distance function on *pairs* of objects is well defined and easy to compute. In such domains, FastMapSVM is more easily applicable than other ML algorithms that focus on the features of individual objects. FastMapSVM also enables domain experts to incorporate their domain knowledge via a distance function instead of relying on complex ML models to infer the underlying structure in the data entirely. Examples of such real-world objects include audio signals, seismograms, DNA sequences, electrocardiograms, and magnetic-resonance images. While these objects are complex and may have many subtle features that are hard to recognize, there exists a well-defined distance function on pairs of objects that is easy to compute. For instance, individual DNA sequences have many complex and subtle features but the *edit distance*<sup>4</sup> between two DNA sequences is well defined and easy to compute. Similarly, the Minkowski distance [1] is well defined for images and the cosine similarity [16] is well defined for text documents.

Second, FastMapSVM facilitates interpretability, explainability, and visualization. Many existing ML algorithms produce results that are hard to interpret or explain. For example, in NNs, a large number of interactions between neurons with nonlinearities makes a meaningful interpretation or explanation of the results very hard. In fact, the very complexity of the objects in the domain can hinder interpretability and explainability. FastMapSVM mitigates these challenges and thereby supports interpretability and explainability. While the objects themselves may be complex, FastMapSVM embeds them in a Euclidean space by considering only the distance function defined on pairs of objects. In effect, it simplifies the description of the objects by assigning Euclidean coordinates to them. Moreover, since the distance function is itself user-supplied and encapsulates domain knowledge, FastMapSVM naturally facilitates interpretability and explainability. In fact, it even provides a perspicuous visualization of the objects and the classification boundaries between them. This aids human interpretation of the data and results. It also enables a human-in-the-loop framework for refining the processes of learning and decision making. As a hallmark, FastMapSVM produces the visualization very efficiently since it invests only linear time in generating the Euclidean embedding.

---

<sup>4</sup> The edit distance between two strings is the minimum number of insertions, deletions, or substitutions that are needed to transform one to the other.

Third, FastMapSVM uses significantly smaller amounts of time and data for model training compared to other ML algorithms. While NNs and other ML algorithms store abstract representations of the training data in their model parameters, FastMapSVM stores explicit references to some of the original objects, referred to as pivots. While making predictions, objects in the test instances are compared directly to the pivots using the user-supplied distance function. Thereby, FastMapSVM obviates the need to learn a complex transformation of the input data and thus significantly reduces the amounts of time and data required for model training. Moreover, given  $N$  training instances, that is,  $N$  objects and their classification labels, FastMapSVM leverages  $O(N^2)$  pieces of information via the distance function that is defined on every pair of objects. In contrast, ML algorithms that focus on individual objects leverage only  $O(N)$  pieces of information.

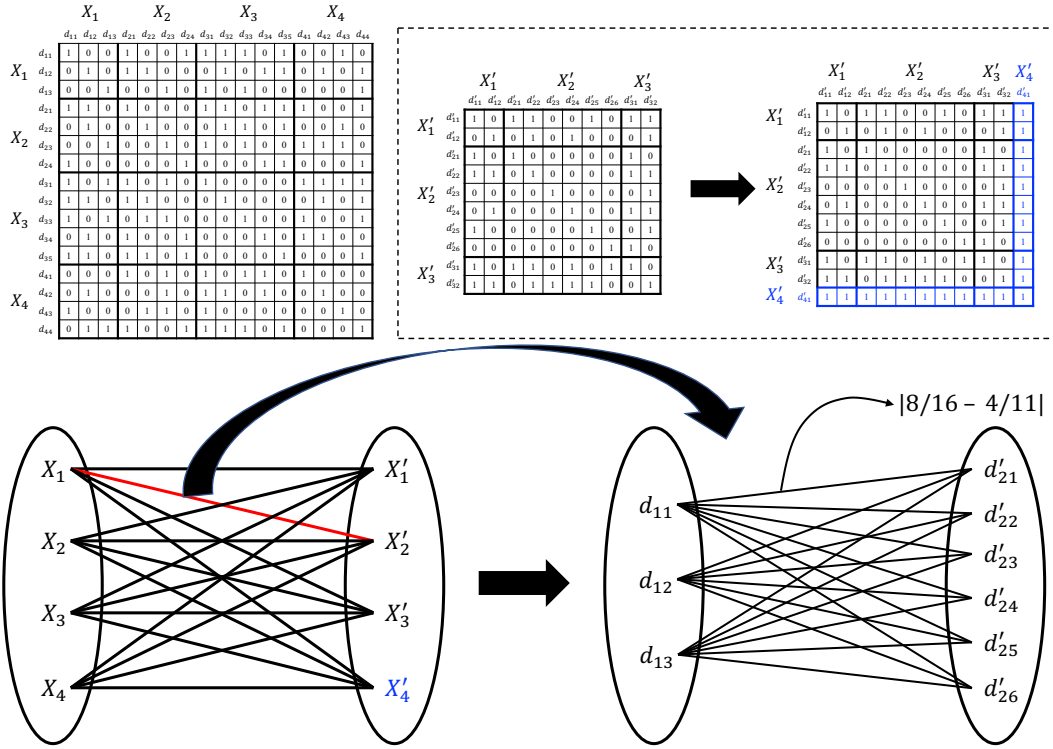
Fourth, FastMapSVM extends the applicability of SVMs and kernel methods to complex objects. Generally speaking, SVMs are particularly good for classification tasks. When combined with kernel methods, they recognize and represent complex nonlinear classification boundaries very elegantly [17]. Moreover, soft-margin SVMs with kernel methods [14] can be used to recognize both outliers and inherent nonlinearities in the data. While the SVM machinery is very effective, it requires the objects in the classification task to be represented as points in a Euclidean space. Often, it is very difficult to represent complex objects as precise geometric points without introducing inaccuracy or losing domain-specific representational features. In such cases, deep NNs have gained more popularity compared to SVMs for the reason that it is unwieldy for SVMs to represent all the features of complex objects in Euclidean space. However, FastMapSVM revives the SVM approach by leveraging a distance function and creating a low-dimensional Euclidean embedding of the complex objects.

#### 4 Distance Function on CSPs

In this section, we describe a distance function on binary CSPs. This distance function is based on maxflow computations and is illustrated in Figure 3. It is well defined for CSP instances  $\mathcal{I}_1$  and  $\mathcal{I}_2$  that may have different sizes. The maxflow computations are utilized in: (a) a single high-level “maximum matching of minimum cost” problem posed on the variables of  $\mathcal{I}_1$  and  $\mathcal{I}_2$ , and (b) multiple low-level “maximum matching of minimum cost” problems posed on the domain values of pairs of variables, one from each of  $\mathcal{I}_1$  and  $\mathcal{I}_2$ .

The high-level “maximum matching of minimum cost” problem is posed on a complete bipartite graph, in which the two partitions of the bipartite graph correspond to the variables of  $\mathcal{I}_1$  and  $\mathcal{I}_2$ , respectively. If the number of variables in  $\mathcal{I}_1$  does not match the number of variables in  $\mathcal{I}_2$ , dummy variables are added to the CSP instance with fewer variables. Figure 3 (top panel) illustrates this for  $\mathcal{I}_1$  and  $\mathcal{I}_2$  with variables  $\{X_1, X_2, X_3, X_4\}$  and  $\{X'_1, X'_2, X'_3\}$ , respectively. A dummy variable  $X'_4$  is added to  $\mathcal{I}_2$ . The dummy variable has a single domain value that is designed to be consistent with all domain values of all other variables, since this does not change the CSP instance.

The distance between  $\mathcal{I}_1$  and  $\mathcal{I}_2$  is defined to be the cost of the “maximum matching of minimum cost” on the high-level bipartite graph. This bipartite graph has an edge between every  $X_i$  in  $\mathcal{I}_1$  and every  $X'_j$  in  $\mathcal{I}_2$ . The cost annotating an edge between  $X_i$  and  $X'_j$  is itself set to be the cost of the “maximum matching of minimum cost” posed at the low level on the domain values of  $X_i$  and  $X'_j$ . Figure 3 (bottom-left panel) shows the high-level bipartite graph and highlights an edge between  $X_1$  and  $X'_2$  for explanation of the low-level “maximum matching of minimum cost”.



■ **Figure 3** The top panel shows two CSP instances with variables  $\{X_1, X_2, X_3, X_4\}$  (left) and  $\{X'_1, X'_2, X'_3\}$  (middle), respectively. A dummy variable  $X'_4$  with a singleton domain is added to the CSP instance with fewer variables (right). The bottom panel (left) shows how a “maximum matching of minimum cost” problem is posed on a complete bipartite graph with the variables of the two CSP instances in each partition. The cost annotating the edge between  $X_i$  and  $X'_j$  is itself derived from a “maximum matching of minimum cost” problem posed on the domain values of  $X_i$  and  $X'_j$ . The bottom panel (right) shows this “maximum matching of minimum cost” problem for the variables  $X_1$  and  $X'_2$ . It is posed on a complete bipartite graph with the domain values  $\{d_{11}, d_{12}, d_{13}\}$  and  $\{d'_{21}, d'_{22}, d'_{23}, d'_{24}, d'_{25}, d'_{26}\}$  in each partition. The cost annotating the edge between  $d_{11}$  and  $d'_{21}$  is the absolute value of the difference between the average compatibility of  $d_{11}$  and the average compatibility of  $d'_{21}$ .

The low-level “maximum matching of minimum cost” problem posed on the domain values of  $X_i$  and  $X'_j$  also uses a complete bipartite graph. The two partitions consist of the domain values of  $X_i$  and  $X'_j$ , respectively. The cost annotating the edge between  $d_{ip}$  and  $d'_{jq}$  is the absolute value of the difference between the average compatibility of  $d_{ip}$  and the average compatibility of  $d'_{jq}$ . Figure 3 (bottom-right panel) shows the low-level “maximum matching of minimum cost” problem posed on the domain values of  $X_1$  and  $X'_2$ . The domains of  $X_1$  and  $X'_2$  are  $\{d_{11}, d_{12}, d_{13}\}$  and  $\{d'_{21}, d'_{22}, d'_{23}, d'_{24}, d'_{25}, d'_{26}\}$ , respectively. Consider the edge between  $d_{11}$  and  $d'_{21}$ . The average compatibility of  $d_{11}$  is the fraction of “1”s in the column “ $d_{11}$ ” in the matrix representation of  $\mathcal{I}_1$ . This fraction is equal to  $8/16$ . The average compatibility of  $d'_{21}$  is the fraction of “1”s in the column “ $d'_{21}$ ” in the matrix representation of  $\mathcal{I}_2$  after adding the dummy variable  $X'_4$ . This fraction is equal to  $4/11$ . Therefore, the cost annotating the edge between  $d_{11}$  and  $d'_{21}$  is equal to  $|8/16 - 4/11|$ .

The “maximum matching of minimum cost” problems in the high level and the low level are posed on bipartite graphs. Since the costs annotating the edges of the bipartite graphs in the high level and the low level are non-negative, the distance function is also non-negative.

Moreover, since the two partitions of any bipartite graph can be viewed interchangeably without affecting the “maximum matching of minimum cost”, the overall distance function is symmetric. It is also easy to observe that the distance between two identical CSP instances is always 0. These properties of the distance function satisfy all the requirements imposed on it by the FastMap component of FastMapSVM.

The high-level bipartite graph is invariant to the orderings on the elements within each partition. That is, it is invariant to the orderings on the variables of  $\mathcal{I}_1$  and  $\mathcal{I}_2$ . Similarly, all low-level bipartite graphs are invariant to the orderings on the domain values of the participating variables. Therefore, the overall distance function is invariant to variable-orderings as well as domain value-orderings. This allows us to bypass data augmentation methods typically required for training other ML models.<sup>5</sup> In the context of CSPs, a CSP instance is typically augmented by changing the ordering on its variables or the ordering on the domain values of individual variables. However, doing so generates an exponential number of CSP training instances within the same equivalence class. This drawback of traditional ML algorithms of having to learn equivalence classes is now intelligently addressed within the framework of FastMapSVM by utilizing a distance function that is invariant to both variable-orderings and domain value-orderings.

We note that the above distance function could have been defined in many other ways. For example, we could have introduced dummy domain values in the low-level “maximum matching of minimum cost” problems to equalize the domain sizes of the participating variables. We could have also chosen not to use dummy variables in the high-level “maximum matching of minimum cost” problem. In addition, we could have defined the costs annotating the edges of the bipartite graphs using many other characteristics of the CSPs. These variations of the distance function are not of fundamental importance to this paper. Instead, in this paper, we focus on the advantages of the FastMapSVM framework as a whole. The study of more refined distance functions is delegated to future work.

## 5 Experimental Results

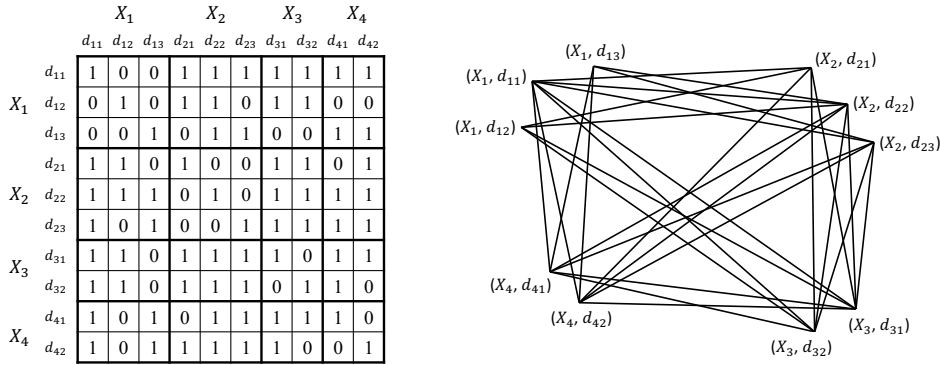
In this section, we describe the comparative performance of FastMapSVM against other state-of-the-art ML approaches on predicting CSP satisfiability.

### 5.1 Experimental Setup

We evaluate FastMapSVM against three competing approaches. The first is a state-of-the-art deep graph convolutional neural network (DGCNN) [23]. The second is a state-of-the-art graph isomorphism network (GIN) [21]. Both these networks ingest a CSP instance in the form of a graph, as shown in Figure 4. In the graphical representation of a binary CSP instance, a vertex represents a domain value and is tagged with the name of the variable that it belongs to. Information in these tags is utilized by the DGCNN and the GIN. An edge between two vertices  $v_1$  and  $v_2$  with tags  $X_i$  and  $X_j$ , respectively, represents the compatible combination  $(X_i \leftarrow v_1, X_j \leftarrow v_2)$  allowed by the direct constraint between  $X_i$  and  $X_j$ .<sup>6</sup> DGCNN and GIN do not require the CSP training and test instances to be of the same size.

The third is a polynomial-time algorithm based on establishing arc-consistency. This algorithm first establishes arc-consistency and then checks whether any variable’s domain is annihilated. If so, it declares the CSP instance to be “unsatisfiable”. Otherwise, it declares

<sup>5</sup> Data augmentation refers to transformations of data without changing their labels, known as label-preserving transformations. For example, to generate more training data serving object recognition tasks in computer vision applications, an image can be augmented by translating it or reflecting it



■ **Figure 4** The left panel shows the matrix representation of a binary CSP instance. The right panel shows its graphical representation. The vertices represent domain values and are clustered into four groups, corresponding to the four variables  $\{X_1, X_2, X_3, X_4\}$ .

the CSP instance to be “satisfiable”. This algorithm is used in our evaluation to demonstrate that FastMapSVM’s capabilities go beyond that of a polynomial-time algorithm.<sup>7</sup> Of course, a polynomial-time algorithm based on establishing path-consistency could also have been used. But we excluded this algorithm since arc-consistency already provides the required proof of concept and establishing path-consistency is prohibitively expensive.

In our experiments, we do not include results from the CSP-cNN framework of [20] for the following reasons. First, this framework has the drawback that it is applicable only to CSP training and test instances of the same size. (FastMapSVM does not have this drawback since the distance function does not require the CSP instances to be of the same size.) Second, the success of CSP-cNN critically depends on data augmentation methods: For each CSP training instance, a very large number of other training instances that permute the variables and their domain values also have to be generated. (This requirement is completely obviated by FastMapSVM since the distance function is invariant to such permutations.) Third, CSP-cNN has been shown to be successful only on Boolean binary CSP instances, that is, the polynomial-time solvable 2-SAT problems. (FastMapSVM does not have this limitation; below, we demonstrate its success on general binary CSP instances.)

We implemented FastMapSVM and arc-consistency in Python3 and ran them on a laptop with an Apple M2 chip with 16 GB memory. We ran DGCNN and GIN on a Linux system with an Intel(R) Xeon(R) Silver 4216 CPU at 2.10 GHz. The different platforms are inconsequential to the comparative performances of these algorithms with respect to effectiveness. For each dataset, we trained DGCNN and GIN for 100 epochs with a learning rate of 0.0001 and a minibatch size of 100 to obtain representative results.

## 5.2 Instance Generation

We generate the binary CSP instances for both training and testing using the Model A method in [18, 20]. We generate a CSP instance by first picking the number of variables  $N$  uniformly at random to be an integer within the range  $[1, 100]$ . Then, we pick the domain

horizontally without changing its label [10].

<sup>6</sup> The graphical representation of a binary CSP instance is obtained by parsing its matrix representation. Thus, we correctly represent the compatible tuples of domain values between every pair of variables, even if there does not exist a direct constraint between those variables.

<sup>7</sup> This is done to avoid the pitfalls of [20], as mentioned before.

size of each variable independently and uniformly at random to be an integer within the range  $[1, 10]$ . We use a probability parameter  $P_1$  to independently determine the existence of a direct constraint between each pair of distinct variables. That is, for each pair of distinct variables  $X_i$  and  $X_j$ , we introduce a direct constraint between them with probability  $P_1$ . Moreover, we use a probability parameter  $P_2$  to determine the compatible tuples of a direct constraint. For a pair of variables  $X_i$  and  $X_j$  with a direct constraint between them, each tuple  $(X_i \leftarrow d_{ip}, X_j \leftarrow d_{jq})$  is independently deemed to be compatible with probability  $1 - P_2$ . We set  $P_1 = 1$  and  $P_2 = 0.4$  to obtain representative results for all approaches.

Model A has a tendency to produce mostly unsatisfiable CSP instances with increasing  $N$  [18, 20]. Therefore, we use a “hidden solution” method to generate satisfiable CSP instances whenever required. In this method, a set of hidden solutions of the CSP instance is chosen a priori.<sup>8</sup> A hidden solution  $(X_1 \leftarrow d_{1p_1}, X_2 \leftarrow d_{2p_2} \dots X_N \leftarrow d_{Np_N})$  is utilized as follows: While generating the direct constraints using Model A, a direct constraint between variables  $X_i$  and  $X_j$  reserves the tuple  $(X_i \leftarrow d_{ip_i}, X_j \leftarrow d_{jp_j})$  as being compatible before the other tuples are set using the probability parameter  $P_2$ . Therefore,  $(X_1 \leftarrow d_{1p_1}, X_2 \leftarrow d_{2p_2} \dots X_N \leftarrow d_{Np_N})$  satisfies all the direct constraints and, consequently, qualifies as a solution. Similarly, multiple hidden solutions can be utilized with the following modification in the generation procedure: A direct constraint between variables  $X_i$  and  $X_j$  reserves multiple tuples as being compatible. For generating satisfiable CSP instances, we pick the number of hidden solutions uniformly at random to be an integer within the range  $[1, 10]$ . We pick a hidden solution itself by assigning a domain value chosen independently and uniformly at random for each variable from its domain.

We generate three datasets: Dataset-1, Dataset-2, and Dataset-3. For each dataset, we generate 1000 training instances and 1000 test instances. Each training and test set has an equal number of satisfiable and unsatisfiable instances.

In Dataset-1, we generate the instances using Model A. Since Model A frequently generates unsatisfiable instances, we use a complete CSP solver to identify and collect such instances. We generate the satisfiable instances using the hidden solution method, as described above.

In Dataset-2, we generate the satisfiable instances as in Dataset-1. However, we design and generate the unsatisfiable instances to be more challenging. We do this by hiding two complementary pseudo-solutions  $(X_1 \leftarrow d_{1p_1}, X_2 \leftarrow d_{2p_2} \dots X_N \leftarrow d_{Np_N})$  and  $(X_1 \leftarrow d_{1q_1}, X_2 \leftarrow d_{2q_2} \dots X_N \leftarrow d_{Nq_N})$ . We identify a pair of distinct variables  $X_i$  and  $X_j$  such that  $d_{ip_i} \neq d_{iq_i}$  and  $d_{jp_j} \neq d_{jq_j}$ . All direct constraints between distinct variables  $X_s$  and  $X_t$  such that  $\{X_s, X_t\} \neq \{X_i, X_j\}$  are generated as before by reserving the tuples  $(X_s \leftarrow d_{sp_s}, X_t \leftarrow d_{tp_t})$  and  $(X_s \leftarrow d_{sq_s}, X_t \leftarrow d_{tq_t})$  as being compatible. However, the direct constraint between  $X_i$  and  $X_j$  reserves the tuples  $(X_i \leftarrow d_{ip_i}, X_j \leftarrow d_{jq_j})$  and  $(X_i \leftarrow d_{iq_i}, X_j \leftarrow d_{jp_j})$  as being compatible and reserves the tuples  $(X_i \leftarrow d_{ip_i}, X_j \leftarrow d_{jp_j})$  and  $(X_i \leftarrow d_{iq_i}, X_j \leftarrow d_{jq_j})$  as being not compatible. We finally use a complete CSP solver to verify that the CSP instance is indeed unsatisfiable.<sup>9</sup>

In Dataset-3, we generate the satisfiable instances as in Dataset-1. However, we design and generate the unsatisfiable instances differently from in Dataset-2. We do this by first hiding two complementary pseudo-solutions  $(X_1 \leftarrow d_{1p_1}, X_2 \leftarrow d_{2p_2} \dots X_N \leftarrow d_{Np_N})$  and  $(X_1 \leftarrow d_{1q_1}, X_2 \leftarrow d_{2q_2} \dots X_N \leftarrow d_{Nq_N})$ , as in Dataset-2. However, we gather all variables  $X_{r_1}, X_{r_2} \dots X_{r_M}$  for which the two pseudo-solutions have different assignments

<sup>8</sup> The CSP instance can have other solutions as well.

<sup>9</sup> This procedure frequently generates unsatisfiable instances, as required. However, satisfiable instances that are generated occasionally are filtered out by the CSP solver.

of domain values, that is,  $d_{r_m p_{r_m}} \neq d_{r_m q_{r_m}}$ , for all  $1 \leq m \leq \bar{M}$ . For any two distinct variables  $X_i$  and  $X_j$  in  $\{X_{r_1}, X_{r_2} \dots X_{r_{\bar{M}}}\}$ , we reserve the tuples  $(X_i \leftarrow d_{ip_i}, X_j \leftarrow d_{jq_j})$  and  $(X_i \leftarrow d_{iq_i}, X_j \leftarrow d_{jp_j})$  as being not compatible. Finally, we pick two distinct variables  $X_s$  and  $X_t$  from  $\{X_{r_1}, X_{r_2} \dots X_{r_{\bar{M}}}\}$  and overwrite the tuples  $(X_s \leftarrow d_{sp_s}, X_t \leftarrow d_{tq_t})$  and  $(X_s \leftarrow d_{sq_s}, X_t \leftarrow d_{tp_t})$  as being compatible and reserve the tuples  $(X_s \leftarrow d_{sp_s}, X_t \leftarrow d_{tp_t})$  and  $(X_s \leftarrow d_{sq_s}, X_t \leftarrow d_{tq_t})$  as being not compatible. As before, we use a complete CSP solver to verify that the CSP instance is indeed unsatisfiable.

### 5.3 Results

We show three sets of results pertaining to FastMapSVM. First, we show the 2-dimensional and the 3-dimensional embeddings that FastMapSVM produces to aid visualization. Second, we show the behavior of FastMapSVM with respect to the hyperparameter  $K$ , that is, the number of dimensions and with respect to the size of the training data. Third, we show the comparative performance of FastMapSVM against DGCNN, GIN, and arc-consistency.

FastMapSVM used the SVM classifier from the scikit-learn library. Its hyperparameter settings were determined by grid search. For Dataset-1 and Dataset-3, the hyperparameters were regularization parameter = 8, kernel = “rbf”, and kernel coefficient = “scale”. For Dataset-2, the hyperparameters were regularization parameter = 8, kernel = “poly”, and kernel coefficient = “scale”.

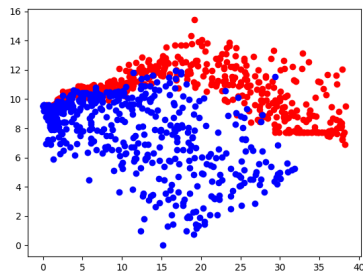
Figure 5 shows a perspicuous visualization of the CSP test instances for all three datasets. This visualization capability is unique to FastMapSVM. We note that while the accuracy, recall, precision, and the F1 score of FastMapSVM typically increase with increasing  $K$ ,  $K = 2$  and  $K = 3$  are the only two values that support visualization. Still, in most cases, Figure 5 shows a clear separation between the satisfiable and unsatisfiable instances. Moreover, the separation is clearer in the 3-dimensional embeddings compared to their 2-dimensional counterparts.

Figure 6a shows the behavior of FastMapSVM with respect to the number of dimensions  $K$  on Dataset-1. Its behavior on the other datasets is similar. The performance metrics, that is, the accuracy, recall, precision, and the F1 score, improve with increasing  $K$ . This is intuitively expected since the distances between the CSP instances can be embedded with lower distortion in higher dimensions. However, Figure 6a also shows that a point of diminishing returns is attained rather quickly at around  $K = 8$ . This shows that  $K = 8, 9$ , or  $10$  is good enough for the CSP domain. Finally, Figure 6a also shows that the improvements in the performance metrics are significant between  $K = 2$  and  $K = 8$ .

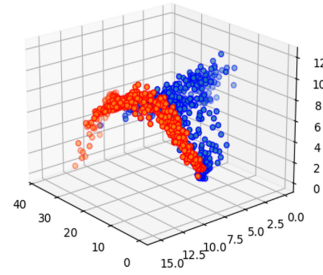
Figure 6b shows the behavior of FastMapSVM with respect to the size of the training data on Dataset-1. Its behavior on the other datasets is similar. The performance metrics improve with increasing size of the training data. Figure 6b also shows that the improvements in the performance metrics are significant between 128 and 256 training data instances. Further improvements are gradual between 256 and 1000 training data instances. This shows that FastMapSVM has the capability to achieve good performance from relatively small amounts of training data and training time.

Table 1 shows a comparison of all the competing methods on all three datasets with respect to all of the performance metrics. It uses  $K = 8$  for FastMapSVM. It also shows two versions of DGCNN and GIN: the “labeled” version and the “unlabeled” version.

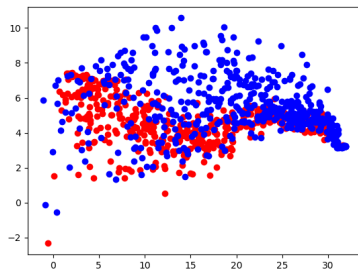
We recollect that in the graphical representation of a binary CSP instance, a vertex represents a domain value and is tagged with the name of the variable that it belongs to. Information in these tags is available to be utilized by the DGCNN and the GIN. The labeled versions of DGCNN and GIN utilize this information while the unlabeled versions ignore this



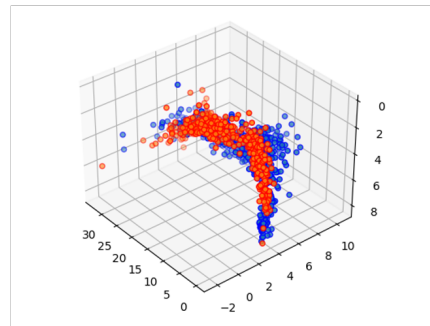
(a) Dataset-1 CSP instances embedded in a 2-dimensional Euclidean space by FastMapSVM



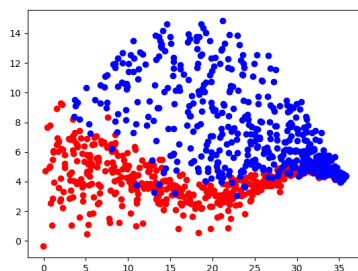
(b) Dataset-1 CSP instances embedded in a 3-dimensional Euclidean space by FastMapSVM



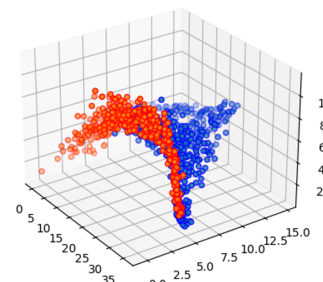
(c) Dataset-2 CSP instances embedded in a 2-dimensional Euclidean space by FastMapSVM



(d) Dataset-2 CSP instances embedded in a 3-dimensional Euclidean space by FastMapSVM



(e) Dataset-3 CSP instances embedded in a 2-dimensional Euclidean space by FastMapSVM



(f) Dataset-3 CSP instances embedded in a 3-dimensional Euclidean space by FastMapSVM

■ **Figure 5** The figure shows the low-dimensional Euclidean embeddings produced by FastMapSVM for classifying CSP instances. Mostly, there is a clear separation of satisfiable instances (blue) and unsatisfiable instances (red).

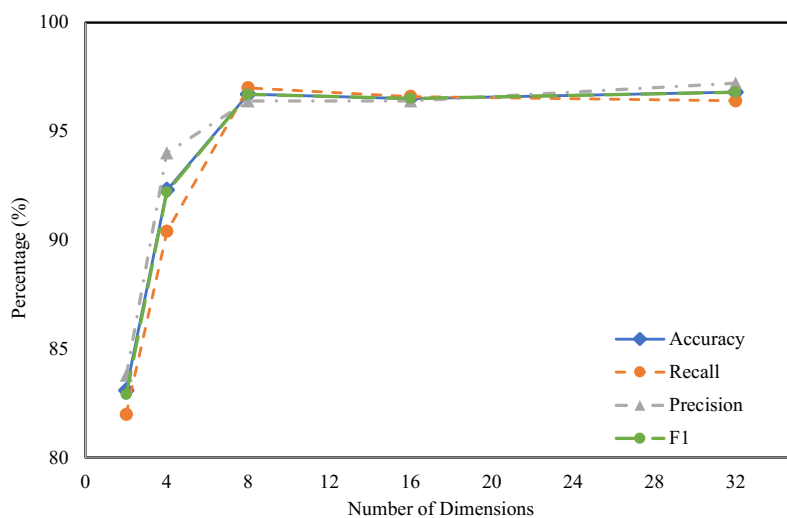
information. Table 1 shows that the unlabeled versions perform better than their labeled counterparts. While this is a little surprising, it is likely that the unlabeled versions indirectly perform permutation reasoning on the tags (names of variables) much more efficiently.

Table 1 shows that FastMapSVM generally outperforms all other competing methods by a significant margin. Even on a particular dataset where it is not the top performer with respect to a particular performance metric, it is a close second. Overall, Dataset-1 seems to be the easiest for all methods and Dataset-2 seems to be the hardest for all methods.

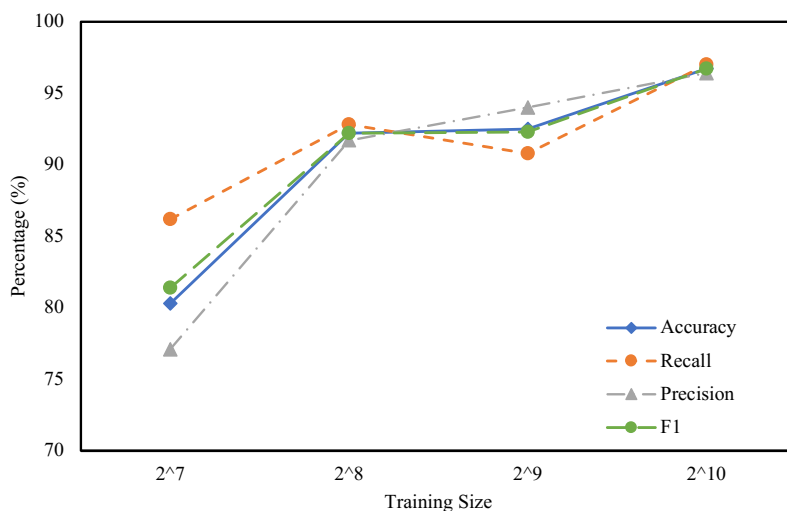
In comparison to arc-consistency, FastMapSVM is significantly better on Dataset-2 and Dataset-3. On these datasets, arc-consistency declares all test instances as being “satisfiable”, leading to a perfect recall score but very poor precision, accuracy, and F1 scores. On the one



## 40:14 FastMapSVM for Predicting CSP Satisfiability



(a) Influence of the number of dimensions on the performance of FastMapSVM



(b) Influence of the size of the training data on the performance of FastMapSVM

■ **Figure 6** The figure shows the behavior of FastMapSVM with respect to the number of dimensions and with respect to the size of the training data. The performance metrics include the accuracy, recall, precision, and the F1 score.

hand, this shows that arc-consistency is ineffective in recognizing unsatisfiable CSP instances. On the other hand, it also shows that CSP instances generated as in Dataset-1 are insufficient to conclusively evaluate competing ML methods. In contrast, FastMapSVM performs well on all three datasets.

In comparison to DGCNN and GIN, FastMapSVM is significantly better on all three datasets. On the accuracy, recall, and F1 scores, FastMapSVM is better than DGCNN, which in turn is better than GIN. GIN generally has high precision scores but very poor recall scores. This shows that it is poor in identifying satisfiable instances but is mostly correct

■ **Table 1** The table shows all performance metrics for all competing methods on all datasets. “AC” represents arc-consistency.

Dataset	Model	Accuracy	Recall	Precision	F1
Dataset-1	FastMapSVM	96.7%	97.0%	96.4%	96.7%
	AC	<b>99.3%</b>	<b>100.0%</b>	96.7%	<b>98.3%</b>
	DGCNN (unlabeled)	94.2%	92.2%	96.0%	94.1%
	DGCNN (labeled)	82.3%	82.0%	82.5%	82.2%
	GIN (unlabeled)	84.1%	67.0%	<b>99.4%</b>	80.0%
	GIN (labeled)	56.4%	52.6%	56.9%	54.7%
Dataset-2	FastMapSVM	<b>82.9%</b>	72.8%	<b>91.2%</b>	<b>81.0%</b>
	AC	50.1%	<b>100.0%</b>	50.1%	66.8%
	DGCNN (unlabeled)	73.4%	61.4%	80.8%	69.8%
	DGCNN (labeled)	53.9%	51.2%	54.1%	52.6%
	GIN (unlabeled)	71.9%	49.0%	90.4%	63.6%
	GIN (labeled)	54.4%	51.6%	54.7%	53.1%
Dataset-3	FastMapSVM	<b>95.4%</b>	94.4%	96.3%	<b>95.3%</b>
	AC	50.0%	<b>100.0%</b>	50.0%	66.7%
	DGCNN (unlabeled)	90.3%	86.6%	93.5%	89.9%
	DGCNN (labeled)	74.7%	71.8%	76.2%	73.9%
	GIN (unlabeled)	78.4%	53.6%	<b>98.5%</b>	69.4%
	GIN (labeled)	57.4%	53.8%	58.0%	55.8%

when it does so. FastMapSVM does not have this drawback. Moreover, on the accuracy and F1 scores, FastMapSVM outperforms the closest competitor (DGCNN) by larger margins with increasing hardness of the CSP instances, that is, in the order of Dataset-1, Dataset-3, and Dataset-2. Even on the metric of efficiency, FastMapSVM outperforms DGCNN and GIN.<sup>10</sup>

## 6 Conclusions and Future Work

In this paper, we introduced a novel ML framework, called FastMapSVM, for the task of predicting CSP satisfiability. FastMapSVM overcomes the hurdles faced by other ML approaches in the CSP domain. It leverages a distance function on CSPs that is defined via maxflow computations. FastMapSVM is applicable to CSP training and test instances of different sizes and is invariant to both variable-orderings and domain value-orderings. This allows it to bypass the onus of having to learn equivalence classes of CSP instances and, therefore, requires significantly smaller amounts of time and data for model training compared to other ML algorithms. FastMapSVM also uses the intelligence of SVMs, kernel methods, and maxflow computations, accounting for its superior empirical performance, even over state-of-the-art graph neural networks. Moreover, it facilitates a perspicuous visualization of the CSP instances, their distribution, and the classification boundaries between them. Overall, the FastMapSVM framework for CSPs has broader applicability and various representational and combinatorial advantages compared to other ML approaches.

There are many avenues for future work. These include the design of better distance functions on CSPs, the application of FastMapSVM to optimization variants of CSPs, and the general facilitation of integrating constraint reasoning and ML methods via FastMapSVM.

<sup>10</sup> DGCNN and GIN ran on a different platform compared to FastMapSVM. However, the ballpark results are still conclusive.

---

**References**

---

- 1 Abder-Rahman Ali, Micael S Couceiro, Aboul Ella Hassanien, and D Jude Hemanth. Fuzzy c-means based on Minkowski distance for liver CT image segmentation. *Intelligent Decision Technologies*, 2016.
- 2 Alejandro Arbelaez, Youssef Hamadi, and Michele Sebag. Continuous search in constraint programming. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence*, 2010.
- 3 Liron Cohen, Tansel Uras, Shiva Jahangiri, Aliyah Arunasalam, Sven Koenig, and T. K. Satish Kumar. The FastMap algorithm for shortest path computations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018.
- 4 Rina Dechter. *Constraint processing*. Morgan Kaufmann, 2003.
- 5 Christos Faloutsos and King-Ip Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1995.
- 6 Ian Philip Gent, Christopher Anthony Jefferson, Lars Kotthoff, Ian James Miguel, Neil Charles Armour Moore, Peter Nightingale, and Karen Petrie. Learning when to use lazy learning in constraint solving. In *Proceedings of the 19th European Conference on Artificial Intelligence*, 2010.
- 7 Alessio Guerri and Michela Milano. Learning techniques for automatic algorithm portfolio selection. In *Proceedings of the 16th European Conference on Artificial Intelligence*, 2004.
- 8 Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC—instance-specific algorithm configuration. In *Proceedings of the 19th European Conference on Artificial Intelligence*, 2010.
- 9 Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, 2016.
- 10 Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 2017.
- 11 Ang Li, Peter Stuckey, Sven Koenig, and T. K. Satish Kumar. A FastMap-based algorithm for block modeling. In *Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2022.
- 12 Jiaoyang Li, Ariel Felner, Sven Koenig, and T. K. Satish Kumar. Using FastMap to solve graph problems in a Euclidean space. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2019.
- 13 Eoin O’Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O’Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of the Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.
- 14 Arti Patle and Deepak Singh Chouhan. SVM kernel functions for classification. In *Proceedings of the International Conference on Advances in Technology and Engineering*, 2013.
- 15 Luca Pulina and Armando Tacchella. A multi-engine solver for quantified boolean formulas. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, 2007.
- 16 Faisal Rahutomo, Teruaki Kitasuka, and Masayoshi Aritsugi. Semantic cosine similarity. In *Proceedings of the International Student Conference on Advanced Science and Technology*, 2012.
- 17 V David Sánchez A. Advanced support vector machines and kernel methods. *Neurocomputing*, 2003.
- 18 Barbara M Smith and Martin E Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 1996.
- 19 Malcolm White, Kushal Sharma, Ang Li, T. K. Satish Kumar, and Nori Nakata. FastMapSVM: Classifying complex objects using the FastMap algorithm and support-vector machines. *arXiv preprint arXiv:2204.05112*, 2022.

- 20 Hong Xu, Sven Koenig, and T. K. Satish Kumar. Towards effective deep learning for constraint satisfaction problems. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming*, 2018.
- 21 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.
- 22 Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 2008.
- 23 Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.



# Improving Local Search for Pseudo Boolean Optimization by Fragile Scoring Function and Deep Optimization

Wenbo Zhou ✉ 


School of Information Science and Technology, Northeast Normal University, Changchun, China  
Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China  
Key Laboratory of Symbolic Computation and Knowledge Engineering of MOE, Jilin University, Changchun, China

Yujiao Zhao ✉ 

School of Information Science and Technology, Northeast Normal University, Changchun, China

Yiyuan Wang<sup>1</sup> ✉ 

School of Information Science and Technology, Northeast Normal University, Changchun, China  
Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China

Shaowei Cai ✉ 

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China


School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China

Shimao Wang ✉

School of Information Science and Technology, Northeast Normal University, Changchun, China

Xinyu Wang ✉

School of Information Science and Technology, Northeast Normal University, Changchun, China

Minghao Yin<sup>1</sup> ✉ 

School of Information Science and Technology, Northeast Normal University, Changchun, China  
Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China

---

## Abstract

Pseudo-Boolean optimization (PBO) is usually used to model combinatorial optimization problems, especially for some real-world applications. Despite its significant importance in both theory and applications, there are few works on using local search to solve PBO. This paper develops a novel local search framework for PBO, which has three main ideas. First, we design a two-level selection strategy to evaluate all candidate variables. Second, we propose a novel deep optimization strategy to disturb some search spaces. Third, a sampling flipping method is applied to help the algorithm jump out of local optimum. Experimental results show that the proposed algorithms outperform three state-of-the-art PBO algorithms on most instances.

**2012 ACM Subject Classification** Computing methodologies → Search methodologies

**Keywords and phrases** Local Search, Pseudo-Boolean Optimization, Deep Optimization

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.41

**Supplementary Material** *Software (Source Code)*: <https://github.com/yiyuanwang1988/DeepOpt-PBO>, archived at `swh:1:dir:a1d58cf93325bed3675f5872f42adf459a518a92`

---

<sup>1</sup> corresponding author



**Funding** This work is supported by the National Natural Science Foundation of China under Grant No.61806050, the Natural Science Research Foundation of Jilin Province of China under Grant Nos.YDZJ202201ZYTS412 and YDZJ202201ZYTS423, Ministry of Education under Grant No.2021BCF01002, CCF-Huawei Populus Grove Fund, and the Fundamental Research Funds for the Central Universities under Grant Nos.2412023YQ003, 2412022ZD015, 2412022ZD018, 2412022QD040 and 93K172022K10.

## 1 Introduction

The Boolean Satisfiability (SAT) problem is a prototypical NP-complete problem, whose aim is to determine whether a given propositional formula is satisfiable or not. The SAT problem plays a core role in many domains of computer science and artificial intelligence [14]. Many real-world problems can be encoded into SAT and its optimization version MaxSAT and solved using their powerful solvers. However, due to the limited expressive power of SAT and MaxSAT, their encodings often generate very large problem instances. For such cases, pseudo-Boolean optimization (PBO) provides a more expressive and natural way to express constraints than SAT and MaxSAT [23]. Besides, PBO is very close enough to SAT to benefit from the recent advances in SAT solving [23].

The PBO consists of a set of pseudo-Boolean constraints and an objective function, whose goal is to find a solution that minimizes the objective function and satisfies all pseudo-Boolean constraints. Solvers for PBO can be divided into complete and incomplete solvers. Up to now, many complete PBO solvers have been proposed, such as Sat4j [3], Open-WBO [21], NaPS [24], RoundingSat [12, 10], RSCard [13], RS/lp [9] and PBO-IHS [26, 27].

Compared to complete PBO solvers, there are relatively fewer incomplete solvers on PBO. The reason may be that PBO is more complicated than SAT and how to find a suitable variable to flip is still difficult. As one of the most popular incomplete approaches, local search can find an approximate solution within a reasonable time [32, 33]. Lei et al. [18] proposed a novel local search algorithm called LS-PBO to handle PBO. Key features of LS-PBO include a converting method to obtain corresponding objective constraints, a weighting scheme to guide the search direction, and a well-designed scoring function to flip some candidate variables. Besides, for some special cases of PBO such as NK-Landscapes and MAX-kSAT, a new perturbation strategy called VIGbP was proposed [29]. According to the literature, the current best incomplete algorithm for PBO is LS-PBO.

In this work, to further improve the performance of local search algorithms on solving PBO, we propose a local search framework for PBO based on three main ideas.

First, we present a two-level selection strategy to choose which variable to flip. In our proposed algorithm, we use the previous scoring function *score* [18] as a primary scoring function, which is defined as the decrease of the total penalty of related constraints and objective function. To address the issue about tie-breaking in the primary scoring function, we propose a fragile scoring function *hhscore* as the secondary scoring function. The proposed *hhscore* can greatly differentiate between two kinds of satisfied constraints by using the definition of satisfied threshold.

Second, we propose a novel deep optimization strategy (DeepOpt) to deeply probe some regions using locked and unlocked operations during the local search. Recently, the DeepOpt strategy was first proposed by Chen et al. [8] and has been successfully applied in solving dominating set problems. In our proposed DeepOpt strategy, we preferentially select some variables in unsatisfied constraints as unlocked operations. Moreover, we use some trigger conditions to decide whether the algorithm calls DeepOpt or not.

Third, we design a sampling flipping method to modify a current candidate solution when the algorithm tramps into local optimum. In the proposed method, we adopt two sampling ways to collect some samples among all unsatisfied constraints. We further employ a probabilistic heuristic “best from multiple selections” (BMS) [6] to select a candidate variable. Besides, we also apply a new scoring function to simultaneously flip two variables.

By incorporating these three ideas and some other tricks, we develop two local search algorithms for PBO. Extensive experiments are carried out to evaluate our algorithms on the benchmarks used in the literature. Experimental results show that the proposed algorithms outperform three state-of-the-art PBO algorithms on almost all the benchmarks.

## 2 Preliminaries

Since a non-linear pseudo-Boolean (PB) constraint can be translated into an equivalent set of linear PB constraints [23], we only start with a review of the basics of linear PB constraints here. A linear PB constraint is defined over a finite set of Boolean variables. Boolean variable  $x_i$  can take only two values *false* (0) and *true* (1). A literal  $l_i$  over a Boolean variable  $x_i$  is either  $x_i$  or  $\bar{x}_i = 1 - x_i$ . A linear PB constraint is a 0-1 integer inequality.

$$\sum_i a_i l_i \triangleright b \quad (1)$$

where  $a_i$  and  $b$  are integer constants,  $l_i$  are literals and  $\triangleright \in \{=, >, \geq, <, \leq\}$  is one of the classical relational operators. All PB constraints can be normalized into the following form.

$$\sum_i a_i l_i \geq b \quad (2)$$

where all the literals  $l_i$  are distinct, and all the coefficients  $a_i$  and the *degree*  $b$  are non-negative integers.

A PB formula is a conjunction of PB constraints, denoted as  $F = C_1 \wedge C_2 \wedge \dots \wedge C_m$ , where  $C_p$  ( $p \in \mathbb{Z}, 1 \leq p \leq m$ ) is a PB constraint. The PBO problem consists of a PB formula  $F$  and an objective function  $O : \sum_i c_i l_i$  where  $c_i$  is a non-negative integer coefficient. Given a PB constraint  $C_p : \sum_i a_i^p l_i^p \geq b^p$ , the sum of its coefficients is defined as  $sum(C_p) = \sum_i a_i^p$ , its average coefficient is defined as  $coeff(C_p) = sum(C_p)/|L(C_p)|$  where  $L(C_p)$  is the set of literals in  $C_p$ , and its maximum coefficient  $a_{max}^p$  is the maximum value of coefficients in  $C_p$ . The average coefficient of an objective function  $O$  is defined as  $coeff(O) = \sum_i c_i/|L(O)|$  where  $L(O)$  is the set of literals in  $O$ .

Given a PB formula  $F$ , its complete assignment is a mapping that assigns 0 or 1 to each variable. Given a complete assignment of  $F$ , if a literal evaluates to true, we say it is a *true literal* and otherwise it is a *false literal*. A PB constraint  $C_p$  is *satisfied* when the left and right terms of the constraint evaluate to integers which satisfy the relational operator. Otherwise,  $C_p$  is *unsatisfied*. The sum of coefficients of true literals in  $C_p$  is denoted as  $SatL(C_p) = \sum_{l_i^p=1 \wedge l_i^p \in C_p} a_i^p$ . An assignment  $\alpha$  of  $F$  is feasible if and only if  $\alpha$  satisfies all PB constraints in  $F$ . The value of the objective function of a feasible solution  $\alpha$  is denoted as  $obj(\alpha)$ . The PBO problem aims to obtain a feasible solution for  $F$  with the minimum objective value.

### 2.1 Review for Weighting and Scoring Function

Constraint weighting techniques have usually been used to guide and diversify the search process [31, 5, 15]. The weighting based scoring function *score* for PBO is recently proposed by Lei et al. [18]. Each PB constraint  $C_p \in F$  and an objective function  $O$  have the property



of weighting, denoted as  $w(C_p)$  and  $w(O)$ , respectively. Before introducing the weighting, we first present a basic concept. Given a PB formula  $F$  and  $m$  is the number of constraints, the average constraint coefficient of  $F$  is defined as  $avg\_coeff = \sum_{p=1}^m w(C_p) \times coeff(C_p)/m$ .

The property of weighting works as follows.

**Weighting Rule 1:** At first,  $w(O) = 1$  and  $w(C_p) = 1$  for an objective function  $O$  and each PB constraint  $C_p$ .

**Weighting Rule 2:** For each unsatisfied constraint  $C_p$ ,  $w(C_p) = w(C_p) + 1$ .

**Weighting Rule 3:** If the current  $obj(\alpha)$  of the objective function is better than the current best-found value of objective function during the search process so far and  $w(O) \times coeff(O) - avg\_coeff \leq \zeta$ , then  $w(O) = w(O) + 1$ . In our work, we use the same parameter value of  $\zeta$  as [18], i.e.,  $\zeta = 100$ .

Given an assignment  $\alpha$  of  $F$ , if a PB constraint  $C_p : \sum_i a_i^p l_i^p \geq b^p$  is unsatisfied, the *penalty* of  $C_p$  is defined as  $w(C_p) \times (b^p - \sum_i a_i^p l_i^p)$ . For the objective function  $O : \sum_i c_i l_i$ , the *penalty* of  $O$  is defined as  $w(O) \times \sum_i c_i l_i$ . Based on the definition of *penalty*, we introduce two scoring functions including hard score *hscore* and objective score *oscore* as below. For a Boolean variable  $x_i$ , the respective  $hscore(x_i)$  and  $oscore(x_i)$  are the decrease of the total penalty of unsatisfied PB constraints and the objective function caused by flipping  $x_i$ . Combing the above scoring functions, we define the score of a variable  $x_i$  as  $score(x_i) = hscore(x_i) + oscore(x_i)$ . Remark that after flipping some variables during the search process, the corresponding *score* values should be updated accordingly.

### 3 Two-Level Selection Strategy

In this section, we introduce a secondary scoring function to reinforce local search algorithms for PBO and then propose a two-level selection strategy to decide a candidate variable.

#### 3.1 Fragile Scoring Function

As a core guidance for the search process, scoring functions play an important role in local search algorithms, which measure the benefits of a candidate variable. In local search algorithms for PBO, such benefits *hscore* can be set as the distance between the sum of coefficients of true literals and the corresponding degree, whereas *oscore* can be set as the distance between the current and best-found objective values [18].

In our algorithm, we consider the sum of *hscore* and *oscore* (i.e., *score*) as the primary scoring function, which is the same method as LS-PBO [18]. According to our preliminary experiments, 9% candidate variables on average have the same largest *score* value during the search. To address the issue about tie-breaking in the primary scoring function, previous work uses the *age* information of variables as the secondary scoring function, where *age* is defined as the number of steps since the last time it is flipped. But the experimental results show that the use of only *age* cannot effectively guide the search process. Thus, to further choose a variable among these variables with the same best *score* value, we design a novel fragile scoring function denoted as *hhscore*.

Before introducing *hhscore*, we first introduce a necessary concept. For a PB constraint  $C_p$ ,  $gap(C_p) = \min\{b^p + a_{max}^p, sum(C_p)\}$  is used to denote the *satisfied threshold* of  $C_p$ , which plays a key role in our proposed *hhscore*. Based on the satisfied threshold of PB constraints, we define a fragile satisfied PB constraint in the following.

► **Definition 1.** For a satisfied PB constraint  $C_p$  (i.e.,  $SatL(C_p) \geq b^p$ ), the  $C_p$  is a fragile satisfied PB constraint if and only if  $SatL(C_p) < gap(C_p)$ .

On the one hand, if a satisfied PB constraint  $C_p$  is fragile, we think that flipping any variable in this PB constraint would probably make this constraint become unsatisfied. On the other hand, if a satisfied PB constraint is not fragile (i.e.,  $SatL(C_p) \geq gap(C_p)$ ), we think this PB constraint is solid, which means that flipping any true literal in  $C_p$  would not make  $C_p$  become unsatisfied with a high probability. Although we have also tried a similar property that measures the number of true literals in a satisfied PB constraint  $C_p$ , such as  $gap(C_p) = b^p + 1$ , we did not find it useful in our algorithm.

To maintain the information of “fragile” for each literal during the search process, we define *inner* as below.

► **Definition 2.** Suppose that a Boolean variable  $x_i$  whose literal  $l_i$  appears in some PB constraints (e.g.,  $C_p$ ) and the coefficient of  $l_i$  in  $C_p$  is  $a_i^p$ . The value of  $inner(x_i, C_p)$  is calculated as follows.

- (a)  $C_p$  is an unsatisfied PB constraint, i.e.,  $SatL(C_p) < b^p$ :
  - If  $l_i$  is a true literal in  $C_p$ ,  $inner(x_i, C_p) = 0$ ;
  - If  $l_i$  is a false literal in  $C_p$ ,  $inner(x_i, C_p) = \max\{a_i^p - (b^p - SatL(C_p)), 0\}$ .
- (b)  $C_p$  is a fragile satisfied PB constraint, i.e.,  $b^p \leq SatL(C_p) < gap(C_p)$ :
  - If  $l_i$  is a true literal in  $C_p$ ,  $inner(x_i, C_p) = -\min\{a_i^p, SatL(C_p) - b^p\}$ ;
  - If  $l_i$  is a false literal in  $C_p$ ,  $inner(x_i, C_p) = \min\{a_i^p, gap(C_p) - SatL(C_p)\}$ .
- (c)  $C_p$  is satisfied but not a fragile PB constraint, i.e.,  $gap(C_p) \leq SatL(C_p)$ :
  - If  $l_i$  is a true literal in  $C_p$ ,  $inner(x_i, C_p) = -\max\{a_i^p - (SatL(C_p) - gap(C_p)), 0\}$ ;
  - If  $l_i$  is a false literal in  $C_p$ ,  $inner(x_i, C_p) = 0$ .

The value of  $inner(x_i, C_p)$  is subject to three distinct factors including the state of  $C_p$  (i.e., *satisfied* or *unsatisfied*), the value of  $l_i$  (i.e., *true* or *false*), and the coefficient of  $l_i$  (i.e.,  $a_i^p$ ). To make the readers easily understand the above Definition 2, we list all the situations corresponding to different *inner* values in Figure 1.

Considering all the PB constraints in which variable  $x$ 's literal appears, the proposed fragile scoring function *hhscore* of variable  $x_i$  is defined as below.

$$hhscore(x_i) = \sum_{p=1}^{n_x} inner(x_i, C_p) \quad (3)$$

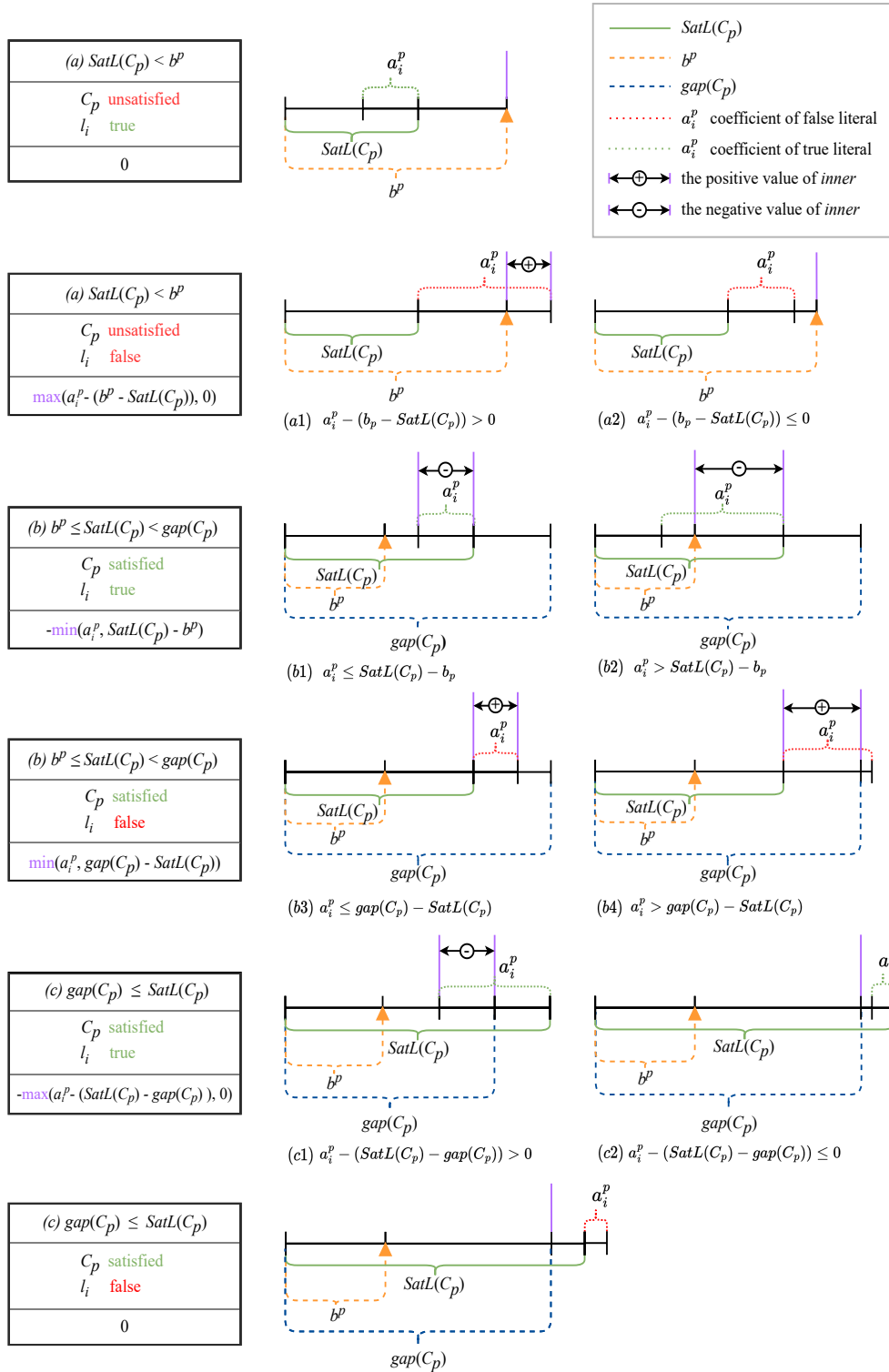
where  $n_x$  is the number of PB constraints including the literal of  $x_i$ .

## 3.2 Selection Rule

Combining the respective advantages of *score* and *hhscore*, we propose a two-level selection strategy as follows.

**Flipping Rule.** Flip a variable  $x_i$  with the biggest  $score(x_i)$  value, breaking ties by preferring the one with the biggest  $hhscore(x_i)$  value, further ties are broken randomly.

The proposed secondary scoring function is inspired by *subscore* [7], but has two essential differences. First, our proposed scoring function can be considered as a general version of *subscore* because the value of satisfied threshold *gap* is equal to 2 for the SAT problem, which is the same function as *subscore*. Second, previous work uses a linear combination of *subscore* and another scoring function as the primary scoring function, whereas our work considers a two-level scoring function to guide the search process. In addition, experiments show that the trigger fraction of using *hhscore* at least once for all tested instances is about 99.6%.



■ Figure 1 A graphical explanation of *inner*.

■ **Algorithm 1** DeepOpt.

---

**Input:** PBO instance  $F$  and a perturbation initialization assignment  $\alpha$   
**Output:** A perturbation solution  $\alpha$  of  $F$

```

1  $UnSatSet := unsat(\alpha)$  and  $SatSet := F \setminus UnSatSet$ ;
2  $CandSet := \emptyset$ ;
3 while  $|CandSet| \leq \gamma \times n$  do
4   if  $UnSatSet \neq \emptyset$  then
5     select a random unsatisfied PB constraint  $C$ ;
6      $UnSatSet := UnSatSet \setminus \{C\}$ ;
7     for each variable  $x \in C$  do
8       if  $x$ 's literal is true && with 50% probability then  $\alpha := \alpha$  with  $x$  flipped ;
9        $CandSet := CandSet \cup \{x\}$ ;
10  else
11    select a random satisfied PB constraint  $C$ ;
12     $SatSet := SatSet \setminus \{C\}$ ;
13    for each variable  $x \in C$  do
14       $CandSet := CandSet \cup \{x\}$ ;
15  if  $|unsat(\alpha)| > MaxHard$  then break ;
16 for  $step = 0; step < L_{opt}; step++$  do
17   select a variable  $x$  among  $|CandSet|/2$  samples from  $CandSet$  based on Flipping Rule;
18    $\alpha := \alpha$  with  $x$  flipped;
19 return  $\alpha$ ;
```

---

## 4 Deep Optimization for PBO

Local search algorithms usually search the entire space and focus on exploring some promising spaces using several heuristic strategies. Recently, a general perturbation mechanism called deep optimization has been proposed by Chen et al. [8], which can deeply probe some regions based on locked and unlocked operations and then converge to a new solution quickly. Note that deep optimization is somewhat similar to some classic search frameworks such as large neighborhood search [25]. According to this general framework, we propose a new deep optimization approach DeepOpt for PBO.

The pseudo-code of our proposed DeepOpt is reported in Algorithm 1 and the corresponding trigger conditions of DeepOpt will be displayed in the next section. We use the  $unsat(\alpha)$  function to denote the set of unsatisfied PB constraints under an assignment  $\alpha$ . At first, the algorithm uses  $UnSatSet$  and  $SatSet$  to store unsatisfied and satisfied PB constraints under a perturbation initialization assignment, respectively (Line 1). Afterward, a candidate variable set  $CandSet$  is initialized to an empty set (Line 2), which stores all unlocked variables in the following search. Note that, in our work, we use  $CandSet$  to store all unlocked variables which means that these variables can be flipped in the following search phase, while the remaining variables can be seen as locked variables.

The DeepOpt usually consists of two phases including a selection phase (Lines 3–15) to generate several candidate local search spaces and a search phase (Lines 16–18) to repair these search spaces.

In the first phase, we flip several variables to achieve the purpose of early preparation. There are two exit conditions in this phase. The first one is that  $|CandSet|$  is larger than  $\gamma \times n$  where  $n$  is the number of variables (Line 3), whereas the second one is that the number of unsatisfied constraints under the current assignment is larger than  $MaxHard$  (Line 15)

where  $\gamma$  and  $MaxHard$  are two parameters in DeepOpt. At each iteration, if  $UnSatSet$  is not empty, the algorithm preferentially chooses a random unsatisfied PB constraint  $C$  from  $UnSatSet$  (Lines 4–5) and puts all variables of  $C$  into  $CandSet$  (Line 9). For each variable  $x \in C$ , if  $x$ 's literal in  $C$  is true, it occurs with 50% probability to flip  $x$  (Line 8). If the algorithm selects a random satisfied constraint  $C$ , the algorithm only adds all variables of  $C$  into  $CandSet$  (Line 14). Note that according to our preliminary experiments, if the first phase only selects the candidate variables and does not flip these variables, then the performance of DeepOpt will become bad.

In the second phase, the algorithm applies the search process to perturb the assignment  $\alpha$  until the limit of iterations  $L_{opt}$  is reached (Line 16). The algorithm employs the BMS strategy [6] in the process of selecting a candidate variable, i.e., randomly choosing  $|CandSet|/2$  variables to compose a candidate set. Afterward, the algorithm flips a variable  $x$  among a candidate set based on the flipping rule (Lines 17–18). At last, the algorithm returns the final assignment  $\alpha$  (Line 19).

## 5 DeepOpt-PBO Algorithm

In this section, we propose an effective local search algorithm DeepOpt-PBO for PBO, whose main framework is presented in Algorithm 2. The DeepOpt-PBO is mainly divided into the initialization and search phases.

In the initialization phase (Lines 1–3), the best solution  $\alpha^*$  is set to  $\emptyset$  and its objective value  $obj^*$  is set to  $+\infty$ . To obtain an initial assignment  $\alpha$ , all variables are set to 0. The algorithm initializes the related weight information according to the weighting rule 1.

In the following, there is an outer loop (Lines 4–32) and an inner loop (Lines 7–31). During the search, whenever a better feasible assignment is obtained,  $\alpha^*$  and  $obj^*$  are updated accordingly (Line 12). After each inner loop, the algorithm uses the *RestartVar* function to restart a current assignment (Line 32), which will be presented in Section 5.2. Finally, the algorithm returns  $\alpha^*$  and  $obj^*$  when reaching a time limit.

In each inner loop ( $step < L$ ), the algorithm searches for a local optimal assignment  $\alpha$ . The algorithm uses *GoodSet* to store variables whose *score* is larger than 0 (Line 15). If *GoodSet* is not empty, the algorithm flips a candidate variable based on the proposed flipping rule (Lines 16–18). Otherwise, it means that the algorithm tramps into local optimum. The algorithm uses the modified weighting rule 2 and weighting rule 3 to update the corresponding weight value (Lines 20–21). Specifically, we optimize previous weighting rule 2, resulting in a novel modified weighting rule 2, which will be mentioned in Section 5.2. Moreover, the algorithm will use a sampling technique *SampleFlip* to flip one or more variables to help the algorithm itself jump out of local optimum, which will be introduced in the next subsection (Line 22).

**Trigger Conditions of DeepOpt.** In the below part, we will introduce some trigger conditions of DeepOpt in our proposed algorithm. At first, three variables are defined as below.

- 1) *unhard* is used to denote the number of unsatisfied PB constraints. Before each inner loop, *unhard* is initialized to the number of unsatisfied constraints under the current assignment (i.e.,  $|unsat(\alpha)|$ ) (Line 6). In the inner loop, whenever  $|unsat(\alpha)| < unhard$ , *unhard* is updated accordingly (Lines 8–9). After calling the DeepOpt method, *unhard* will be updated by  $|unsat(\alpha)|$  (Line 31).

■ **Algorithm 2** DeepOpt-PBO.

---

**Input** : PBO instance  $F$  and cutoff time  $cutoff$   
**Output** : An assignment  $\alpha^*$  of  $F$  and its objective value

```

1  $\alpha^* := \emptyset$  and  $obj^* := +\infty$ ;
2  $\alpha :=$  all variables are set to 0;
3 initialize the weight value of the objective function and each constraint to 1;
4 while  $elapsed\ time < cutoff$  do
5    $step_{opt} := 1$  and  $coff := 1$ ;
6    $unhard := |unsat(\alpha)|$ ;
7   for  $step = 1; step < L; step++$  do
8     if  $|unsat(\alpha)| < unhard$  then
9        $unhard := |unsat(\alpha)|$ ;
10       $step_{opt} := step_{opt}/2$  and  $step := step/2$ ;
11     if  $\alpha$  is feasible &&  $obj(\alpha) < obj^*$  then
12        $\alpha^* := \alpha$  and  $obj^* := obj(\alpha)$ ;
13        $step_{opt} := step := 1$ ;
14       if  $coff \neq 1$  then  $coff := coff/2$ ;
15      $GoodSet := \{x \mid score(x) > 0\}$ ;
16     if  $GoodSet \neq \emptyset$  then
17       select a variable  $x$  in  $GoodSet$  based on Flipping Rule;
18        $\alpha := \alpha$  with  $x$  flipped;
19     else
20       update the weight of each constraint based on Modified Weighting Rule 2;
21       update the weight of the objective function based on Weighting Rule 3;
22        $SampleFlip(F, \alpha)$ ;
23      $step_{opt} := step_{opt} + 1$ ;
24     if  $step_{opt} \% (coff \times MinStep) == 0$  then
25       if  $|unsat(\alpha)| \leq MinHard$  && with 50% probability then
26         if  $coff \neq \delta$  then  $coff := coff \times 2$ ;
27          $DeepOpt(F, \alpha)$ ;
28       else if  $\alpha^* \neq \emptyset$  then
29          $\alpha := \alpha^*$ ;
30          $DeepOpt(F, \alpha)$ ;
31        $unhard := |unsat(\alpha)|$  and  $step_{opt} := 1$ ;
32    $RestartVar(F, \alpha)$ ;
33 return  $(\alpha^*, obj^*)$ ;
```

---

- 2)  $step_{opt}$  records the non-improvement steps after the initialization phase or the last DeepOpt operation. Before each inner loop, the algorithm sets  $step_{opt}$  to 1 (Line 5). In each iteration of the inner loop,  $step_{opt}$  is increased by 1 (Line 23). When the algorithm obtains a better assignment (Line 13) or the algorithm calls the DeepOpt method (Line 27 or 30),  $step_{opt}$  will be set to 1 (Line 31). If  $|unsat(\alpha)| < unhard$ , then  $step_{opt}$  will be cut in half (Lines 8 and 10).
- 3)  $coff$  is used to control the frequency of using the DeepOpt method. Before each inner loop,  $coff$  is initialized to 1 (Line 5). When the algorithm finds a better assignment, the value of  $coff$  is divided by 2 to reduce the frequency of perturbations (Line 14). If  $|unsat(\alpha)|$  is smaller than parameter  $MinHard$ , it means unsatisfied PB constraints are few enough to consider perturbations, avoiding tramping into a local optimum. The value of  $coff$  will be doubled with a 50% probability (Line 26). During this process, parameter  $\delta$  controls the maximum value of  $coff$ .

## 41:10 Improving Local Search for Pseudo Boolean Optimization

The algorithm judges whether to call the DeepOpt method under each ( $coeff \times MinStep$ ) iteration (Line 24). If  $|unsat(\alpha)| \leq MinHard$ , the algorithm calls the DeepOpt method with a 50% probability (Line 27). Otherwise, if  $\alpha^*$  is not empty, the algorithm will use  $\alpha^*$  as a perturbation initialization assignment and then employ the DeepOpt method (Lines 28–30).

### Algorithm 3 SampleFlip.

---

```

Input : PBO instance  $F$  and an assignment  $\alpha$ 
Output : A modified assignment  $\alpha$ 
1 if  $|unsat(\alpha)| == 0$  then
2   select a random variable  $x$  with  $oscore(x) > 0$ ;
3    $\alpha := \alpha$  with  $x$  flipped;
4   return  $\alpha$ ;
5 select a random unsatisfied constraint  $C$ ;
6  $CSet := VSet := \emptyset$ ;
7 if  $|unsat(\alpha)| \geq \beta$  then
8   for  $i = 1$  to  $\beta$  do
9     select a random unsatisfied constraint  $C_i$ ;
10     $CSet := CSet \cup \{C_i\}$ ;
11 if  $|L(C)| == 1$  then
12   select only variable  $x$  in  $C$  and  $\alpha := \alpha$  with  $x$  flipped;
13 else if  $|L(C)| == 2$  then
14   /* Two variables  $x_1$  and  $x_2$  in  $C$  */
15   select a variable  $x_i$  with the largest  $score_t(x_1, x_i)$  value, breaking ties randomly;
16   select a variable  $x_j$  with the largest  $score_t(x_2, x_j)$  value, breaking ties randomly;
17   if  $score_t(x_1, x_i) > 0 \parallel score_t(x_2, x_j) > 0$  then
18     if  $score_t(x_1, x_i) > score_t(x_2, x_j)$  then
19        $\alpha := \alpha$  with  $x_1$  and  $x_i$  flipped;
20     else  $\alpha := \alpha$  with  $x_2$  and  $x_j$  flipped;
21   else
22     select a variable  $x$  among  $x_i$  and  $x_j$  based on Flipping Rule and  $\alpha := \alpha$  with  $x$ 
23     flipped;
24 else
25   if  $|unsat(\alpha)| \geq \beta$  then
26     for each constraint  $C_p \in CSet$  do
27       select  $|L(C_p)|/2$  samples from  $L(C_p)$  and put them into  $VSet$ ;
28     else
29       select  $|L(C)|/2$  samples from  $L(C)$  and put them into  $VSet$ ;
30     select a variable  $x$  from  $VSet$  based on Flipping Rule;
31      $\alpha := \alpha$  with  $x$  flipped;
32 return  $\alpha$ ;

```

---

## 5.1 Sampling Flipping

The pseudo-code of *SampleFlip* is outlined in Algorithm 3. First, we introduce a new scoring function, which is used in our *SampleFlip* method. When flipping two variables  $x_s$  and  $x_z$  simultaneously,  $score_t(x_s, x_z)$  is defined as the sum of the decrease of the total penalty of unsatisfied PB constraints and the objective function. Although it is easy to see the computation complexity of  $score$  is quite lower than  $score_t$ ,  $score_t$  can find a better

flipping operation compared to *score*. The method of selecting more candidate elements simultaneously has also been used in some different NP-hard problems [28, 34]. Moreover, in the *SampleFlip* method, we only consider  $score_t$  in a special case that the number of literals in the selected constraint  $C$  is 2 (i.e.,  $|L(C)| = 2$ ).

In the beginning, if there are no unsatisfied constraints, the algorithm selects and flips a random variable  $x$  with  $oscore(x) > 0$  (Lines 1–4). There are two main important components, including clause sampling (Lines 5–10) and variable flipping (Lines 11–29) in the algorithm. In the phase of clause sampling, the algorithm adopts two kinds of sampling ways. The first sampling method is to select a random unsatisfied constraint  $C$  (Line 5). If  $|unsat(\alpha)| > \beta$ , then the algorithm activates the second sampling method, i.e., adding  $\beta$  unsatisfied constraints into  $CSet$  (Lines 7–10).

In the phase of variable flipping, if  $C$  is a unit clause (i.e.,  $|L(C)| = 1$ ), the algorithm flips the only variable  $x$  in  $C$  (Lines 11–12). If  $C$  is a binary clause (i.e.,  $L(C) = \{x_1, x_2\}$ ), the algorithm tries to find two pairs of variables  $\{x_1, x_i\}$  and  $\{x_2, x_j\}$  with the largest  $score_t$  value (Lines 14–15). If there exists a positive flipping operation among these two pairs, the algorithm flips a pair with the better  $score_t$  value (Lines 16–19). Otherwise, the algorithm still flips only one variable among  $x_1$  and  $x_2$  based on the flipping rule (Lines 20–21). In the subsequent process, the algorithm will depend on the value of parameter  $\beta$  to collect some samples from  $CSet$  or  $C$  into  $VSet$  (Lines 23–27). At last, the algorithm selects and then flips the best variable  $x$  from  $VSet$  (Lines 28–29).

The intuitive explanations behind *SampleFlip* come from two aspects. First, single flipping mechanism for local search is easy to fall into a local optimum, while sampling strategy can explore the solution space more effectively in a look-ahead way. Second, the sampling strategy selects variables from various unsatisfied constraints, providing more search directions. In addition, we specially focus on the case of two literals, to keep running time low but a good performance.

## 5.2 Some Other Techniques

Based on the main part as shown above, we also introduce some additional heuristics to further improve the efficiency, including a weighting rule and a restart strategy.

**Modified Weighting Rule 2.** The weighting scheme plays an important role in the search process. By increasing the weight of unsatisfied constraints, we can accurately guide the search process toward more efficient directions. In each iteration of the inner loop, if  $GoodSet$  is empty, the visited times of each unsatisfied constraint will be increased by 1. For an unsatisfied constraint  $C_p$ , when its visited times are larger than  $b^p / coeff(C_p)$ , the value of  $w(C_p)$  will be increased by 1, and the value of its visited times is reset to 0.

**Restart Strategy.** The second adopted technique is the restart strategy. Under a current assignment  $\alpha$ , we can change the original value of each variable with a certain probability, i.e., from 1 (0) to 0 (1). Besides, the restart strategy will reset the weight value of each constraint according to the weighting rule 1. If the number of literals in an objective function  $O$  is larger than  $\beta$  (i.e.,  $|L(O)| > \beta$ ), then  $w(O)$  will be reset based on the weighting rule 1, too.

## 6 Experimental Evaluation

We first introduce the seven selected benchmarks, three state-of-the-art PBO competitors, and the adopted experimental setup. Then, we carry out extensive experiments to evaluate the performance of our proposed algorithm.



## 6.1 Experiment Preliminaries

Since the literature on local search algorithms for handling PBO is very sparse, we selected all used instances from [18, 10]. To be specific, we considered 3738 instances obtained from three application benchmarks and four standard benchmarks: (1) 24 instances from the minimum-width confidence band problem (MWCB) [2]. These MWCB instances were obtained based on the MIT-BIH arrhythmia database<sup>2</sup>. (2) 18 instances from the wireless sensor network optimization problem (WSNO) [16, 17]. We used the same encoding as previous work [18] to obtain an optimization version of WSNO. (3) 21 instances from the seating arrangements problem (SAP) [1]. These instances were originally proposed in the MaxSAT Evaluation 2017. (4) 1600 OPT-SMALL-INT instances from the most recent PB Competition in 2016 (PB2016)<sup>3</sup>. The PB2016 benchmark is often considered as the main target for comparing against some other PB solvers [10, 27]. (5) The 0-1 integer linear programming optimization benchmark (MIPLIB) contains 267 instances from the mixed integer programming library MIPLIB 2017<sup>4</sup>. (6) 1025 crafted combinatorial instances (CRAFT) are provided in the literature [30]. (7) The Knapsack benchmark (KNAP) consists of a total of 783 instances [22].

■ **Table 1** Tuned parameters of our proposed algorithms.

Parameter	Range	Final value
DeepOpt		
<i>MinStep</i>	{ $10^3, 10^4, 10^5, 10^6$ }	$10^5$
$\delta$	{64, 128, 256}	128
$\gamma$	{0.02, 0.05, 0.08, 0.11}	0.05
<i>MaxHard</i>	{30, 50, 70, 90}	50
<i>MinHard</i>	{5, 10, 15}	10
<i>L<sub>opt</sub></i>	{10, 30, 50, 70}	50
Some other parameters in our proposed algorithms		
$\beta$	{50, 100, 150, 200}	100
<i>L</i>	{ $10^2, 10^3, 10^4, 10^5$ }	$10^4$

According to the frequency of using the proposed restart strategy, we propose two versions of DeepOpt-PBO, resulting in DeepOpt-PBO-v1 and DeepOpt-PBO-v2. In detail, DeepOpt-PBO-v1 does not use the restart strategy (i.e., parameter  $L$  is set to INT\_MAX). DeepOpt-PBO-v2 is run for half of the computation time given to the algorithm with the restart strategy, while the second half of the available computation time of DeepOpt-PBO-v2 is given to the algorithm without the restart strategy. We also attempted to use the restart strategy during the whole time, and the result is not satisfactory. According to our preliminary experiments by using the automatic configuration tool irace [20], Table 1 shows the parameter values. Specifically, since these benchmarks have different scales, we built a training set and randomly selected 10 instances from the corresponding tested benchmark. The tuning process is given a budget of 5000 runs for the training set with a time budget of 3600s per run.

The proposed algorithms are compared against three state-of-the-art PBO algorithms, including a local search algorithm LS-PBO [18] and two exact PBO solvers, i.e., RoundingSat [10] and PBO-IHS [27]. Although mixed integer programming solvers (e.g., SCIP [4]) and

<sup>2</sup> <http://physionet.org/physiobank/database/mitdb/>

<sup>3</sup> <http://www.cril.univ-artois.fr/PB16/>

<sup>4</sup> <http://miplib.zib.de>

MaxSAT solvers (e.g., CASHWMaxSAT-CorePlus [19]) can be directly applied to solving PBO, we mainly focus on evaluating the performance of specialized solvers for PBO. The codes of all competitors were kindly provided by the authors. As for LS-PBO, RoundingSat and PBO-IHS, we employ the default parameters in the corresponding literature, respectively. Our code will be made publicly available. All algorithms are implemented in C++ and compiled by g++ with -O3 option. All the algorithms are run on Intel Xeon Gold 6238 CPU @ 2.10GHz with 512GB RAM under CentOS 7.9.

For the application benchmarks, our proposed algorithms and LS-PBO are both run 20 times whose seed is from 1 to 20 on each instance, whereas the exact solvers are run once on each instance. For the other four standard benchmarks (i.e., PB2016, MIPLIB, CRAFT and KNAP), following the settings of previous works [10, 27], all the algorithms are run only once on each instance. We test the algorithms with a time limit of 3600 seconds.

For the application benchmarks, we use *min* to denote the best solution value found and *avg* to denote the average value over the 20 runs. For all the benchmarks, we report the number of instances where the algorithm finds the best solution value among all algorithms, denoted by *#win*. There are some unsatisfied instances in the PB2016 benchmark. RoundingSat and PBO-IHS can guarantee the optimality of the solutions they obtain and thus can prove some of these unsatisfied instances, whereas DeepOpt-PBO and LS-PBO cannot do it because these two algorithms belong to incomplete algorithms. For the above case, following the similar method from the literature [18], if all the algorithms fail to obtain any feasible solution for such an unsatisfied instance, then *#win* value of all the algorithms for this instance needs to be increased by 1. The bold value indicates the best solution value obtained by all the algorithms. In detail, the bold value of each *avg* column indicates the best average solution when some algorithms obtain the same minimal solution values. For one instance, if only one algorithm finds the best minimal solution value, only its corresponding *min* column should be marked.

## 6.2 Experimental Results

Note that for the application benchmarks, two exact solvers RoundingSat and PBO-IHS can obtain the same best solution as our proposed algorithms for only 4 instances. Thus, for the sake of space, we do not report the detailed results of these two exact solvers. We mainly compare DeepOpt-PBO-v1 and DeepOpt-PBO-v2 with LS-PBO. The experimental results on the application benchmarks are presented in Tables 2–4. According to our results, our proposed algorithms are consistently superior on the MWCB and SAP benchmarks. For the WSNO benchmark, two proposed algorithms and LS-PBO can obtain the same best solution values. Furthermore, LS-PBO can find a minimal average solution for 15 instances, while our proposed algorithms do it for only 6 instances. Our proposed algorithms adopt some perturbation mechanisms, which may make them difficult for the algorithms to steadily obtain a good solution.

Table 5 gives a summary on all the benchmarks. According to the experimental results, DeepOpt-PBO-v2 outperforms other algorithms in terms of obtaining more optimal solution values on PB2016, MIPLIB and CRAFT, except for the benchmark KNAP where PBO-IHS has a better performance. In addition, under a given time limit, we observe that PBO-IHS can prove the optimality of a solution for more instances compared to RoundingSat in all benchmarks. Although our proposed algorithms cannot prove the optimality due to the natural property of local search, DeepOpt-PBO-v2 can find more minimal solution values overall. Because some instances from the above benchmarks have different kinds of problem structures and all the local search algorithms are run only once on each instance, the restart

■ **Table 2** Experiment results on MWCB.

Instance	DeepOpt-PBO-v1		DeepOpt-PBO-v2		LS-PBO	
	min	avg	min	avg	min	avg
1000_200_90	<b>103363</b>	104188.35	103430	104233	110450	111314.25
1000_250_90	<b>140223</b>	141182.2	140237	141194.9	148181	149828.65
1200_200_90	<b>104063</b>	<b>104572.25</b>	<b>104063</b>	104589.1	110993	112897.1
1200_250_90	<b>141211</b>	142310.6	141238	142343.2	150212	152888.7
1400_200_90	<b>103948</b>	104867.35	104057	104901.55	110792	112975.65
1400_250_90	<b>141567</b>	142675.7	141746	142722.15	150981	152932.9
1600_200_90	<b>119226</b>	<b>120182.4</b>	<b>119226</b>	120215.9	136944	143371.9
1600_250_90	<b>162651</b>	163893.3	162656	163937.1	183797	196547.75
1800_200_90	<b>203135</b>	<b>205888.75</b>	<b>203135</b>	205959.2	219536	224144.95
1800_250_90	<b>253073</b>	257501.75	253078	257715.25	276336	283192.95
2000_200_90	<b>227294</b>	229591.95	227424	229676.1	246109	250958.4
2000_250_90	<b>286970</b>	<b>289889.4</b>	<b>286970</b>	290058.1	309645	314528.5
1000_200_95	<b>113384</b>	114749.8	113501	114855.95	117064	117945.7
1000_250_95	<b>151882</b>	153581.05	152007	153622	156543	157976.1
1200_200_95	<b>114585</b>	115920.2	114675	115975	118045	119544.4
1200_250_95	<b>153104</b>	155765.8	153138	155798.35	159310	161454.1
1400_200_95	<b>114195</b>	<b>115231.3</b>	<b>114195</b>	115305.95	118913	119779.4
1400_250_95	<b>155158</b>	156149.65	155174	156180.15	161658	162917.95
1600_200_95	<b>168271</b>	<b>171985.9</b>	<b>168271</b>	172024.6	185707	190763.55
1600_250_95	<b>215266</b>	218013.5	215316	218069.55	236547	244060.35
1800_200_95	<b>238699</b>	241690.55	238702	241779.85	251744	256988.75
1800_250_95	<b>297981</b>	<b>302487.3</b>	<b>297981</b>	302927.25	314968	318961.25
2000_200_95	<b>257696</b>	<b>262062.9</b>	<b>257696</b>	262113.6	272832	277406.7
2000_250_95	<b>324379</b>	<b>328952.6</b>	<b>324379</b>	329032.9	340859	346007.95

■ **Table 3** Experiment results on WSNO.

Instance	DeepOpt-PBO-v1		DeepOpt-PBO-v2		LS-PBO	
	min	avg	min	avg	min	avg
100_40_4	<b>210</b>	<b>210</b>	<b>210</b>	<b>210</b>	<b>210</b>	<b>210</b>
150_60_4	<b>602</b>	612.85	<b>602</b>	<b>602</b>	<b>602</b>	602.2
200_80_4	<b>715</b>	719.45	<b>715</b>	716.65	<b>715</b>	<b>715.1</b>
250_100_4	<b>1305</b>	1401.2	<b>1305</b>	1330.05	<b>1305</b>	<b>1305</b>
300_120_4	<b>1257</b>	1330.25	<b>1257</b>	1373	<b>1257</b>	<b>1257.05</b>
350_140_4	<b>1737</b>	1957.4	<b>1737</b>	1997.95	<b>1737</b>	<b>1744.05</b>
400_160_4	<b>2240</b>	2509.55	2241	2598.85	<b>2240</b>	<b>2240.5</b>
450_180_4	<b>1869</b>	2780.7	1878	2598.7	<b>1869</b>	<b>1889.25</b>
500_200_4	<b>2577</b>	3676.8	2674	3637.95	<b>2577</b>	<b>2616.2</b>
100_40_6	<b>140</b>	<b>140</b>	<b>140</b>	<b>140</b>	<b>140</b>	<b>140</b>
150_60_6	<b>402</b>	<b>402</b>	<b>402</b>	<b>402</b>	<b>402</b>	<b>402</b>
200_80_6	<b>477</b>	480	<b>477</b>	<b>477.05</b>	<b>477</b>	477.7
250_100_6	<b>870</b>	893.35	<b>870</b>	<b>870.1</b>	<b>870</b>	870.5
300_120_6	<b>839</b>	866.55	<b>839</b>	862.15	<b>839</b>	<b>839.3</b>
350_140_6	<b>1158</b>	1267.7	<b>1158</b>	1288.25	<b>1158</b>	<b>1158.85</b>
400_160_6	<b>1493</b>	1671.8	<b>1493</b>	1656.5	<b>1493</b>	<b>1494.25</b>
450_180_6	<b>1246</b>	1588.45	1265	1641.6	<b>1246</b>	<b>1247.8</b>
500_200_6	<b>1718</b>	1984.05	<b>1718</b>	1927	<b>1718</b>	<b>1727.65</b>

strategy plays a key role in the performance of DeepOpt-PBO-v2. To sum up, for three application benchmarks, PB2016, MIPLIB, and CRAFT, our algorithm totally dominates all the competitors, whereas PBO-IHS performs better than other PBO solvers for the KNAP benchmark.

Additionally, we evaluate the performance of all the solvers on seven benchmarks using performance profile [11]. As shown in Figure 2, the plot captures the probability of reaching a fixed quality in a time, at most a factor  $\tau$  slower than the optimal algorithm. Specially,

■ **Table 4** Experiment results on SAP.

Instance	DeepOpt-PBO-v1		DeepOpt-PBO-v2		LS-PBO	
	min	avg	min	avg	min	avg
100	<b>579</b>	<b>579.7</b>	<b>579</b>	579.8	580	583
110	<b>618</b>	<b>618.75</b>	<b>618</b>	618.85	619	626.05
120	<b>675</b>	<b>677.15</b>	<b>675</b>	677.8	679	685.6
130	<b>733</b>	736.9	734	738.4	738	744.6
140	<b>750</b>	<b>751.9</b>	<b>750</b>	753.75	757	763.65
150	<b>803</b>	<b>808.4</b>	<b>803</b>	810.3	821	828.45
160	<b>849</b>	<b>854.85</b>	<b>849</b>	855.75	867	872.75
170	<b>880</b>	<b>884.4</b>	<b>880</b>	885.55	897	907.5
180	<b>939</b>	948.45	941	951.7	971	977.4
190	<b>965</b>	973.2	970	977	996	1005.25
200	<b>1029</b>	<b>1040</b>	<b>1029</b>	1042.55	1067	1073.4
210	<b>1067</b>	1074.7	1068	1077.9	1094	1112.75
220	<b>1114</b>	<b>1127.3</b>	<b>1114</b>	1129.6	1151	1163.1
230	<b>1151</b>	1166.85	1162	1169.7	1195	1205.75
240	<b>1179</b>	1192.6	1183	1195.5	1219	1234.55
250	<b>1235</b>	1243.1	1237	1245.5	1274	1290.55
260	<b>1275</b>	1286.15	1277	1287.45	1318	1334.9
270	<b>1344</b>	1353.1	1348	1356.1	1392	1403.55
280	<b>1348</b>	1370.75	1354	1375.35	1407	1424.3
290	<b>1403</b>	1416.35	1405	1419.5	1448	1471.15
300	<b>1471</b>	1491.5	1480	1496.1	1538	1548.4

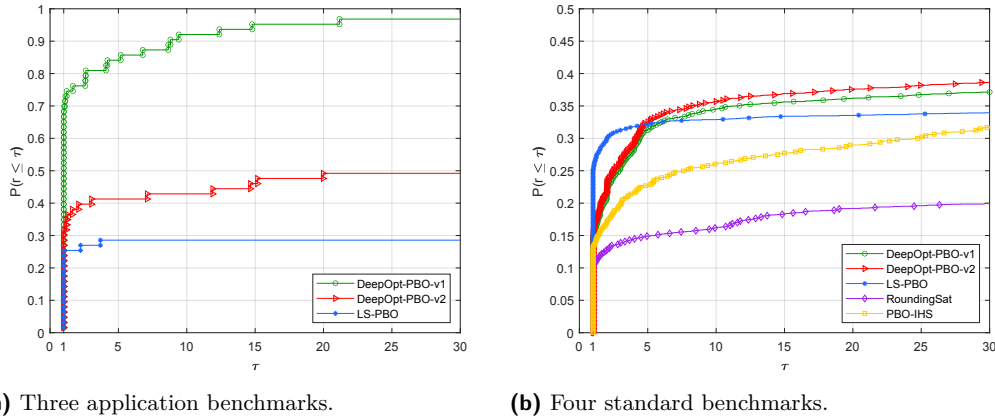
■ **Table 5** Summary results of comparing our proposed algorithms to its competitors on all the benchmarks. #inst denotes the number of instances in each benchmark.

Benchmark	#inst	DeepOpt-PBO-v1	DeepOpt-PBO-v2	LS-PBO	RoundingSat	PBO-IHS
		#win	#win	#win	#win	#win
MWCB	24	<b>24</b>	9	0	0	0
WSNO	18	<b>18</b>	14	<b>18</b>	4	4
SAP	21	<b>21</b>	9	0	0	0
PB2016	1600	1141	<b>1226</b>	1060	1195	1101
MIPLIB	267	163	<b>168</b>	135	125	130
CRAFT	1025	903	<b>930</b>	878	882	917
KNAP	783	695	693	649	392	<b>778</b>
Total	3738	2965	<b>3049</b>	2740	2598	2930

■ **Table 6** Comparative results for DeepOpt-PBO-v1 and its modified versions with different strategies on the application benchmarks. #better and #worse denote the number of instances where DeepOpt-PBO-v1 obtains better and worse solution values, respectively.

Benchmark	#inst	vs. v1+nohh		vs. v1+nodo		vs. v1+nosf	
		#better	#worse	#better	#worse	#better	#worse
MWCB	24	<b>11</b>	8	<b>24</b>	0	<b>24</b>	0
WSNO	18	<b>1</b>	0	<b>1</b>	0	0	0
SAP	21	<b>11</b>	5	<b>18</b>	0	<b>20</b>	0
Total	63	<b>23</b>	13	<b>43</b>	0	<b>44</b>	0

when  $\tau$  equals 1, we obtain the probability that the algorithm is the fastest. Figure 2 (a) demonstrates the superior performance of DeepOpt-PBO-v1 on the first three application benchmarks. It can be observed that DeepOpt-PBO-v1 has a higher probability of finding the optimal solution for each  $\tau$ . In Figure 2 (b), LS-PBO performs the best within a short time scale. However, DeepOpt-PBO-v2 surpasses LS-PBO around  $\tau = 6$  and maintains its leadership consistently.



■ **Figure 2** Performance profiles for DeepOpt-PBO and all competitors for reaching the best solution on the three application benchmarks (a) and four standard benchmarks (b).

For three application benchmarks, to determine better solution values, we increase the execution time to 5400 seconds, and once again our algorithms perform better than LS-PBO. For the PB2016 benchmark, when increasing the execution time to 5400 seconds, although the solution values obtained by RoundingSat and PBO-IHS are better than before, our algorithm can still find more best solution values than RoundingSat and PBO-IHS for 29 and 140 instances, respectively.

To verify the effectiveness of the proposed strategies, we compare DeepOpt-PBO-v1 with three alternative versions: 1) v1+nohh utilizes the *age* strategy instead of *hhscore*; 2) v1+nodo does not use the DeepOpt method; 3) v1+nosf does not employ the *SampleFlip* function. The results in Table 6 demonstrate that all proposed strategies are effective. In addition, we randomly select 20 instances for each standard benchmark. In total, 80 instances are picked. We also test the performance of our proposed algorithm and three alternative versions on these picked instances. All the algorithms are run 20 times on each instance. Once again, the results show that our proposed algorithm obviously performs better than three alternative versions.

Here, we give a discussion to compare our algorithm with MIP solvers and MaxSAT solvers. First, although MaxSAT solvers (e.g., CASHWMaxSAT-CorePlus [19]) can be directly applied to solving PBO, Lei et al. [18] observed that several of the instances from some benchmarks, such as PB2016, are too large to admit practical encodings into MaxSAT. Thus, MaxSAT solvers have usually poor performance for PBO. Second, more general solvers (e.g., MIP solvers) could also be used for comparison, but in our work, we mainly focus on evaluating the performance of specialized solvers for PBO. We have also tested the performance of the non-commercial MIP solver SCIP [4] on all the seven benchmarks. In detail, for the KNAP benchmark, the performance of PBO-IHS and SCIP is better than our algorithm. But, for the remaining six benchmarks, our algorithms perform better than SCIP.

## 7 Conclusion

In this paper, we propose a two-level selection strategy, a novel deep optimization strategy, and a sampling flipping method. Based on the above strategies and some other tricks, we develop two local search algorithms. Experiments show that the proposed algorithms significantly outperform the state-of-the-art PBO algorithms.

---

**References**

---

- 1 Carlos Ansotegui, Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT evaluation 2017: Solver and benchmark descriptions, 2017.
- 2 Jeremias Berg, Emilia Oikarinen, Matti Järvisalo, and Kai Puolamäki. Minimum-width confidence bands via constraint optimization. In *Proceedings of the Twenty-Third Principles and Practice of Constraint Programming*, volume 10416, pages 443–459, 2017.
- 3 Daniel Le Berre and Romain Wallon. On dedicated CDCL strategies for PB solvers. In *Proceedings of the Twenty-Fourth Theory and Applications of Satisfiability Testing*, volume 12831, pages 315–331, 2021.
- 4 Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, et al. The SCIP optimization suite 8.0. *arXiv:2112.08872*, 2021.
- 5 Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the Sixteenth European Conference on Artificial Intelligence*, volume 16, pages 146–150, 2004.
- 6 Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research*, 59:463–494, 2017.
- 7 Shaowei Cai and Kaile Su. Comprehensive score: Towards efficient local search for SAT with long clauses. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 489–495, 2013.
- 8 Jiejiang Chen, Shaowei Cai, Yiyuan Wang, Wenhao Xu, Jia Ji, and Minghao Yin. Improved local search for the minimum weight dominating set problem in massive graphs by using a deep optimization mechanism. *Artificial Intelligence*, 314:103819, 2023.
- 9 Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 26(1):26–55, 2021.
- 10 Jo Devriendt, Stephan Gocht, Emir Demirović, Jakob Nordström, and Peter J. Stuckey. Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, volume 35, pages 3750–3758, 2021.
- 11 Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.
- 12 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 1291–1299, 2018.
- 13 Jan Elffers and Jakob Nordström. A cardinal improvement to pseudo-Boolean solving. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, volume 34, pages 1495–1503, 2020.
- 14 John Franco and John Martin. A history of satisfiability. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 3–74. 2009.
- 15 Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. The MIT press, 2009.
- 16 Gergely Kovásznai, Balázs Erdélyi, and Csaba Biró. Investigations of graph properties in terms of wireless sensor network optimization. In *Proceedings of the IEEE International Conference on Future IoT Technologies*, pages 1–8, 2018.
- 17 Gergely Kovásznai, Krisztián Gajdár, and Laura Kovács. Portfolio SAT and SMT solving of cardinality constraints in sensor network optimization. In *Proceedings of the Twenty-First International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 85–91, 2019.

- 18 Zhendong Lei, Shaowei Cai, Chuan Luo, and Holger H. Hoos. Efficient local search for pseudo Boolean optimization. In *Proceedings of the Twenty-Fourth Theory and Applications of Satisfiability Testing*, volume 12831, pages 332–348, 2021.
- 19 Zhendong Lei, Yiyuan Wang, Shiwei Pan, Shaowei Cai, and Minghao Yin. CASHWMaxSAT-CorePlus: Solver description. *MaxSAT Evaluation*, page 8, 2022.
- 20 Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.
- 21 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: A modular maxsat solver. In *Proceedings of the Seventeenth Theory and Applications of Satisfiability Testing*, volume 8561, pages 438–445, 2014.
- 22 David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.
- 23 Olivier Roussel and Vasco Manquinho. Pseudo-Boolean and cardinality constraints. In *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1087–1129. 2021.
- 24 Masahiko Sakai and Hidetomo Nabeshima. Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. *IEICE Transactions on Information and Systems*, E98.D(6):1121–1127, 2015.
- 25 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the Fourth Principles and Practice of Constraint Programming*, volume 1520, pages 417–431, 1998.
- 26 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Pseudo-Boolean optimization by implicit hitting sets. In *Proceedings of the Twenty-Seventh Principles and Practice of Constraint Programming*, volume 210, pages 51:1–51:20, 2021.
- 27 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Improvements to the implicit hitting set approach to pseudo-Boolean optimization. In *Proceedings of the Twenty-Fifth International Conference on Theory and Applications of Satisfiability Testing*, volume 236, pages 13:1–13:18, 2022.
- 28 Zhouxing Su, Qingyun Zhang, Zhipeng Lü, Chu-Min Li, Weibo Lin, and Fuda Ma. Weighting-based variable neighborhood search for optimal camera placement. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, volume 35, pages 12400–12408, 2021.
- 29 Renato Tinós, Michal W. Przewozniczek, and Darrell Whitley. Iterated local search with perturbation based on variables interaction for pseudo-Boolean optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 296–304, 2022.
- 30 Marc Vinyals, Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, and Jakob Nordström. In between resolution and cutting planes: A study of proof systems for pseudo-Boolean SAT solving. In *Proceedings of the Twenty-First Theory and Applications of Satisfiability Testing*, volume 10929, pages 292–310, 2018.
- 31 Chris Voudouris and Edward Tsang. Partial constraint satisfaction problems and guided local search. In *Proceedings of the Practical Application of Constraint Technology*, pages 337–356, 1996.
- 32 Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. Scwalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Artificial Intelligence*, 280:103230, 2020.
- 33 Yiyuan Wang, Dantong Ouyang, Liming Zhang, and Minghao Yin. A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity. *Science China Information Sciences*, 60:062103, 2017.
- 34 Jiongzhi Zheng, Jianrong Zhou, and Kun He. Farsighted probabilistic sampling based local search for (weighted) partial maxsat. *CoRR*, abs/2108.09988, 2021.

# Predict-Then-Optimise Strategies for Water Flow Control

Vincent Barbosa Vaz   




The University of Melbourne, Australia  
the Australian Research Council OPTIMA ITTC, Melbourne, Australia

James Bailey   

The University of Melbourne, Australia

Christopher Leckie   

The University of Melbourne, Australia

Peter J. Stuckey   

Monash University, Australia  
the Australian Research Council OPTIMA ITTC, Melbourne, Australia

---

## Abstract

A pressure sewer system is a network of pump stations used to collect and manage sewage from individual properties that cannot be directly connected to the gravity driven sewer network due to the topography of the terrain. We consider a common scenario for a pressure sewer system, where individual sites collect sewage in a local tank, and then pump it into the gravity fed sewage network. Standard control systems simply wait until the local tank reaches (near) capacity and begin pumping out. Unfortunately such simple control usually leads to peaks in sewage flow in the morning and evening, corresponding to peak water usage in the properties. High peak flows require equalization basins or overflow systems, or larger capacity sewage treatment plants. In this paper we investigate combining prediction and optimisation to better manage peak sewage flows. We use simple prediction methods to generate realistic possible future scenarios, and then develop optimisation models to generate pumping plans that try to smooth out flows into the network. The solutions of these models create a policy for pumping out that is specialized to individual properties and which overall is able to substantially reduce peak flows.

**2012 ACM Subject Classification** Applied computing → Operations research; Mathematics of computing → Mixed discrete-continuous optimization

**Keywords and phrases** Water Flow Control, Optimization, Machine Learning

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.42

**Category** Short Paper

**Funding** This research was partially funded by the Australian Government through the Australian Research Council Industrial Transformation Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), with industry partner South East Water, Project ID IC200100009.

## 1 Introduction

A Pressure Sewer System (PSS) is a network of pump stations used to collect and manage sewage from individual properties that cannot be directly connected to the classic gravity sewer network due to the topography of the terrain (gravity limitation). The sewerage gravitates to the pump station and is then pumped through a pressure main to a main sewer and on to wastewater treatment plants. This is illustrated in Figure 1a. A PSS is often composed of intermediate, bigger pump stations between sub parts of the whole network (Figure 1b). Pump stations collect household sewage from a sub part of the network and pump it to the main. Our



© Vincent Barbosa Vaz, James Bailey, Christopher Leckie, and Peter J. Stuckey;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

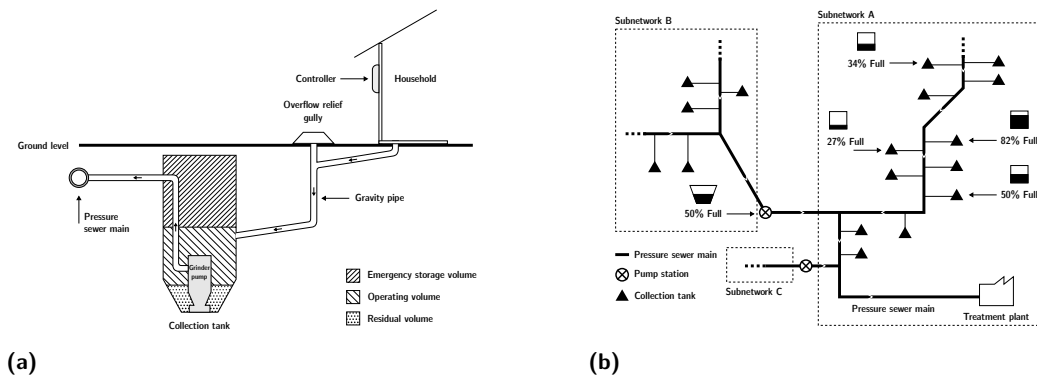
Editor: Roland H. C. Yap; Article No. 42; pp. 42:1–42:10

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany





■ **Figure 1** (a) Pressure sewer unit. (b) Pressure sewer network.

focus in this paper is to optimise the operation of individual pump stations to balance the overall load at the treatment plant. In this paper, we only consider an existing network managed by our industry partner South East Water. The network that we consider is free of intermediate pump stations. Extending the approach to handle intermediate pump stations would require adding constraints to ensure the capacity of the pump station is not violated at any time.

We consider a classic scenario for PSS where volumes at the residential level are released without optimised control. When the collection tanks reach their capacity, water is released entirely into the network. This represents a considerable amount of sewage conveyed through the sewer network that must be handled by the treatment plant, resulting in stress on the treatment processes and increased capital costs of upsized pipe and pump networks. This simple control policy does not make good use of the network. Volumes can be retained in pressure sewer tanks at the residential level and selectively released in a way that can optimise the flows to provide network capacity increases for the current network, and improve operations of the downstream treatment plant.

Most of the water usage occurs in the morning and the evening peak loads, when people are home. This results in two, identifiable peaks of activity in the network. This sudden surge in activity represents a challenge for the water treatment processes that follow. Huge amounts of resources need to be deployed at the treatment plant at these times to cope with the volumes to be treated and the associated unpredictability. South East Water would like to flatten the input flow at the treatment plant to achieve greater plant operational efficiency. This is possible by leveraging the buffer capabilities of the collection tanks, assuming that each tank can be controlled individually.

**Current operational strategy.** The water industry has yet to integrate data-driven models and optimisation techniques to facilitate and control their processes in a more efficient and systematic way. Most water companies rely on operators' knowledge and experience to parameterise and control their network. In current operation each tank fills until it reaches its capacity (cut-in setpoint) and then fully (until cut-off setpoint) empties the tank. In a previous approach to tackling the problem of reducing maximum outflows on the network we investigated simply modifying the (cut-in) set points of the tanks to try to reduce homogeneous behaviour but this was not really successful. Without adjustment the tanks quickly reached a steady state where the usual morning and evening peak inflows again resulted in high outflow. Due to the location of the network in a holiday zone the set points needed to be adjusted frequently as usage behaviour fluctuated rapidly over weekends, summer etc.

In this paper we define an optimisation based approach to controlling the maximum outflow, by deciding in which time periods to empty the tanks. In order to flatten out peaks we need to have some idea of the future inflows into the tanks. Because patterns of water usage are quite distinct generating realistic time series of inflows is challenging with machine learning models. Hence we use simple historical sampling to generate plausible future inflows. We show how combining (simplistic) prediction models with an optimisation model to determine when to release sewage into the network from individual properties, can substantially reduce the peak flows in the network.

## 2 Problem Description

We model the problem of controlling a PSS system to reduce peak flows as a MIP. We discretize the control problem by working over a finite time horizon of  $n$  discrete time steps, which for our experiments are always 1 hour long. Note that this discretization is fine enough that none of our historical data has examples where a tank is filled from empty in under an hour. Finer discretization would allow some further reduction in peak outflows, but we expect that we capture most of the possible reduction using 1 hour discretization. The parameters for the water flow control model are shown in Table 1. In each time step we decide whether to empty the tank at a particular site. In the default control mechanism, when we decide to pump out of a tank, we empty it. This has the least wear and tear on the pumping and control mechanism.

■ **Table 1** Parameters for the water flow control model.

Description	Parameter
Time horizon	$T = 1..n$
Set of tanks	$S$
Capacity of tank at site $s$ (cut-in setpoint)	$c_s$
Minimum amount of water to be pumped out	$m$
Inflow to tank at site $s$ during time period $t$	$i_{s,t}$
Tank level of site $s$ tank at the beginning of the first time period	$l_{s,0}$

### 2.1 Direct Formulation

In this section, we propose a direct MIP formulation for the PSS problem, over a given set of tank sites  $S$  and time horizon  $T$ . The control systems for the tanks have simple functionality, we can (re-)set minimum and maximum tank levels, or initiate a pump out of the tank, but not control exactly how much volume is pumped out of the tank. This leads to the important decision we must make – *for each tank  $s$  at what times  $t$  should it be emptied?*  $X_{s,t} \in \{0, 1\}$ . The model makes use of auxiliary variables:  $l_{s,t}$  the level of tank  $s$  at (the end of) time  $t$ ; and  $o_{s,t}$  the volume of water pumped out of tank  $s$  during time period  $t$ . The model is:

$$\text{minimize } \max_{t \in T} \sum_{s \in S} o_{s,t}$$

$$l_{s,t} = (l_{s,t-1} + i_{s,t}) * (1 - X_{s,t}), \quad \forall s \in S, t \in T \quad (1)$$

$$l_{s,t} \leq c_s, \quad \forall s \in S, t \in T \quad (2)$$

$$o_{s,t} = (l_{s,t-1} + i_{s,t}) * X_{s,t}, \quad \forall s \in S, t \in T \quad (3)$$

$$X_{s,t} = 1 \rightarrow o_{s,t} \geq m, \quad \forall s \in S, t \in T \quad (4)$$

$$X_{s,t} \in \{0, 1\}, \quad \forall s \in S, t \in T$$

where Equation (1) computes the level  $l_{s,t}$  in each tank  $s$  at the end of time period  $t$  (previous level plus inflows, unless emptied); Equation (2) ensures that each tank's level remains below tank capacity; Equation (3) computes the outflow  $o_{s,t}$  from tank  $s$  at time  $t$ ; and Equation (4) ensures that if the tank is emptied there is a minimum volume present (to prevent accelerated wear and tear on the infrastructure). The objective is to minimize maximum outflow across the period considered. Note that each of the constraints, and the objective are linear or easy to linearise.

## 2.2 Packing Formulation

The model above straightforwardly models the problem, but can become challenging to solve as the problem size grows. Next we instead consider the inflow to tank  $s$  at time  $t$  as a fixed amount of flow, we then decide when this should be pumped out. By precomputation we can then specify simple constraints to enforce that the capacity of tank is not exceeded. This leaves a packing problem, deciding when each chunk of water is pumped into the network.

The tank inflows are aggregated by hours and constitute the items of the problem. We require that the inflows are pumped out in order of inflow. The inflow at time  $t$ ,  $i_{s,t}$ , must be pumped out, no earlier than  $t$ , and not later than  $latest(s, t) = \min(\{t' - 1 \mid t \leq t' \leq n, \sum_{t \leq i \leq t'} i_{s,i} \geq c_s\} \cup \{n\})$  which would mean the tank capacity was exceeded, since no later flows can be pumped out before  $i_{s,t}$ . Note that we treat the starting tank level  $l_{s,0}$  as an inflow at time 0,  $i_{s,0} = l_{s,0}$ . We can also define the last inflow that must be pumped out in the considered time period,  $last(s) = \min\{t \mid \sum_{t \leq i \leq n} i_{s,i} \leq c_s\} - 1$ . And for each tank and time define  $below(s, t) = \max\{t' \mid t \leq t' < latest(s, t), \sum_{t < i \leq t'} i_{s,i} < m\}$  to be the latest time  $t'$  such that if the tank is emptied at time  $t$  the sum of inflows after it up to  $t'$  is too small to empty.

The new decision variables  $p_{s,t,t'}$  determine the time  $t'$  when the inflow to  $s$  at time  $t$  is pumped out (including the original tank volume  $t = 0$ ). The model is defined by:

$$\text{minimize } \max_{t \in T} \sum_{s \in S, t' \leq t} p_{s,t',t} \times i_{s,t} \quad (5)$$

$$\sum_{t \leq i \leq latest(s,t)} p_{s,t,i} = 1 \quad s \in S, t \in 0..last(s) \quad (6)$$

$$p_{s,0,0} = 0 \quad s \in S \quad (7)$$

$$\sum_{t \leq i \leq t'} p_{s,t,i} \geq p_{s,t+1,t'}, \quad \forall s \in S, t \in 1..n, t' \in t+1..latest(s, t) \quad (8)$$

$$X_{s,t'} = 1 \rightarrow \sum_{t \leq i \leq t'} p_{s,t,i} \geq 1, \quad s \in S, t \in T, t' \in t..latest(s, t) \quad (9)$$

$$\sum_{t' \leq t \leq latest(s,t')} p_{s,t',t} \geq 1 \rightarrow X_{s,t} = 1, \quad s \in S, t \in T \quad (10)$$

$$X_{s,t} = 1 \rightarrow \sum_{t < i \leq below(s,t)} X_{s,i} \leq 0, \quad \forall s \in S, t \in T \quad (10)$$

$$X_{s,t} \in \{0, 1\}, \quad \forall s \in S, t \in T$$

$$p_{s,t,t'} \in \{0, 1\}, \quad \forall s \in S, t \in 0..n, t' \in t..latest(s, t)$$

where Equation (5) enforces we pump out each inflow (before  $last(s)$ ) exactly once; Equation (6) enforces that nothing is pumped out at time 0; Equation (7) enforces that the inflow to tank  $s$  at time  $t$  is pumped out no later than the time the inflow at time  $t + 1$  is pumped out, i.e. the inflows are pumped out in order; Equation (8) connects the emptied variables to the

pump out variables by requiring that if tank  $s$  is emptied at time  $t'$  then each inflow before  $t'$  is pumped out by time  $t'$ ; Equation (9) connects them in the other direction requiring that if any inflow is pumped out of tank  $s$  at time  $t$  then tank  $s$  is emptied at time  $t$ ; and Equation (10) enforces that if tank  $s$  is emptied at time  $t$  then it is not emptied again until at least  $m$  units of flow have entered the tank. The objective minimizes maximum outflow, computed from the pumped out variables. Again the entire model is easy to linearise.

### 3 Predicted Water Usage Generation

Online optimisation problems can be augmented with a predictor that informs the model on future instances. Simple predictors can sample the inputs or sample the distribution of the inputs. More complex Machine Learning based predictors can learn from the distribution of the inputs as the online problem is being solved. Research [5, 1, 2, 3] has demonstrated that sampling the distribution of the inputs or providing estimates can significantly improve the quality of the solution. In practice, not all optimisation problems have access to the distribution of the inputs.

We wish to generate water usage predictions for each site. The collected water usage comes from diverse households exhibiting different behavioural patterns. Care must be taken when building a predictor to capture the seasonality and cycles in the data. Because of these properties, building an individual predictor for each site and time instance is unlikely to produce realistic distributions of water usage.

#### 3.1 Historical Sampling

We use *historical sampling* as described by Bent and Van Hentenryck [4]. This generates samples from past subsequences in the historical data. Despite its simplicity, historical sampling captures the structure of the sequence while providing fast outcomes compared to ML techniques that require training during the online algorithm. We adapt the algorithm to sample historical data while preserving the structural periodic information of our samples. In particular, we “retrieve” a prediction sequence on inflows for tank  $s$  for times  $T = 1..n$  from the historical data sequence  $S$  of inflows for tank  $s$  including  $\bar{d}$  days of data, by randomly selecting starting position  $t'$  which is the start of some day (since our experiments always commence from the first hour of a day) in  $S$ , where  $S[l..u]$  returns the slice of sequence  $S$  from index  $l$  to  $u$  inclusive.

■ **Algorithm 1** Historical sampling.

---

```

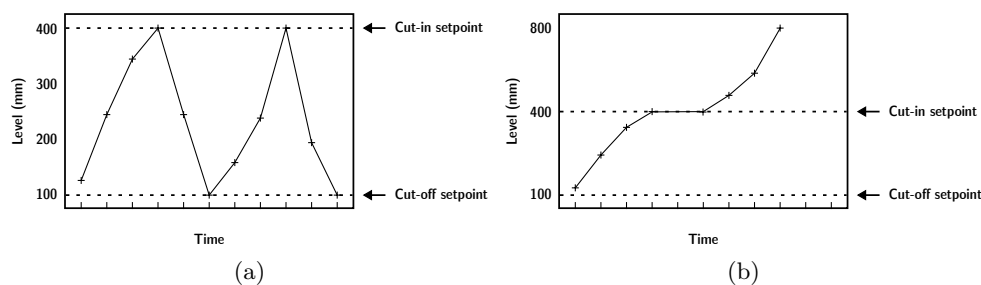
1 historicalSample( $S, n$ )
2    $\bar{d} \leftarrow |S| \text{ div } 24$ 
3    $t' \leftarrow 24 \times \text{RANDOM}([0, \bar{d} - 1]) + 1$ 
4   return  $S[t'..t' + n - 1]$ 

```

---

### 4 Experimental Evaluation

In this section, we present the results of using proposed models on a set of benchmark instances. The models are implemented in MiniZinc [8], a high-level and solver-independent modelling language, allowing for fast experimentation across existing solvers (OR-tools, Gurobi, CPLEX) without compromising on efficiency. All experiments were run on the same



■ **Figure 2** (a) Original level data available from the SCADA system, and (b) the reconstructed level (from where we compute inflow data).

machine which has an Apple M1 Pro 3.22 GHz CPU with 10 cores and 16 GB of RAM. All approaches were given a time limit of one hour per instance. The solver used was Gurobi version 9.5.2.

#### 4.1 Benchmark Instances

The data is provided by our industry partner, South East Water, and corresponds to pressure sewer readings. This catchment has been selected as it is of reasonable scale and is free of infiltration. The data is collected, through a series of scripts, from the SCADA server and corresponds to 3 years (2019-2021) of historical readings from approximately 4200 individual households. A range of attributes can be extracted from the readings; we focus on the water level and pump activation. The network is free of intermediate pump stations.

In order to make different decisions, we first need to rollback any previous decisions to obtain the original system inputs. The tanks are not equipped with water meters to measure the inflows but have level sensors. From the water level, we can derive the inflows to be the difference between two positive consecutive readings. This is illustrated in Figures 2(a) and 2(b). We generated inflow data for each site for each hour of the day in the periods used for instance generation.

We cluster similar sites using the  $k$ -means algorithm where distance is defined as the sum of absolute differences over their inflow data. We observe the average inflow for each cluster. We determined four identifiable water usage profiles amongst the households: two diurnal/bimodal (with a morning and afternoon peak) and two uni-modal usages (with just a morning peak), at a time translation respectively. We combined the bimodal and uni-modal sites respectively. The average inflow for each cluster is shown in Figure 4 in the supplementary material.

We created 6 problem instances from our industry partner. These instances represent different levels of complexity. To ensure they are different we choose different kinds of flows. For each instance we pick  $|S|$  sites to use, either from the unimodal clusters, the bimodal clusters, or the complete set of clusters. We choose a number of hours  $n$  to solve over and a uniform capacity  $C$  for each tank. For each instance we create 30 scenarios, corresponding to different date ranges for the actual flow, and generate different historical sampling predictions for each site in each of the scenarios. Details of the problem instances are shown in Table 2. We consider scenarios of 24 and 48 hours length, the 48 hour instances allow water to be kept overnight in the tanks in order to smooth the outflows. We also briefly explored longer scenarios of 1 week but they did not lead to much greater peak outflow reductions than 48 hour scenarios.

■ **Table 2** Statistics of the 6 difference problem instances giving: the kinds water usages: unimodal (only using tanks that have a single peak in usage), bimodal (only using tanks that have bimodal water usage) and complete (using all types of tanks); number of tanks  $|S|$ ; number of periods  $n$ ; and the peak capacity of each tank  $C$ .

Inst.	Types of usage	$ S $	$n$	$C$
<b>I1</b>	unimodal	1250	24	500
<b>I2</b>	unimodal	1250	48	500
<b>I3</b>	bimodal	1250	24	500
<b>I4</b>	bimodal	1250	48	500
<b>I5</b>	complete	2500	24	500
<b>I6</b>	complete	2500	48	500

## 4.2 Alternative approaches to controlling outflow

Because we only have a prediction of the future, the decisions made by the optimisation models may not be implementable with the actual inflows. Thus we implement our decisions as a “policy” for the tank to follow. If  $X_{s,t} = 1$  then tank  $s$  will empty *only if* there is sufficient volume in the tank ( $\geq m$ ), and if  $X_{s,t} = 0$  then the tank will *still empty* if the level would reach capacity  $c_s$ . This means that the decisions always lead to operation of the tank within specification. We denote this approach as HS (historical sampling).

The current operational approach and baseline is that a tank  $s$  only empties when it reaches capacity. We can understand this as a set of decisions where  $X_{s,t} = 0, \forall s \in S, t \in T$ , since emptying only happens when capacity is reached. We denote this policy  $[0, 0]$ .

An alternate baseline strategy is to set  $X_{s,t} = 1, \forall s \in S, t \in T$ , this guarantees to empty each tank as soon as it has more than the minimal capacity. While unattractive in practice, since it induces significant wear and tear on the pumps, this may reduce peak outflows. We denote this policy  $[1, 1]$ .

We also consider a random policy by counting the average proportion of pump out periods  $prop_s$  for each site  $s$  using the baseline  $[0, 0]$  policy. We draw a random number in  $[0..1)$  for each site, and try to pump out if it is below  $prop_s$ . The random policy is defined as  $X_{s,t} = 1 \rightarrow RANDOM([0..1)) \leq prop_s, \forall s \in S, t \in T$ .

Only using a single historical scenario is not robust, although since we are sampling for many tanks, certainly some of the variance of the problem is considered. We can make more robust plans by sampling for each tank  $s$  instead  $k$  scenarios, and computing the plan that leads to the minimum average maximum outflow across the scenarios (the deterministic equivalent). But the models are already slow, and this would substantially increase solve time. Instead we construct an artificial worst case scenario  $w$ , by in each time period defining the  $i_{s,t}^w = \max\{i_{s,t}^i \mid i \in 1..k\}$ , that is, the maximum inflow over all the  $k$  scenarios. This has the same size, and hence solving difficulty, as a single scenario. We denote this approach as WHS (worst case historical sampling) where we use  $k = 7$ .

## 4.3 Results

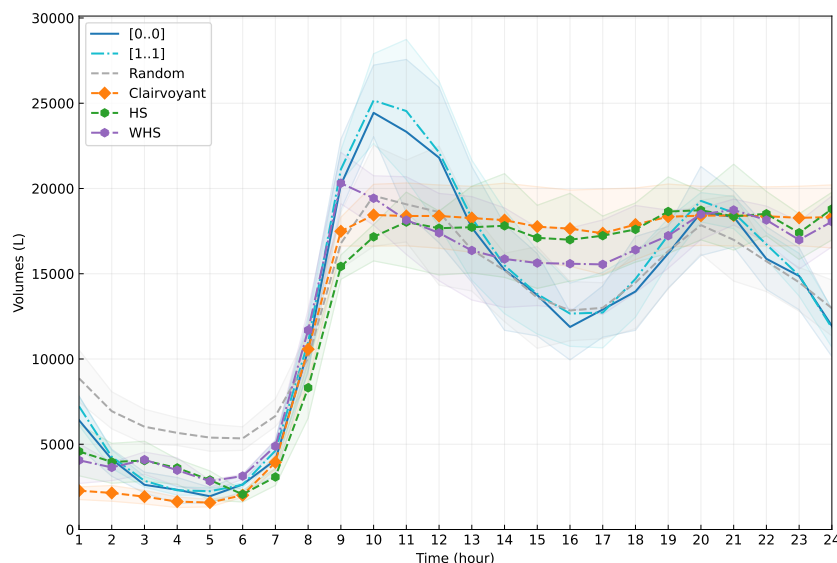
The direct formulation while asymptotically smaller,  $O(|S||T|)$ , is challenging to solve for the size of problem we tackle. On instances with 2500 sites it struggles to find solutions quickly. The packing formulation which is ostensibly  $O(|S||T|^2)$  but since  $latest(s, t) - t$  is either small, or there are many time periods where  $i_{s,t} = 0$  which do not require any pumped out decision variables, the actual size grows as  $O(|S||T|)$ , and the constraints are much simpler and hence faster to solve. For the remainder of the paper, we only report results for the packing formulation.

■ **Table 3** (Maximum mean outflow/standard deviation of outflow) for each problem instance and method across 30 scenarios.

	[0,0]	[1,1]	random	clairvoyant	HS	WHS
<b>I1</b>	(13,793/223)	(14,444/170)	(10,858/204)	(8,596/169)	<b>(8,998/170)</b>	(9,514/173)
<b>I2</b>	(13,793/223)	(14,444/170)	(11,408/209)	(8,110/206)	(8,702/205)	<b>(8,592/202)</b>
<b>I3</b>	(12,176/254)	(12,425/217)	(11,156/236)	(10,859/212)	(11,387/221)	<b>(10,877/213)</b>
<b>I4</b>	(12,176/254)	(12,425/217)	(11,042/239)	(9,717/272)	<b>(10,382/241)</b>	(10,865/237)
<b>I5</b>	(24,438/303)	(25,165/273)	(19,578/284)	(18,446/261)	<b>(18,793/276)</b>	(20,309/270)
<b>I6</b>	(24,438/303)	(25,165/273)	(19,225/285)	(17,035/281)	<b>(17,314/290)</b>	(17,368/287)

Our proposed methods are compared against the current operational strategy [0,0] and an alternate base line [1,1]. In order to see how close to optimal we get we also compare against the *clairvoyant* approach which is running the optimisation model with the actual inflow data for the tested time period. This computes the minimal maximum outflow possible.

Figure 3 shows the behavior of the models running on instance I5. The clairvoyant approach illustrates that there is a significant reduction in peak outflow available if we make wise emptying decisions. The historical sampling approach actually gets quite close to the best solution on average but it is clear that the variance is large, often well over the clairvoyant solution. The worst case approach pays some penalty, it is unable to reduce the peak flow as well, but still is not that far from the clairvoyant solution, and its standard deviation is much smaller.



■ **Figure 3** Mean total outflows in each hour of the day for different approaches applied to instance I5. The shaded regions show the 25% - 75% confidence interval, across the 30 scenarios.

Table 3 gives the summary results across the 6 instances. First note that the current baseline [0,0] is much better at reducing mean peak outflow then the alternative [1,1] of always pumping out when possible, but the standard deviation of the second method is much lower. The clairvoyant method shows that there is considerable reduction in peak available compared to the current baseline. The historical sampling optimisation approach HS is able to capture much of the available reduction in peak outflow and while the standard

deviation is larger than the clairvoyant approach it is not that much larger and considerably better than the current baseline. The worse case WHS approach also beats the baseline, and for some instances can be better than HS, its main strength is that usually reduces the standard deviation compared to HS. The random policy performs well for 24 period instances comparatively to 48 period instances. The HS and WHS approaches consistently perform better at reducing the peak flow when considering larger periods. The random policy is able to use the morning buffer capability of the tanks that is overlooked by the other approach for 24 period instances. For larger periods instances, the HS and WHS approaches systematically beat the random policy. This suggests a potential performance gain for CP based approach for larger period instances.

## 5 Conclusion

In this work, we introduced a novel practical problem from the water industry, controlling a pressure sewer system to reduce peak outflow. We proposed a direct MIP formulation and a packing formulation to solve practical scenarios. We provide an extensive experimental study on challenging and realistic instances of considerable size. The results show an optimisation model can significantly reduce the peak outflow of the system compared to the current operational approach.

So far we have only considered the most simple prediction approach, we plan to investigate forecasting models such as LSTM and LightGBM, to see whether they can produce realistic future flows. Ideally we would also extend the forecast to take into account parameters that affect the likely inflows, such as day of the week, season, and weather (the PSS we study is in a holiday zone, so inflow patterns change significantly on weekends, during summer, and when the weather is hot). It would be interesting to investigate Predict+Optimise approaches [6, 7] applied to the problem, but seeing that the predictions are for individual tanks and the objective results from considering all tanks simultaneously this appears challenging. While the simple optimisation approach we use here works we also plan to investigate decomposition approaches such as Benders or column generation, since the tanks are only weakly coupled by the objective. As future work, we plan to take into account more of the real features of the network such as the distance of tanks from the sewer treatment plant, the full topology of the network, and the inclusion of intermediate pump stations.

---

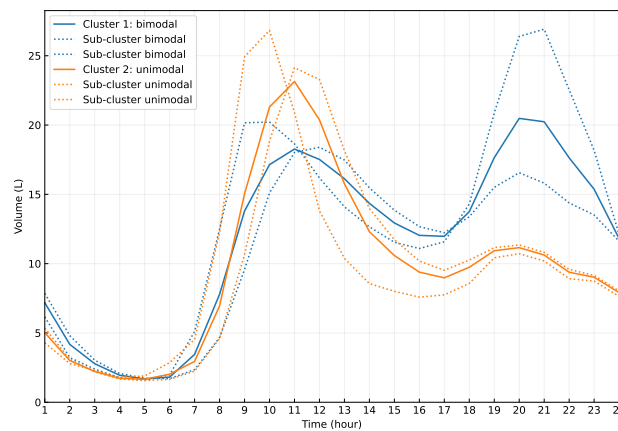
## References

- 1 Russell Bent and Pascal Van Hentenryck. Regrets only! Online stochastic optimization under time constraints. In *Proceedings of the 19th National Conference on Artificial Intelligence, AAAI'04*, pages 501–506. AAAI Press, 2004.
- 2 Russell Bent and Pascal Van Hentenryck. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52:977–987, 2004. doi: 10.1287/opre.1040.0124.
- 3 Russell Bent and Pascal Van Hentenryck. The value of consensus in online stochastic scheduling. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling, ICAPS 2004*, pages 219–226, 2004.
- 4 Russell Bent and Pascal Van Hentenryck. Online stochastic optimization without distributions. In *ICAPS 2005 - Proceedings of the 15th International Conference on Automated Planning and Scheduling*, pages 171–180, 2005.
- 5 Hyeong Soo Chang, Robert Givan, and Edwin K. P. Chong. On-line scheduling via sampling. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, AIPS'00*, pages 62–71. AAAI Press, 2000.



- 6 Adam Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022.
- 7 Jayanta Mandi, Emir Demirovic, Peter Stuckey, and Tias Guns. Smart predict-and-optimize for hard combinatorial optimization problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:1603–1610, 2020. doi:10.1609/aaai.v34i02.5521.
- 8 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74970-7\_38.

**A Clusters**



■ **Figure 4** Inflows (averaged) for the two types of flow: unimodal and bimodal. Cluster 1 shows a diurnal water usage (morning and evening peak). Cluster 2 shows a unique morning peak. Each consists of 2 sub clusters where the peaks are time shifted.

# Constraint Programming Models for Depth-Optimal Qubit Assignment and SWAP-Based Routing

Kyle E. C. Booth ✉

Amazon Quantum Solutions Lab, Seattle, WA, USA

---

## Abstract

Due to the limited connectivity of gate model quantum devices, logical quantum circuits must be compiled to target hardware before they can be executed. Often, this process involves the insertion of SWAP gates into the logical circuit, usually increasing the depth of the circuit, achieved by solving a so-called qubit assignment and routing problem. Recently, a number of integer linear programming (ILP) models have been proposed for solving the qubit assignment and routing problem to proven optimality. These models encode the objective function and constraints of the problem, and leverage the use of automated solver technology to find hardware-compliant quantum circuits. In this work, we propose constraint programming (CP) models for this problem and compare their performance against ILP for circuit depth minimization for both linear and two-dimensional grid lattice device topologies on a set of randomly generated instances. Our empirical analysis indicates that the proposed CP approaches outperform the ILP models both in terms of solution quality and runtime.

**2012 ACM Subject Classification** Computing methodologies → Planning and scheduling; Theory of computation → Constraint and logic programming

**Keywords and phrases** Qubit routing, quantum computing, constraint programming, combinatorial optimization

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.43

**Category** Short Paper

## 1 Introduction

The quantum circuit model of computation specifies quantum algorithms as sequences of logical quantum gates [12]. Quantum circuit compilation is the process of compiling a logical quantum circuit to a target quantum device such that the compiled circuit adheres to device-specific connectivity constraints. Commonly, this process involves the insertion of SWAP gates into the original circuit enabling qubits to move to neighboring locations on the hardware. Determining the initial qubit allocation as well as when and where these gates should be inserted has been studied under the name *qubit routing* [6].

Figure 1 provides an illustration of an input logical quantum circuit, a quantum device topology (represented as an undirected graph that specifies device connectivity), and a valid compiled circuit that inserted a single SWAP gate. Due to the effects of quantum decoherence, particularly in superconducting devices, it is often beneficial to solve the qubit assignment and routing problem while minimizing circuit depth (where the depth is the number of layers of parallelized gates). For this reason, additional gates should be used sparingly and with high parallelization, when possible.

A variety of techniques have been proposed for solving this problem in the literature, including both exact and heuristic approaches. Exact methods typically involve the use of search-based solvers leveraging smart inference techniques that, given enough time, will find and prove the optimal solution [4, 3, 10, 9, 14]. Alternatively, heuristic methods (which have,



© Kyle E. C. Booth;

licensed under Creative Commons License CC-BY 4.0

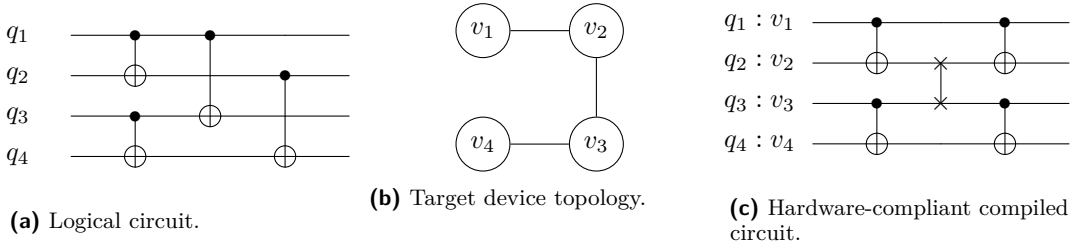
29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 43; pp. 43:1–43:10

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Qubit assignment and routing problem example specifying the input logical circuit (a), target device topology (b) and a compiled circuit (c). The input circuit has a depth of 2 (gates  $(q_1, q_3)$  and  $(q_2, q_4)$  can be executed in parallel), while the compiled circuit has a depth of 3.

until recently, been the focus of previous work) sacrifice completeness in favor of rapidly producing high quality circuits [8, 15, 6]; these methods tend to scale more effectively to larger problem instances as well.

In this short paper, we focus on the development of exact model-based approaches that outperform those from the literature. Specifically, we introduce and report initial results for two new CP models for the qubit assignment and routing problem. Our models have a relatively simple implementation, and leverage constraints supported by a variety of CP solvers. The first model is designed for linear array quantum device topologies, while the second can be used to solve problems involving more general architectures. We conduct an empirical analysis of the ILP models proposed by Boccia et al. [3] and Nannicini et al. [10] using circuit depth minimization as our objective function. We demonstrate, through empirical evaluation using different solvers, that our CP models outperform these existing ILP approaches in terms of solution quality and runtime for depth minimization.

The remainder of this paper is organized as follows. In Section 2 we define the specific variant of the qubit assignment and routing problem that we consider in this paper. In Section 3 we present our new CP models for the studied problem. In Section 4 we conduct an empirical assessment of the presented models on both linear array and 2D lattice architectures of varying size. Finally, in Section 5 we provide concluding remarks.

## 2 Preliminaries and definitions

We follow previously used notation with minor alterations to ease the presentation of the problem definition [10]. The input to the qubit assignment and routing problem consists of: i) a sequence of quantum gate groups (e.g., Figure 1a), and a hardware graph (e.g., Figure 1b). The hardware graph,  $H = (V, E)$ , consists of nodes  $i \in V$ , where each node represents a physical qubit on the quantum computer. The graph has edges  $e \in E$ , where each edge  $e = \{i, j\}$  dictates pairs of physical qubits that can execute two-qubit gates (i.e., qubits that are neighboring each other on the architecture). Often, as in previous work [10], it is useful to define a directed graph  $A$  such that each undirected edge in  $E$  corresponds to two directed edges in  $A$ ,  $(i, j)$  and  $(j, i)$ . In this paper we study both linear and general hardware graphs (e.g., lattices).

The sequence of quantum gate groups is represented by  $G = (G^1, G^2, \dots, G^{|L|})$  where  $L$  is the set of layers in the original circuit and  $G^\ell$  indicates the set of gates that must be executed in the  $\ell^{th}$  layer. In this paper, we consider two-qubit quantum gates,  $g = \{p, q\}$ , involving a pair of logical qubits  $p, q \in Q$  such that  $G^\ell = \{g_1^\ell, g_2^\ell, \dots\}$ . All of the two-qubit

gates executed in a given layer involve a set of non-overlapping pairs of qubits. This way, the gates in a layer are parallelized. To account for inserted SWAP gates, we augment the original circuit with a number of “auxiliary layers” between each of the logical circuit layers. We denote the total set of layers as  $L'$ , and use  $L_{SWAP}$  to specify the auxiliary layers only.

Qubit *assignment* involves producing an initial mapping of logical qubits,  $Q$ , to physical qubits,  $V$ , on the quantum hardware. For simplicity, we refer to logical qubits and physical qubits as qubits and nodes from now on, respectively. We also assume that  $|Q| = |V|$ .<sup>1</sup>

Qubit *routing* is the process of moving qubits around the hardware architecture such that the gates at each layer can be executed in a way that satisfies the connectivity of the device. In this work, we consider routing accomplished via the use of SWAP gates (although we note there are other routing techniques [2]), two-qubit gates which exchange the position of two neighboring qubits. These SWAP gates are inserted into the auxiliary layers between the layers of the idealized quantum circuit and (often) increase its overall depth.

The objective of the qubit assignment and routing problem studied in this paper is to: i) assign qubits to nodes on the hardware, and ii) route the positions of the qubits such that the gates at each layer can be executed while satisfying the connectivity of the device. We seek to minimize the number of auxiliary layers utilized, effectively resulting in circuits with lower depth. This problem is known to be NP-Complete [5].

Following previous work, our assumptions on the formulation of the problem include:

- Qubits involved in a two-qubit gate can also swap positions. This is also called “merging” SWAPs with adjacent two qubit gates [7].
- Two-qubit gates are unidirectional (i.e., the cost of executing a two-qubit gate in the forward configuration is the same as in the reverse configuration).
- We restrict our attention to circuits involving two-qubit gates, since single qubit gates can be merged with two-qubit gates and thus do not need to be considered.

### 3 Constraint programming models

For each of our proposed models, the main decision variable is  $x_{q\ell} \in \{1, \dots, |V|\}$  which represents the node location of logical qubit  $q \in Q$  at layer  $\ell \in L'$ . Since ILP is less flexible than CP (e.g., due to linearity restrictions) the ILP models from the literature [10, 3, 9] must introduce additional variables to properly model the problem. In our models, we exploit the structure of the problem such that it is sufficient to constrain the values of  $x_{q\ell}$  from one layer to the next and avoid additional variable overhead.

#### 3.1 Linear array architectures

The first CP model we present is applicable to linear array architectures (such as the one presented in Figure 1b). The linear array is represented as a graph with nodes  $V = \{1, 2, \dots, |V|\}$  and edges  $E = \{(i, i + 1) : i \in V \setminus |V|\}$ . This model is motivated by previous work in ILP [9], and employs absolute value constraints to ensure that qubit movement between layers is valid, and to ensure that qubits involved in a gate are at adjacent locations on the device.<sup>2</sup>

<sup>1</sup> Note that, in the case where  $|Q| < |V|$ , we can just introduce and route auxiliary qubits.

<sup>2</sup> We note that the work in [9] does not minimize circuit depth, but rather the number of SWAPs inserted, and so we do not compare to it in this paper (non-trivial changes to the model must be made in order to express a depth minimization objective).

#### 43:4 CP Models for Depth-Optimal Qubit Assignment and Routing

The first constraint in our model ensures that, at each layer, each logical qubit is located at one node on the architecture, and all of the locations of the logical qubits are different. We use the `alldifferent` global constraint [13] to accomplish this:

$$\text{alldifferent}(\{x_{1\ell}, \dots, x_{|Q|\ell}\}), \forall \ell \in L' \quad (1)$$

The next set of constraints ensure that the gates specified in each layer are executed while adhering to the connectivity constraints of the target hardware. These constraints are expressed as follows:

$$|x_{p\ell} - x_{q\ell}| = 1, \forall \ell \in L, g = (p, q) \in G^\ell \quad (2)$$

Intuitively, these constraints specify that the locations of logical qubits  $(p, q)$  involved in two-qubit gate  $g$  must be neighboring. Note that, since this model is for a linear architecture (as defined above), this constraint is enough to ensure the gates in a layer can be executed; more complex architectures require more sophisticated modeling (as described in the next section).

A similar set of constraints is used to constrain the movement of qubits from one layer to the next. These are expressed as follows:

$$|x_{q\ell} - x_{q\ell+1}| \leq 1, \forall \ell \in \{1, \dots, |L'| - 1\}, q \in Q \quad (3)$$

The above constraints dictate that, from one layer to the next, a given qubit cannot move more than one location away from its current location (preventing, for example, two simultaneous SWAP gates involving the same qubit). Additional constraints must be added to ensure that pairs of qubits, where qubit is involved in a gate at a given layer and the other is not (recall that the qubit pairs involved in a gate are permitted to SWAP), cannot SWAP from that layer to the next. This constraint is expressed as follows:

$$|x_{p\ell} - x_{q\ell+1}| \leq 1, \forall \ell \in L, g = (p, q) \in G^\ell \quad (4)$$

Constraint (4) ensures that the qubit pair involved in a gate are still neighboring at the next layer. Note that this permits the qubit pair to exchange positions, but it does not allow either qubit to swap with another qubit not involved in the gate.

Finally, we encode the objective function of the optimization which is to minimize the number of auxiliary layers added to the circuit to support SWAP insertions; this is effectively the same as circuit depth, and follows previous work [10]. To express this objective, we introduce a binary decision variable  $z_\ell \in \{0, 1\}$  for each layer,  $\ell \in L_{SWAP}$ . We constrain the variable such that it takes on a value of 1 whenever a SWAP gate is inserted into an auxiliary layer:

$$z_\ell \geq |x_{q\ell} - x_{q\ell+1}|, \forall q \in Q, \ell \in L_{SWAP}. \quad (5)$$

Constraint (5) tracks each time a qubit changes locations from one layer to the next, and the location change is not due to a (non-SWAP) gate operation. Finally, the objective function is expressed as:

$$\min \sum_{\ell \in L_{SWAP}} z_\ell \quad (6)$$

Note that if all qubit routing operations can be achieved by merged SWAPs, the optimal solution will be zero as no auxiliary SWAP gates need to be introduced.

The linear architecture CP model has  $O(|Q||L'|)$  variables, each with a domain of  $O(|V|)$ , and  $O(|Q||L'|)$  constraints. While the model is designed according to the assumptions stated in Section 2, it can be altered fairly easily to accommodate problem variations. For example, if qubits involved in a two-qubit gate *cannot* also SWAP places, we could simply constrain the locations of the qubits involved in the gate as necessary; though this would undoubtedly increase the depth of the produced circuits.

### 3.2 General architectures

When the problem is extended to more general architectures, the linear architecture model is no longer valid. This is because in the linear model, there is a clear mapping between the locations of the logical qubits and the chip connectivity; any qubits whose positions are within one of each other could have a gate applied between them. In the more general case, however, this is no longer true.<sup>3</sup>

Our model for general architectures is similar to that for linear array architectures, however, instead of using absolute value constraints we use **table** constraints. The **table** constraint specifies the list of tuples (solutions) to which a vector of variables can be fixed. For example, the constraint **table** $((y_1, y_2), \{(1, 2), (2, 3)\})$  specifies that, in a solution, the variables  $y_1$  and  $y_2$  can be assigned values  $(1, 2)$  or  $(2, 3)$ , respectively.

For our model, we utilize the **table** constraint to conveniently encode the connectivity of the hardware device, using  $A$  as the list of tuples representing the locations of neighboring pairs of qubits. Our model for general architectures is given as follows:

$$\min \sum_{\ell \in L_{SWAP}} z_\ell \quad (7)$$

$$\text{alldifferent}(\{x_{1\ell}, \dots, x_{|Q|\ell}\}) \quad \forall \ell \in L' \quad (8)$$

$$\text{table}((x_{p\ell}, x_{q\ell}), A) \quad \forall \ell \in L, g = (p, q) \in G^\ell \quad (9)$$

$$\text{table}((x_{q\ell}, x_{q\ell+1}), A \cup \{(i, i) : i \in V\}) \quad \forall q \in Q, \forall \ell \in \{1, \dots, |L'| - 1\} \quad (10)$$

$$x_{q\ell} = x_{q\ell+1} \vee x_{q\ell} = x_{p\ell+1} \quad \forall \ell \in L, g = (p, q) \in G^\ell \quad (11)$$

$$z_\ell |Q| \geq |x_{q\ell} - x_{q\ell+1}| \quad \forall \ell \in L_{SWAP}, q \in Q \quad (12)$$

The first constraint in the general architecture model, Constraint (8), is the same as Constraint (1) in the linear array model, leveraging the **alldifferent** global constraint.

Constraint (9) uses the **table** constraint to enforce that, at each gate layer, the qubits involved in each gate are at neighboring locations on the architecture (i.e., at locations represented by one of the tuples in arc set  $A$ ).

Constraint (10) uses the **table** constraint to dictate the permissible movement of qubits from one layer to the next. Specifically, this constraint requires a qubit to stay in the same location (represented by the value  $(i, i)$ ), or move to a neighboring location on the hardware. Constraint (11) adds some additional restriction regarding the movement of qubits involved in a two-qubit gate, similar to Constraint (4) in the linear array model. Finally, Constraint (12) links the objective function to the main variables, acting as a flag each time a qubit changes positions from one layer to the next (not including original circuit layers).

As with the linear array model, the general architecture model has  $O(|Q||L'|)$  variables and constraints.

<sup>3</sup> In a simple square topology with four nodes,  $v_1$  and  $v_4$  can be neighboring, but the absolute value of the difference of their node labels is not equal to one.

## 4 Experiments

We present an empirical analysis of our models on a benchmark set of randomly generated instances. Following previous work [10] we generate “square” circuits (similar to those used for quantum volume testing) where  $|Q| = |L|$  (e.g.,  $4 \times 4$  indicates a circuit with four qubits and four layers). For a given problem size, we generate a random permutation of the qubits  $\{1, 2, \dots, |Q|\}$  for each  $\ell \in L$ . Based on this permutation, we group neighboring pairs of qubits into the  $\lfloor \frac{|Q|}{2} \rfloor$  two-qubit gates for that layer (e.g., permutation 3-1-2-4 would yield gates (3, 1) and (2, 4)). By increasing the size of  $Q$  and  $L$ , we test the scalability of our models (e.g., a  $10 \times 10$  instance will involve an initial circuit with 50 two-qubit gates). For each problem size, we generate ten problem instances.

The target devices used for experimentation include a linear array and two-dimensional lattice grids of increasing size. We generate the devices such that the number of hardware vertices,  $|V|$ , is equal to the number of logical qubits,  $|Q|$ .<sup>4</sup> The linear array device topologies are straightforward. The lattice grids are generated for  $|Q| \in \{4, 6, 8, 9, 10\}$ , where  $Q \in \{4, 9\}$  corresponds to square lattices, and the remainder are rectangular lattices.

The CP models are implemented with the CP-SAT solver in OR-Tools (v9.3) [11] using the Python interface. We include the symmetry breaking constraints used in previous ILP models [10]. The absolute value expressions in the CP models were implemented using auxiliary variables and the `AddAbsEquality` constraint, while the `table` constraint was implemented using the `AddAllowedAssignments` constraint. Finally, in the general architecture CP model, the disjunction in Constraint (11) was implemented using auxiliary boolean variables and the `OnlyEnforceIf` enforcement literal. Following previous work [10], we set the number of auxiliary layers between each original layer to four.

To thoroughly assess the previously proposed ILP models, we implement them in SCIP [1], Xpress (v8.13.5), and using the CP-SAT solver in OR-Tools.<sup>5</sup> For the SCIP experiments, we use the OR-Tools modeling interface and select SCIP as the backend solver. All experiments are run with default search and inference settings on a machine with a 2.6 GHz 6-Core Intel i7 processor and 16GB of RAM. Additionally, some modifications to the model presented by Boccia et al. [3] are made to enable fair comparison. Specifically, auxiliary variables and constraints were added to permit the merged SWAP functionality.

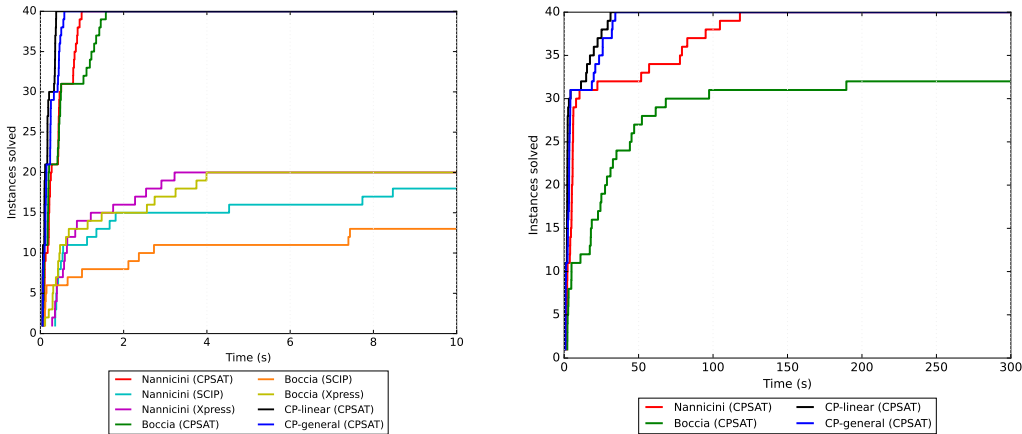
### 4.1 Linear array architectures

Our first set of experiments involve linear array architectures. For these experiments, both of the CP models we propose can be used to solve the problem. The results are visualized in Figure 2. Our proposed models are denoted “CP-linear” (for the linear array model) and “CP-general” (for the general model). We split the instances into two classes: the first involves randomly generated square circuits of size  $4 \times 4$ ,  $5 \times 5$ ,  $6 \times 6$ , and  $7 \times 7$  while the second involves square circuits of sizes  $8 \times 8$ ,  $9 \times 9$ ,  $10 \times 10$ , and  $11 \times 11$ .

In Figure 2a, we illustrate the performance of all of the implementations on Class 1 instances with a solver time limit set to 10 seconds. The figure details the number of instances solved to proven optimality vs. runtime. We can see that all of the methods implemented with the CP-SAT solver in OR-Tools are able to quickly find and prove the optimal solution to all 30 problems in less than two seconds. Conversely, the ILP models

<sup>4</sup> Our approaches are not restricted to this case; auxiliary qubits can be used to handle the situation where  $|V| > |Q|$ .

<sup>5</sup> Since the ILP models do not contain continuous variables, this is straightforward.



(a) Class 1 instances ( $|Q| \in \{4, 5, 6, 7\}$ ).

(b) Class 2 instances ( $|Q| \in \{8, 9, 10, 11\}$ ).

■ **Figure 2** Empirical results: CP models against ILP models from the literature for linear array device topologies. Number of instances solved to proven optimality over time. Time limit of 10 seconds (Class 1) and five minutes (Class 2).

implemented in SCIP and Xpress are much slower at finding and proving optimal solutions. For these approaches, the SCIP implementations seem to perform the worst, and Xpress provides a modest improvement. The experiments indicate that ILP (and ILP-based solvers) may not be the best candidate approach for this problem; this is in-line with previously reported results that showed ILP (implemented with an ILP solver) struggled to solve square circuit instances to proven optimality in short runtimes beyond six qubits [10]. As such, we elect to only investigate the models solved with OR-Tools CP-SAT for larger problem instances.

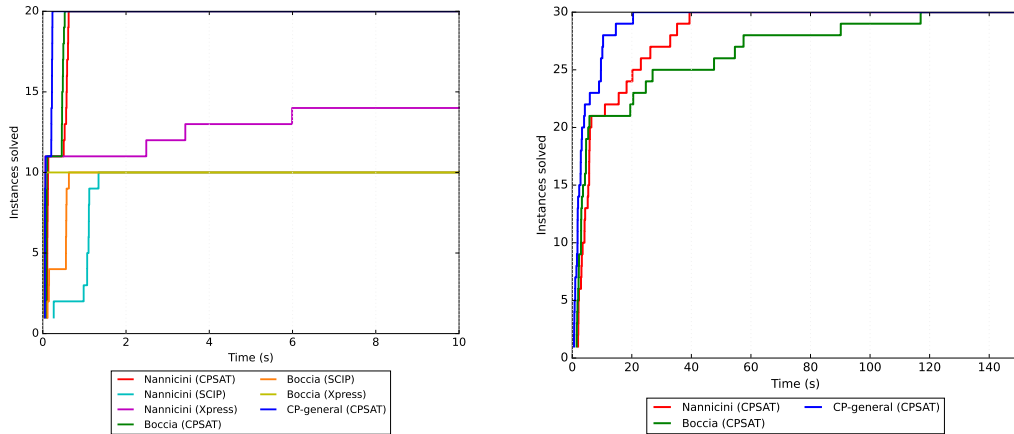
Figure 2b illustrates the performance of the CP-based implementations on medium-sized instances with a solver time limit of 5 minutes. From the figure, it is evident that our proposed CP models outperform the ILP models even when using the OR-Tools CP-SAT solver for all methods. Further, the linear array model (using absolute value constraints) performs slightly better than the more general model. Both models are able to find and prove optimality for all problem instances in less than 50 seconds, whereas the implementations of the ILP models require significantly more time.

Recall that, due to the insertion of SWAP gates as a result of our optimization, the number of layers in the final compiled circuits almost always increases. In Table 1 we detail the average circuit depth for each of the problem sizes, obtained by our depth-optimal CP methods. From the table, we can see that the inclusion of SWAP gates often doubles the depth of the circuit.

### 4.2 General architectures

Our second set of experiments involve 2D grid lattice topologies, and are visualized in Figure 3. We use the same solvers and settings as in the linear array experiments (recall that we cannot run the proposed linear array model for these lattice topology problems). For these tests, Class 1 instances are  $4 \times 4$  and  $6 \times 6$  (to permit grid lattice construction), while Class 2 instances are  $8 \times 8$ ,  $9 \times 9$ , and  $10 \times 10$ .





(a) Class 1 instances ( $4 \times 4, 6 \times 6$ ). (b) Class 2 instances ( $8 \times 8, 9 \times 9, 10 \times 10$ ).

**Figure 3** Empirical results: CP models against ILP models from the literature for 2D grid lattice device topologies. Number of instances solved to proven optimality over time. Time limit of 10 seconds (Class 1) and five minutes (Class 2).

In Table 2 we summarize the average depths of the optimally compiled circuits for each of the problem sizes. Comparing to the linear array results in Table 2, it is immediately apparent that the increased connectivity offered by 2D lattice topologies results in significantly shorter circuits. In the  $10 \times 10$  case, for example, only 5.4 layers were added (on average) to permit SWAP operations, versus the 17.0 layers added (on average) in the linear case.

In terms of model/solver runtime performance, from Figure 3a we see a similar trend to the linear topology case: the ILP methods implemented in SCIP and Xpress struggle to find and prove optimal solutions within the runtime limit, while the OR-Tools implementations rapidly solve these problems (in less than one second). Figure 3b illustrates the performance of the OR-Tools implementations for the larger class of instances. As visualized in the figure, our proposed CP approach is able to find provably optimal solutions to all instances significantly faster than the ILP models from the literature.

## 5 Conclusions

In this paper we propose CP models for depth-optimal qubit assignment and SWAP-based routing. Our first model is specific to linear array topologies, while our second model is applicable to more general architectures (e.g., grid lattices). We conduct a series of experiments on randomly generated circuits, and demonstrate that the CP-based approaches

**Table 1** Empirical results: CP model solution circuit depths by class, linear array device topologies.

Class 1	Compiled circuit depth (avg.)	Class 2	Compiled circuit depth (avg.)
$4 \times 4$	6.1	$8 \times 8$	18.0
$5 \times 5$	6.2	$9 \times 9$	20.9
$6 \times 6$	10.9	$10 \times 10$	27.0
$7 \times 7$	12.4	$11 \times 11$	30.7

■ **Table 2** Empirical results: CP model solution circuit depths by class, 2D grid lattice device topologies.

Class 1	Compiled circuit depth (avg.)	Class 2	Compiled circuit depth (avg.)
$4 \times 4$	4.0	$8 \times 8$	10.3
$6 \times 6$	6.5	$9 \times 9$	12.2
–	–	$10 \times 10$	15.4

provide superior performance over their ILP counterparts. Our results suggest that CP is a promising technology for producing provably depth-optimal circuits when qubit routing is accomplished via SWAP gate insertion.

### References

- 1 Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.
- 2 Aniruddha Bapat, Andrew M Childs, Alexey V Gorshkov, Samuel King, Eddie Schoute, and Hrishee Shastri. Quantum routing with fast reversals. *Quantum*, 5:533, 2021.
- 3 Maurizio Boccia, Adriano Masone, Antonio Sforza, and Claudio Sterle. SWAP minimization in nearest neighbour quantum circuits: An ILP formulation. In *Advances in Optimization and Decision Science for Society, Services and Enterprises*, pages 255–265. Springer, 2019.
- 4 Kyle E. C. Booth, Minh Do, J Beck, Eleanor Rieffel, Davide Venturelli, and Jeremy Frank. Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 366–374, 2018.
- 5 Adi Botea, Akihiro Kishimoto, and Radu Marinescu. On the complexity of quantum circuit compilation. In *Proceedings of the International Symposium on Combinatorial Search*, volume 9, pages 138–142, 2018.
- 6 Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. *arXiv preprint arXiv:1902.08091*, 2019.
- 7 Petar Jurcevic, Ali Javadi-Abhari, Lev S Bishop, Isaac Lauer, Daniela F Bogorin, Markus Brink, Lauren Capelluto, Oktay Günlük, Toshinari Itoko, Naoki Kanazawa, et al. Demonstration of quantum volume 64 on a superconducting quantum computing system. *Quantum Science and Technology*, 6(2):025020, 2021.
- 8 Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014, 2019.
- 9 Jesse Mulderij, Karen I Aardal, Irina Chiscop, and Frank Phillipson. A polynomial size model with implicit SWAP gate counting for exact qubit reordering. *arXiv preprint arXiv:2009.08748*, 2020.
- 10 Giacomo Nannicini, Lev S Bishop, Oktay Günlük, and Petar Jurcevic. Optimal qubit assignment and routing via integer programming. *ACM Transactions on Quantum Computing*, 2021.
- 11 Laurent Perron. Operations research and constraint programming at Google. In *Principles and Practice of Constraint Programming—CP 2011: 17th International Conference, CP 2011, Perugia, Italy, September 12–16, 2011. Proceedings 17*, pages 2–2. Springer, 2011.
- 12 Eleanor G Rieffel and Wolfgang H Polak. *Quantum computing: A gentle introduction*. MIT Press, 2011.
- 13 Willem-Jan Van Hoeve. The AllDifferent constraint: A survey. *arXiv preprint cs/0105015*, 2001.

## 43:10 CP Models for Depth-Optimal Qubit Assignment and Routing

- 14 Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology*, 3(2):025004, 2018.
- 15 Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2018.

# Constraint Model for the Satellite Image Mosaic Selection Problem

**Manuel Combarro Simón** ✉ 

University of Luxembourg, Luxembourg  
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg

**Pierre Talbot** ✉ 

University of Luxembourg, Luxembourg  
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg

**Grégoire Danoy** ✉ 

University of Luxembourg, Luxembourg  
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg

**Jedrzej Musial** ✉ 

Poznan University of Technology, Poland

**Mohammed Alswaitti** ✉ 

University of Luxembourg, Luxembourg  
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg

**Pascal Bouvry** ✉ 

University of Luxembourg, Luxembourg  
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg

---

## Abstract

Satellite imagery solutions are widely used to study and monitor different regions of the Earth. However, a single satellite image can cover only a limited area. In cases where a larger area of interest is studied, several images must be stitched together to create a single larger image, called a mosaic, that can cover the area. Today, with the increasing number of satellite images available for commercial use, selecting the images to build the mosaic is challenging, especially when the user wants to optimize one or more parameters, such as the total cost and the cloud coverage percentage in the mosaic. More precisely, for this problem the input is an area of interest, several satellite images intersecting the area, a list of requirements relative to the image and the mosaic, such as cloud coverage percentage, image resolution, and a list of objectives to optimize. We contribute to the constraint and mixed integer lineal programming formulation of this new problem, which we call the *satellite image mosaic selection problem*, which is a multi-objective extension of the polygon cover problem. We propose a dataset of realistic and challenging instances, where the images were captured by the satellite constellations SPOT, Pléiades and Pléiades Neo. We evaluate and compare the two proposed models and show their efficiency for large instances, up to 200 images.

**2012 ACM Subject Classification** Computing methodologies → Discrete space search

**Keywords and phrases** constraint modeling, satellite imaging, set covering, polygon covering

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.44

**Category** Short Paper

**Supplementary Material** *Software (Source Code)*: [https://github.com/mancs20/mosaic\\_image\\_combination/tree/cp2023](https://github.com/mancs20/mosaic_image_combination/tree/cp2023), archived at `swh:1:dir:c776d54cd26c5685fd8a0a8ddfc2131771884b32`

**Funding** *Manuel Combarro Simón*: This work is partially funded by the Luxembourg National Research Fund (FNR) – ASTRAL Project, ref. 17043604, and by the joint research programme UL/SnT-ILNAS on Technical Standardization for Trustworthy ICT, Aerospace, and Construction.



© Manuel Combarro Simón, Pierre Talbot, Grégoire Danoy, Jedrzej Musial, Mohammed Alswaitti, and Pascal Bouvry;  
licensed under Creative Commons License CC-BY 4.0

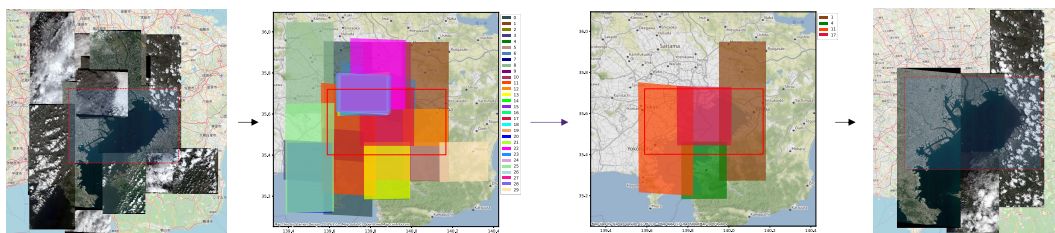
29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 44; pp. 44:1–44:15



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Optimization of the subset of images to make a mosaic of the Tokyo Bay region.

*Pierre Talbot:* This work is supported by the FNR – COMOC Project, ref. C21/IS/16101289.

*Jedrzej Musial:* This work is funded by the FNR – PolLux program under the SERENITY Project, ref. C22/IS/17395419.

## 1 Introduction

The space industry is continuously growing and is no longer an exclusive market for military and government applications. According to the most recent report of the European Union Agency for the Space Programme (EUSPA) [13], the global market for navigation systems and earth observation (EO) had revenues of around €200 billion in 2022 and is expected to reach €500 billion by 2031. As access to space has become cheaper, an increasing number of private companies have entered the space business. Due to advances in satellite design and high-resolution remote sensors, the number of satellite launches dedicated to EO in 2021 was greater than the sum of launches between 2012 and 2016 [37]. In 2020 more than 100 terabytes of satellite imagery was generated per day [28].

There are several EO-based applications that analyze a vast area of interest (AOI) that can only be covered by combining several adjacent images into a larger one, called a mosaic. Mosaics are crucial for applications such as crop classification [21, 14], environmental monitoring [30, 15], and urban development analysis [38, 32]. The mosaicking of satellite images is a complicated process that presents challenges, such as color balancing [39] and image stitching [27].

In this work, we focus on the combinatorial problem of selecting the images to create the mosaic by optimizing one or several criteria. This problem is an extended version of the NP-hard problem of finding the minimum axis-parallel rectangle cover of a rectilinear polygon without holes [11], where the axis-parallel rectangles can be seen as the satellite images, and the rectilinear polygon as the AOI. In our problem, a cover is the subset of images that can be used to generate a mosaic. In Figure 1 a particular example of this problem is shown, where the objective is to build a mosaic using the smaller number of images. There are 30 images to choose from, and the optimization algorithm finds an optimal subset of four images.

In this paper, we present a multi-objective approach for this problem that seeks to optimize four popular parameters of satellite images for mosaic generation: cloud cover, incidence angle [1], resolution and cost of the images. In general, there might not be an objective that is more important than the other, which is why we propose a multi-objective approach, instead of a linear aggregation or lexicographic ordering of the objectives.

As the number of available satellite images has increased significantly, it is becoming more challenging to select the optimal combination of images to build a mosaic. The number of images covering one place can reach hundreds. This is even more difficult if the user is interested in optimizing several parameters. Without a computational approach for this,

users have to select by hand the images they want for the cover. Providing a Pareto front from which users can choose a cover is crucial to save money and time, considering that high-resolution satellite images are expensive.

In this paper, we propose a constraint programming (CP) model and a mixed integer linear programming (MILP) model to solve the problem presented. However, directly modeling this geometric problem with constraints is challenging, as we would need to encode geometric operations such as union and intersection of polygons. Instead, we preprocess each instance by computing a discretization where the intersections of all images are first computed (Section 2.1). We obtain a set of non-overlapping polygons where each polygon is simply an integer and the geometric characteristics can be ignored. This problem is a multi-objective extension of the well-known *set covering problem* (Section 3).

A unique aspect of this problem is to minimize the cloud coverage of the mosaic, which, in contrast to other objectives, does not have the same value throughout the image. While any part of the image has the same resolution, not all parts of the images have the same amount of clouds, except for images with 0 or 100% of cloud coverage. Because of this particularity of the problem, it is possible to reduce the cloud coverage percentage in the final mosaic by choosing a specific combination of images in such a way that cloudy regions of an image are overlapped by non-cloudy regions of other images. To the best of our knowledge, there is no work taking this into account to reduce the cloud coverage in the final mosaic. We call this problem the *satellite image mosaic selection problem* (SIMS) (Section 2).

The main contribution of this paper is to introduce the SIMS problem and present a CP model, as well as MILP model that can successfully find solutions to real instances of up to 200 images (Section 4). The constraint and mixed integer programming approaches are part of a larger framework where the images are automatically retrieved from different marketplaces and the solutions found by the solver can be visualized.

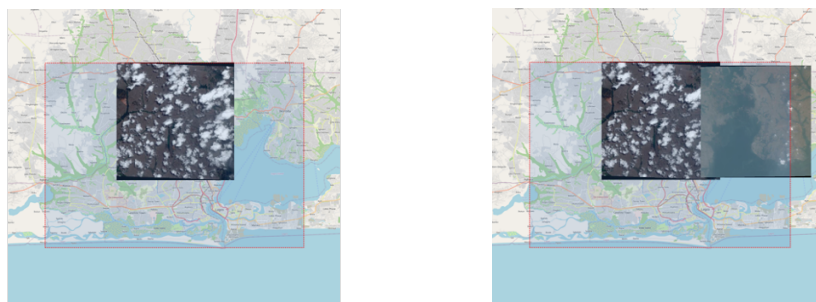
Although SIMS can be expressed as a linear problem (Section A), we choose to rely on constraint programming for two reasons: to ease the formulation of the model – in particular, it is convenient to use set variables – and because this problem aims to be extended for new requirements, which can be non-linear. The flexibility of the model is of utmost importance in this work, which is why constraint programming is our main choice.

## 2 Satellite Image Mosaic Selection Problem

The input for the SIMS problem is an area of interest (AOI) on Earth and a set of satellite images that intersect it. Each image has a cost and a list of parameters including the resolution, incidence angle and cloud coverage. The AOI is represented as a simple closed polygon without holes, and the images are represented as quadrilaterals. For both the AOI and satellite images, the corner coordinates are provided. With that information, the AOI and the images can be represented in the plane.

As clouds are not usually even distributed in the images, having images with a certain cloud coverage percentage does not guarantee that the final mosaic has less than that cloud coverage. Depending on the cloud distribution in the images, the final mosaic can have a lower or higher percentage of cloud coverage as depicted in Figure 2. This is not the case for the other objectives because they have a unique value along the image. For example, if all images have a determined resolution, the final mosaic will have the same resolution.

## 44:4 Satellite Image Mosaic Selection Problem



■ **Figure 2** A cloudy region of an image can be covered by a non-cloudy region of another image, impacting the cloud coverage percentage of the final mosaic.

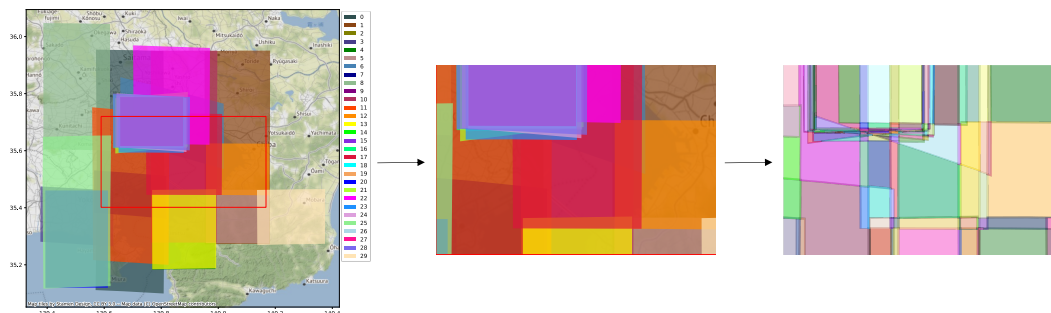
■ **Table 1** Number of intersections for the instances covering the Tokyo Bay region.

Images	Intersections
30	298
50	806
100	3278
150	8079
200	14855

### 2.1 Preprocessing of the Problem

To make the discretization, we first remove the parts of the images that are outside the AOI, and then we find all the polygons resulting from the intersection of the images, we find the polygons using the GEOS library [17]. The universe is partitioned into a set of polygon elements, and each of them is assigned to its corresponding images. In Figure 3, we show an example of this process, where we generate 254 polygons from 30 images. In Table 1, we show the number of intersections for different cardinality of the images set to cover the Tokyo Bay region.

The following step is to detect the clouds in the images and add them to the universe and to the corresponding sets. The objective of doing this is to know whether a region of the final mosaic, that is represented by one element of the universe, is free of clouds or not. We consider that a region of the AOI is free of clouds if there is at least one image in the cover in which that region does not have clouds.



■ **Figure 3** 298 polygons are obtained after preprocessing of 30 images. First the area of the images outside the AOI is removed and then the polygons resulting from the images intersections are found.



(a) Before the cloud integration the universe had 3 elements and 7 after the integration.

(b) Regions represented by element 5 and 6 are free of clouds in the final mosaic.

■ **Figure 4** Possible cases for the integration of the detected clouds to the universe and the corresponding sets. Set I and II are represented by the rectangles with red and blue borders respectively.

A cloudy region is an element in the universe that is present in all the sets that cover that region. For the set that represents the image in which the cloud occurred, we make a distinction and we say that the element is cloudy for that set. In this way, a set is composed of cloudy elements and non-cloudy elements.

In real applications, clouds can be detected using cloud detectors [36, 19, 22]. As this is a problem orthogonal to our work, we do not detect the clouds, but instead randomly allocate them in the parts of the image. For each image, we have metadata indicating the cloud coverage percentage of the image. With that information and knowing the elements that belong to the image, we randomly set one of the elements as cloudy. We repeat this operation until the cloud coverage percentage of the image is achieved.

In Figure 4a, all possible scenarios of how clouds are converted to elements of the universe are shown for the general case. To facilitate the understanding of this process, only two overlapping images are shown, but the procedure is the same when more than two images overlap. Images I and II are represented as rectangles with red and blue borders, respectively. The clouds in image I are colored red, and the ones in image II are colored blue. Initially, both sets have in common its intersection, element 3 ( $I = \{1, 3\}$  and  $II = \{2, 3\}$ ). We can see that both clouds of I are partially covered by II; in one case is because one part of the cloud is in the intersection, and in the other case is because one part of the cloud is overlapped by a cloud from II, so it can not be completely covered by a non-cloudy region of II. From the previous, we can see that three elements are created, 4, 5 and 7. Element 4 are the clouds that are only present in I, element 5 is the cloudy region of I that is not cloudy in II, i.e. covered by II, and element 7 is a cloudy region of I that is also cloudy in II. Element 6 is similar to element 5, is a cloudy area in II that is not cloudy in I. When all the clouds are detected and incorporate to the universe, the original three elements 1, 2 and 3 are modified as follows: element 1 is the non-cloudy region of I that is not overlapped by any other image, element 2 is equivalent to element 1 but for image II, and element 3 is the non-cloudy area of the intersection between I and II. Finally, the universe has seven elements, and the sets are  $I = \{1, 3, 4, 5, 6, 7\}$ ,  $II = \{2, 3, 5, 6, 7\}$ . In Figure 4b, we show a resulting mosaic after covering the clouds. Importantly, taking into account the clouds in this way increases the cardinality of the universe, but the problem itself does not change, which is why we took a simpler approach to randomly assign clouds to each element.



### 3 Constraint Model

Let  $U = \{k_1, \dots, k_n\} \subset \mathbb{N}$  be a set of  $n$  parts of the area of interest, called the universe. The set  $U$  is a polygon partition of the area of interest, i.e. two parts do not overlap and their union is exactly the area of interest. Each satellite image is represented by a collection  $P_i \subset U$  of parts. We write  $I = \{P_1, \dots, P_m\}$  the set of all  $m$  satellite images. The goal is to find a subset  $T \subset \{1, \dots, m\}$  of images that covers the area of interest. The parameters of the model are  $U$  and  $I$  while  $T$  is the main decision variable. The set covering constraint is captured by the following:

$$\bigcup_{i \in T} P_i = U \quad (1)$$

A trivial solution to this constraint is to take all the images, but we usually consider an optimization version where the cardinality of  $T$  is minimized. In our case, each image  $i \in \{1, \dots, m\}$  has a cost  $W_i \in \mathbb{N}$  that we seek to minimize:

$$\min \sum_{i \in T} W_i \quad (2)$$

Along with Equation 1, this problem is called *weighted set cover*. Depending on the user requirements, we can consider other objectives such as resolution and incidence angle. For each part  $k \in U$ , we have its area  $A_k \in \mathbb{N}$ . And for each image  $i \in \{1, \dots, m\}$ , we have its resolution  $R_i \in \mathbb{N}$  and its incidence angle  $F_i \in \mathbb{N}$ . We seek to minimize (the resolution is given in how many  $cm^2$  represents a pixel, the less the better) the best resolution obtained for each part:

$$\min \sum_{k \in U} \min\{R_i \mid i \in T, k \in P_i\} \quad (3)$$

For the incidence angle, we seek to minimize the maximal angle, although other choices would be possible such as minimizing the average.

$$\min \{\max \{F_i \mid i \in T\}\} \quad (4)$$

A more challenging aspect of this problem is to minimize the area covered by clouds. To achieve that, we consider that each part is either cloudy or not. We leave the cloud detection and the splitting of the image into cloudy and non-cloudy parts to a preprocessing step. Let  $C_i \subset P_i$  the cloudy parts of the image  $i$ . For each part  $k \in U$ , we define  $D_k := \{i \in \{1, \dots, m\} \mid k \in P_i \setminus C_i\}$  the set of all images containing a non-cloudy view of the part  $k$ . For each part  $k \in U$ , we can now define the Boolean variable  $V_k$  to be true when the part  $k$  is cloudy in the cover:

$$V_k \Leftrightarrow \bigwedge_{i \in D_k} i \notin T \quad (5)$$

We can now minimize the area covered by clouds:

$$\min \sum_{k \in U} V_k * A_k \quad (6)$$

This objective can also be turned into a constraint if the user only wants covers with a certain cloud coverage threshold.

The model introduced is actually linear, as shown in Appendix A, and can be solved by mixed integer programming solvers. We simply represent the set  $T$  by  $m$  0-1 variables  $\{x_1, \dots, x_m\}$  such that  $x_i = 1$  if we take the image and  $x_i = 0$  otherwise. We also use this representation for constraint programming solvers, because it is not possible to represent the set covering constraint otherwise – this is due to  $T$  having a non-fixed cardinality.

### 3.1 Search Strategy Based On Greedy Algorithm

A well-known greedy algorithm for the set covering problem consists in taking the images covering the most uncovered parts of the universe first [9]. We model this heuristic as a search strategy within the MiniZinc constraint model. This has the advantage of always producing a solution that is at least as good as the greedy heuristics – since it is the first solution found. To achieve that, we reuse an existing search strategy provided by MiniZinc. A second advantage is that our search strategy can be reused with any constraint solver compatible with MiniZinc. We select the variable using the `anti_first_fail` strategy – the variable with the largest domain is selected first – and we take the highest value in its domain (`indomain_max`). The trick is to model a set of variables  $\{G_1, \dots, G_m\}$  such that  $G_i \in \{0, \dots, |P_i|\}$  is equal to the number of parts covered by the image  $i$ . Actually, in any solution, we have  $G_i = |P_i|$  since the whole universe must be covered. What is interesting is the value of  $G_i$  in partial assignments during the search. The difference between the upper and lower bounds  $\max(G_i) - \min(G_i)$  is the number of parts that are currently uncovered by the partial assignment, and that can be covered by the image  $i$ . Since the anti-first-fail strategy selects the largest domain first, it effectively implements the greedy heuristics. We model  $G_i$  as follows:

$$G_i = \sum_{k \in P_i} (\bigvee_{i \in T} k \in P_i) \quad (7)$$

We note that the new variables  $G_i$  are fully defined with the parts, and therefore once the main decision variable  $T$  is assigned, the variables  $G_i$  must be assigned as well.

### 3.2 Multi-Objective Constraint Optimization Algorithm

The multi-objective constraint programming algorithm used in this work was pioneered by Gavaneli [16] and has been frequently used in constraint optimization [23, 33, 18]. The main idea is to run a *satisfaction constraint solver* iteratively and add new constraints representing the Pareto front to ensure the next solution is not dominated by any point in the current Pareto front. To illustrate this algorithm, suppose a biobjective maximization problem where  $x$  and  $y$  are the two variables to optimize. We run the constraint solver which returns a first satisfiable solution where  $x = 10$  and  $y = 5$ . At that point the Pareto front is  $\{(10, 5)\}$ . We add to the model the constraint  $x > 10 \vee y > 5$  which guarantees that the next solution will not be dominated by the current points in the Pareto front. The solver might then find the solution  $x = 2$  and  $y = 6$  which is incomparable to the previous solution and is added to the Pareto front  $\{(10, 5), (2, 6)\}$ . The constraint generated from the Pareto front is now  $(x > 10 \vee y > 5) \wedge (x > 2 \vee y > 6)$ . This process continues until the solver finds an unsatisfiable solution, in which case we are guaranteed to have found the optimal Pareto front.

## 4 Evaluation

### 4.1 Dataset Description

For each experiment, there is an AOI and a number of images that cover the AOI. The objective is to find the Pareto front, where each point in the front represents a subset of images that must cover the AOI and optimize four objectives: cost, resolution, incidence angle and cloud coverage.

To carry out this research, we developed a framework capable of retrieving image metadata from different satellite marketplaces, preprocessing it (discretization and cloud integration to the universe), calling a CP or a MILP solver, and visualizing the solutions from the Pareto front.

We selected five AOIs from around the world: Mexico City (Mexico), Rio de Janeiro (Brazil), Paris (France), Lagos (Nigeria), and Tokyo Bay (Japan). For each AOI, we obtained all available images that were captured from 01-01-2021 to 01-01-2023 by the following satellite constellations SPOT [4], Pléiades [2] and Pléiades Neo [3]. We opted for those satellite constellations as they have all the metadata used in the experiments; other satellite constellations lacked some parameters such as cloud coverage or incidence angle.

Five instances were generated for all AOIs, except Lagos. Each of these instances differs from each other by the number of images given to cover the AOIs. The number of images for the instances were 30, 50, 100, 150 and 200. For Lagos, the total number of images available for the specified date range was 145, so the number of images for the Lagos instances were 30, 50, 100 and 145.

### 4.2 Experimental Setup

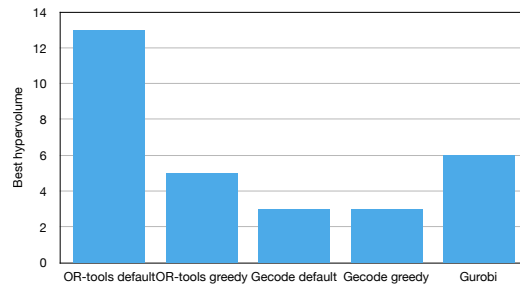
Each instance was solved using the CP model and the MILP model. We run the CP model with two solvers, OR-tools [31] and Gecode 6.3.0 [34]. For each of these solvers, we run the experiments twice; one with the default solver search strategy, and the other one with the greedy search strategy proposed in Section 3.1. The MILP model was implemented using the Gurobi solver [20]. We will refer to these five approaches as *OR-tools default*, *OR-tools greedy*, *Gecode default*, *Gecode greedy* and *Gurobi*.

For the MILP model, the algorithm used to obtain the exact Pareto front was SAUGMENCON [40] which is based on the AUGMECON [25, 26] algorithm and on the well-known  $\epsilon$ -constraint method. In the  $\epsilon$ -constraint methods, one objective is optimized, and the others are added as constraints to the model. The right-hand side of the objective constraints gradually changes from the less restrictive values of the objectives to the most restricted ones. This process continues until all combinations of values for the constraint objectives have been explored. The SAUGMENCON method introduces two acceleration mechanisms to improve the computational efficiency of the front generation.

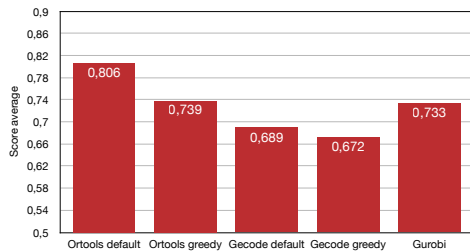
The experiments were run on an AMD Epyc ROME 7H12 processor (64 cores, 280W). All the solvers were configured to run in parallel with 8 cores and 16 threads. The running time for each experiment was 1 hour.

### 4.3 Experimental Results

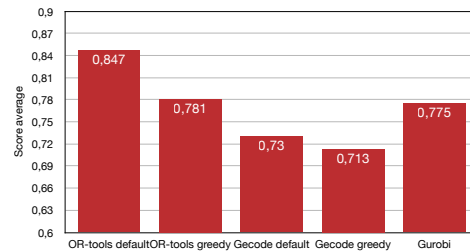
To compare the results, we used the hypervolume of the Pareto front, which is a standard metric for comparing fronts in multiobjective optimization. For each instance, we score the strategies, calculating how worst they are compared to the best. For example, a score of 1 means that the strategy has the same hypervolume value as the best one, and a score of 0.5 means that the hypervolume of the strategy is half of the best hypervolume for that instance.



■ **Figure 5** Number of times each approach had the best hypervolume.



(a) Considering the 24 instances.



(b) Considering the 21 instances where all strategies found a solution.

■ **Figure 6** Average score for each approach.

In Table 2 from Appendix C we can see the hypervolume values for each strategy for all instances.

For only one instance, the complete Pareto front was found in the running time, the strategies that found the complete front were *OR-tools default*, *OR-tools greedy* and *Gurobi*. For the rest of the instances, the whole Pareto front was not found, and the hypervolume corresponds to the partial front found during the running time. For 3 of the 4 instances with 200 images, the CP strategies could not find any point of the front; the entire running time was employed by the FlatZinc submodule of Minizinc, to flatten the model with the data file. However, Gurobi for 3 of these instances could find one point of the Pareto front. For the rest of the instances, all the strategies could find at least one point of the Pareto front, and generally the CP approaches obtained superior results compared to MILP.

As we can see in Figures 5 and 6a, for these experiments, the best approach was *OR-tools default*, being the best for 13 out of 24 instances and with a score average of 0.806. The second and third best strategies were *OR-tools greedy* and *Gurobi*, with a similar performance. The fourth and fifth places were occupied by Gecode default and greedy, being really close.

If we do not consider the 3 instances in which the CP strategies could not find a solution, the score averages change; see Figure 6b. *OR-tools default* has an average score very close to 1, and *OR-tools greedy* has a much better average score than *Gurobi*, which for these instances is the worst strategy in average. The difference in the average score between both Gecode strategies remains very close.

It is interesting to note that *Gurobi* showed excellent performance for small instances comprising 30 and 50 images. However, its performance was not as impressive for larger instances. This could be related to the way solutions are discovered and added to the front. For future research, it could be interesting to compare the hypervolume anytime behavior for different approaches used for CP and MILP to get the exact Pareto front.

## 5 Related Work

Geometric set covering problems can be divided into two categories based on the requirements of the covering shapes. In one category, the covering shapes do not have a fixed position in the plane, for example, covering a polygonal region with the minimum amount of fixed sized rectangles [24] or with a set of known rectangles that can freely move on the plane [35]. The other category is where the covering shapes have a fixed size and position, for example, covering a polygonal region with discs with a fixed size and position on the plane [10]. SIMS, belongs to the second category as the satellite images represent a fixed region on the Earth.

Considering the cloud coverage percentage in the final mosaic makes SIMS problem different from polygon cover and other geometrical cover problems, where the covering shapes only have to cover the polygon or the universe of points in the space. In this problem, the covering shapes, besides covering the polygon, should also cover certain regions (clouds) that are present in the shapes. Interestingly, this can be seen as solving two weighted set covering problems; in one, the AOI must be covered and in the other, the clouds.

The main approaches to solving geometric set covering problems are local search [6, 12, 5] and linear programming (LP) [7, 8]. In most of the papers, opposite to SIMS, the universe is a set of points instead of a region. In [10], they provide an exact algorithm for the case where the universe is a set of regions, and the covering objects are discs. The algorithm is effective when the minimum number of discs to cover the space is low.

In [29], set covering problem is tackled using constraint programming. There, the authors propose a way to prune the domain of possible solution using a lower and upper bound for the objective value. The lower bound consists of determining the minimum number of sets that can cover the space. This is equivalent to answering the following NP-complete problem: does a cover of the universe exist with  $K$  sets. They propose a new strategy to get an approximation of this lower bound and compare it against two other well-known lower-bound values: the value of the LP relaxation problem and a greedy algorithm. The proposed prune strategy is good for problems where the size of the sets is small, for bigger subsets, they recommend alternating between the LP relaxation and the greedy algorithm.

## 6 Conclusion

In this paper, we introduce a novel geometrical NP-hard problem, SIMS, inspired by the selection of satellite images for mosaic generation. CP and MILP models are provided for this problem, together with a search strategy for the CP model, based on the well-known greedy algorithm used for set covering problems. In the experiments performed, the CP solved with OR-tools got the best result, evidencing the power of this solver. Our proposed search strategy could not outperform the default search strategies, but in the case of the Gecode solver, it produced similar results. Generally, the CP model outperformed the MILP model. This could be related to the method used to generate the Pareto front. For future work, it will be interesting to compare different approaches to generate the exact Pareto front for the CP and MILP models, based on the metric anytime behavior for the hypervolume. We also plan to propose heuristics to tackle larger instances and to evaluate their performance against the proposed CP and MILP models.

---

## References

- 1 Airbus. Incidence angle. URL: <https://www.intelligence-airbusds.com/en/8719-angle-conversion>.

- 2 Airbus. Pléiades. URL: <https://www.intelligence-airbusds.com/en/8692-pleiades>.
- 3 Airbus. Pléiades Neo. URL: <https://www.airbus.com/en/products-services/space/earth-observation/earth-observation-portfolio/pleiades-neo>.
- 4 Airbus. SPOT. URL: <https://www.intelligence-airbusds.com/en/8693-spot-67>.
- 5 Pradeesha Ashok, Aniket Basu Roy, and Sathish Govindarajan. Local search strikes again: PTAS for variants of geometric covering and packing. *Journal of Combinatorial Optimization*, 39(2):618–635, June 2019. doi:10.1007/s10878-019-00432-y.
- 6 Nikhil Bansal and Kirk Pruhs. Weighted geometric set multi-cover via quasi-uniform sampling. In *Algorithms – ESA 2012*, pages 145–156. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-33090-2\_14.
- 7 Timothy M. Chan and Qizheng He. Faster approximation algorithms for geometric set cover. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 164. Schloss Dagstuhl- Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, June 2020. doi:10.4230/LIPIcs.SocG.2020.27.
- 8 Chandra Chekuri, Sarel Har-Peled, and Kent Quanrud. Fast LP-based approximations for geometric packing and covering problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1019–1038. Society for Industrial and Applied Mathematics, January 2020. doi:10.1137/1.9781611975994.62.
- 9 V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, August 1979. doi:10.1287/moor.4.3.233.
- 10 Claudio Contardo and Alain Hertz. An exact algorithm for a class of geometric set-cover problems. *Discrete Applied Mathematics*, 300:25–35, 2021. doi:10.1016/j.dam.2021.05.005.
- 11 Joseph C. Culberson and Robert A. Reckhow. Covering polygons is hard. *Annual Symposium on Foundations of Computer Science (Proceedings)*, pages 601–611, 1988. doi:10.1109/SFCS.1988.21976.
- 12 Minati De and Abhiruk Lahiri. Geometric Dominating-Set and Set-Cover via Local-Search. *Computational Geometry*, 113:102007, August 2023. doi:10.1016/j.comgeo.2023.102007.
- 13 European Union Agency for the Space Programme. EUSPA EO and GNSS Market Report. Technical Report 1, European Union Agency for the Space Programme, 2022. URL: [https://www.euspa.europa.eu/sites/default/files/uploads/euspa\\_market\\_report\\_2022.pdf](https://www.euspa.europa.eu/sites/default/files/uploads/euspa_market_report_2022.pdf).
- 14 Shilan Felegari, Alireza Sharifi, Kamran Moravej, Muhammad Amin, Ahmad Golchin, Anselme Muzirafuti, Aqil Tariq, and Na Zhao. Integration of Sentinel 1 and Sentinel 2 Satellite Images for Crop Mapping. *Applied Sciences*, 11:10104, 2021. doi:10.3390/app112110104.
- 15 Neil Flood, Fiona Watson, and Lisa Collett. Using a U-net convolutional neural network to map woody vegetation extent from high resolution satellite imagery across Queensland, Australia. *International Journal of Applied Earth Observation and Geoinformation*, 82:101897, October 2019. doi:10.1016/J.JAG.2019.101897.
- 16 Marco Gavanelli. An algorithm for multi-criteria optimization in CSPs. In *ECAI 2002: 15th European Conference on Artificial Intelligence, July 21-26, 2002, Lyon France: Including Prestigious Applications of Intelligent Systems (PAIS 2002): Proceedings*, volume 77, page 136. IOS Press, 2002.
- 17 GEOS contributors. *GEOS coordinate transformation software library*. Open Source Geospatial Foundation, 2021. URL: <https://libgeos.org/>.
- 18 Tias Guns, Peter J. Stuckey, and Guido Tack. Solution Dominance over Constraint Satisfaction Problems, 2018. arXiv:1812.09207 [cs]. URL: <http://arxiv.org/abs/1812.09207>.
- 19 Yanan Guo, Xiaoqun Cao, Bainian Liu, and Mei Gao. Cloud detection for satellite imagery using attention-based u-net convolutional neural network. *Symmetry*, 12(6):1056, June 2020. doi:10.3390/sym12061056.
- 20 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: <https://www.gurobi.com>.

- 21 Ola Hall, Sigrun Dahlin, Håkan Marstorp, Maria Francisca Archila Bustos, Ingrid Öborn, and Magnus Jirström. Classification of maize in complex smallholder farming systems using UAV imagery. *Drones*, 2(3):1–8, 2018. doi:10.3390/drones2030022.
- 22 Jacob Høxbroe Jeppesen, Rune Hylsberg Jacobsen, Fadil Inceoglu, and Thomas Skjødeberg Toftegaard. A cloud detection algorithm for satellite imagery based on deep learning. *Remote Sensing of Environment*, 229:247–259, August 2019. doi:10.1016/j.rse.2019.03.039.
- 23 Martin Lukaszewicz, Michael Glaß, Christian Haubelt, and Jürgen Teich. Solving Multi-objective Pseudo-Boolean Problems. In *Theory and Applications of Satisfiability Testing – SAT 2007*, pages 56–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-72788-0\_9.
- 24 Sina Sharif Mansouri, George Georgoulas, Thomas Gustafsson, and George Nikolakopoulos. On the covering of a polygonal region with fixed size rectangles with an application towards aerial inspection. In *2017 25th Mediterranean Conference on Control and Automation (MED)*. IEEE, July 2017. doi:10.1109/med.2017.7984284.
- 25 George Mavrotas. Effective implementation of the  $\epsilon$ -constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213(2):455–465, July 2009. doi:10.1016/j.amc.2009.03.037.
- 26 George Mavrotas and Kostas Florios. An improved version of the augmented  $\epsilon$ -constraint method (AUGMECON2) for finding the exact pareto set in multi-objective integer programming problems. *Applied Mathematics and Computation*, 219(18):9652–9669, May 2013. doi:10.1016/j.amc.2013.03.002.
- 27 V. Megha and K. K. Rajkumar. Automatic Satellite Image Stitching Based on Speeded Up Robust Feature. In *Proceedings - 2021 1st IEEE International Conference on Artificial Intelligence and Machine Vision, AIMV 2021*, pages 1–6, Gandhinagar, India, 2021. Institute of Electrical and Electronics Engineers Inc. doi:10.1109/AIMV53313.2021.9670954.
- 28 Doug Mohny. Terabytes From Space: Satellite Imaging is Filling Data Centers, 2020. URL: <https://datacenterfrontier.com/terabytes-from-space-satellite-imaging-is-filling-data-centers/>.
- 29 Sébastien Mouthuy, Yves Deville, and Grégoire Doooms. Global constraint for the set covering problem. *Journées Francophones de Programmation par Contraintes*, pages 183–192, 2007.
- 30 Konstantinos G. Nikolakopoulos, Paraskevi Lampropoulou, Elias Fakiris, Dimitris Sardelianos, and George Papatheodorou. Synergistic use of UAV and USV data and petrographic analyses for the investigation of beachrock formations: A case study from Syros Island, Aegean sea, Greece. *Minerals*, 8(11):534, November 2018. doi:10.3390/min8110534.
- 31 Laurent Perron and Vincent Furnon. Or-tools. URL: <https://developers.google.com/optimization/>.
- 32 Simone Piaggese, Laetitia Gauvin, Michele Tizzoni, Natalia Adler, Stefaan Verhulst, Andrew Young, Rihannan Price, Leo Ferres, Ciro Cattuto, and André Panisson. Predicting city poverty using satellite imagery. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, volume 2019-June, pages 90–96, 2019.
- 33 Pierre Schaus and Renaud Hartert. Multi-Objective Large Neighborhood Search. In *Principles and Practice of Constraint Programming*, volume 8124, pages 611–627. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-40627-0\_46.
- 34 Christian Schulte, Guido Tack, and Mikael Lagerkvist. *Modeling and Programming with Gecode*, 2020.
- 35 Y. G. Stoyan, T. Romanova, G. Scheithauer, and A. Krivulya. Covering a polygonal region by rectangles. *Computational Optimization and Applications 2009 48:3*, 48(3):675–695, May 2009. doi:10.1007/S10589-009-9258-1.
- 36 Lin Sun, Xu Yang, Shangfeng Jia, Chen Jia, Quan Wang, Xinyan Liu, Jing Wei, and Xueying Zhou. Satellite data cloud detection using deep learning supported by hyperspectral data. *International Journal of Remote Sensing*, 41(4):1349–1371, September 2019. doi:10.1080/01431161.2019.1667548.

- 37 Union of Concerned Scientists. UCS Satellite Database. URL: <https://www.ucsusa.org/resources/satellite-database>.
- 38 Haibo Wang, Xueshuang Gong, Bingbing Wang, Chao Deng, and Qiong Cao. Urban development analysis using built-up area maps based on multiple high-resolution satellite data. *International Journal of Applied Earth Observation and Geoinformation*, 103:102500, December 2021. doi:10.1016/J.JAG.2021.102500.
- 39 Lei Yu, Yongjun Zhang, Mingwei Sun, and Xinyu Zhu. Colour balancing of satellite imagery based on a colour reference library. *International Journal of Remote Sensing*, 37(24):5763–5785, December 2016. doi:10.1080/01431161.2016.1249306.
- 40 Weihua Zhang and Marc Reimann. A simple augmented  $\epsilon$ -constraint method for multi-objective mathematical integer programming problems. *European Journal of Operational Research*, 234(1):15–24, April 2014. doi:10.1016/j.ejor.2013.09.001.

## A Linear Programming Model

For the mixed integer linear programming, we use the same nomenclature as for the constraint model. We just add the necessary variables to linearize the model.

To linearize the cover constraint (1) it is necessary to associate each image  $P_i$  with a decision variable  $x_i$  that is equal to 1 if the image  $i$  is selected, otherwise it is 0. We rewrite the constraint as follows:

$$\sum_{i:k \in P_i} x_i \geq 1, \text{ for all } k \in U \quad (8)$$

The previous constraint guarantees that all the parts are covered by at least one image.

To linearize the cost constraint (2) it is necessary to associate each image  $P_i$  with an auxiliary variable  $w_i$  that represents the cost of the image. The linear constraint can be written as:

$$\min \sum_{P_i \in I} x_i w_i \quad (9)$$

The constraints (8) and (9) are the classical constraints used for set covering problems.

The resolution objective is a min-min problem, where the objective is to minimize the sum of the min resolution of each part. The min resolution of a part is the minimum resolution of the images that contain them and belong to a cover. We need to add an auxiliary decision variable  $r_k$  representing the best resolution of the part  $k$  and a big constant  $B$ , bigger than the maximum image resolution. Also, we need to add an auxiliary binary decision variables  $z_{k_j}$  for each image  $P_j$  that contains  $k$ . For each part  $k$ , we define  $L_k := \{i \in \{1, \dots, m\} \mid k \in P_i\}$  as the set of all images containing the part  $k$ . For each part  $k$  we can now define a constraint for the values that can take the variables  $z_{k_j}$ .

$$\sum_{k=1}^{|L_k|} z_{k_j} = |L_k| - 1 \quad (10)$$

The constraint expressed above states that only one of the  $z_{k_j}$  variables can be 0, the rest have to be 1. We define the minimum resolution of a part as  $r_k$ . With the following two constraints, we can linearize (3).

$$r_k \geq (x_j R_j + B(1 - x_i)) - 2Bz_{k_j} \text{ for all } j \in L_k \quad (11)$$

$$\min \sum_{k \in U} r_k \quad (12)$$



## 44:14 Satellite Image Mosaic Selection Problem

The first term on the right-hand side of (11) affects how the part  $k$  perceives the resolution of the images to which it belongs. If the image is in a cover, the resolution is equal to  $R_j$ . If the image is not in a cover, then the resolution is equal to the big constant  $B$ . As we minimize  $r_k$ , the lower this first term, the better. This term forced the images with lower resolution to be in a cover. The second term on the right-hand side of (11) is used to force  $r_k$  equal to the minimum value of the resolution of the images that contain the part  $k$  and are in the cover. When  $z_{k_j} = 1$  the right-hand side is negative and when 0 the value is positive, and it is the value that  $r_k$  takes.

To linearize the incidence angle objective, we need to minimize an auxiliary variable  $max_f$  that represents the maximum incidence angle of the images in the cover.

$$\min max_f \geq x_i F_i, \text{ for all } i = 1, \dots, m \quad (13)$$

To minimize the area of the clouds, we can model this as a partial set cover problem, where the universe  $C = \{1, \dots, c\}$  is formed by all the clouds, and the sets are the images that can cover the clouds. For example, if we have the following set  $P_{2_c} = \{c_1, c_2, c_5\}$  it means that image 2 can cover clouds 1, 2 and 5, i.e. parts 1, 2 and 5 are not cloudy in image 2. For each cloud  $c_i$  we have a variable  $y_i$  that is 1 if the cloud is covered or 0 otherwise and  $A_c$  indicating the area of the cloud. To maximize the covering of the cloudy areas, we will minimize the following expression:

$$\min - \sum_{c \in C} y_c A_c \quad (14)$$

Subject to the following constraint, which forces  $y_c$  to be 0 if any of the images that cover it is selected to cover the AOI.

$$\sum_{i: c \in P_{i_c}} x_i \geq y_c, \text{ for all } c \in C \quad (15)$$

### **B** MiniZinc Model

We describe the full MINIZINC constraint model implementing the mathematical model given in Section 2.

```
int: num_images;
int: universe;
int: max_cloud_area;

set of int: IMAGES = 1..num_images;
set of int: UNIVERSE = 1..universe;

array[IMAGES] of set of int: images;
array[IMAGES] of set of int: clouds;
array[IMAGES] of int: costs;
array[UNIVERSE] of int: areas;
array[IMAGES] of int: resolution;
array[IMAGES] of int: incidence_angle;

array[IMAGES] of var bool: taken;

% Which images have a universe 'u' without cloud?
% That is, uclear[u] = {i1, i2, ..} means that the images numbered i1, i2
% , ... contains 'u' without clouds.
array[UNIVERSE] of set of int: uclear = [{ i | i in IMAGES where not (u
in clouds[i]) /\ u in images[i] } | u in UNIVERSE];
```

```

% Set covering constraint.
constraint forall(u in UNIVERSE)(
  exists(i in IMAGES)(taken[i] /\ u in images[i]));

% cloudy[u] is true iff no image containing a version of 'u' without
  clouds is taken.
array[UNIVERSE] of var bool: cloudy;
array[UNIVERSE] of var int: num_clear_images;
constraint forall(u in UNIVERSE)(
  num_clear_images[u] = sum(i in uclear[u])(taken[i])
);
constraint forall(u in UNIVERSE)(cloudy[u] = (num_clear_images[u] == 0));

var int: cloudy_area = sum(u in UNIVERSE)(cloudy[u] * areas[u]);

var int: total_cost = sum(i in IMAGES)(costs[i] * taken[i]);
var int: max_resolution = sum(u in UNIVERSE)(min(i in IMAGES where u in
  images[i] /\ taken[i])(resolution[i]));
var int: max_incidence = max(i in IMAGES)(taken[i] * incidence_angle[i]);

array[1..4] of var int: objs;
constraint objs[1] = total_cost;
constraint objs[2] = cloudy_area;
constraint objs[3] = max_resolution;
constraint objs[4] = max_incidence;

```

## C Experimental results detailed

■ **Table 2** Hypervolume values for all the experiments.

Instance	OR-tools default	OR-tools greedy	Gecode default	Gecode greedy	Gurobi
lagos_nigeria_30	<b>5.46E+33</b>	5.21E+33	4.89E+32	4.91E+33	4.87E+33
mexico_city_30	4.83E+32	4.82E+33	4.62E+33	4.59E+32	<b>4.83E+33</b>
paris_30	<b>1.95E+34</b>	<b>1.95E+34</b>	1.59E+34	1.58E+34	1.95E+33
rio_de_janeiro_30	<b>5.76E+33</b>	5.65E+33	5.65E+33	5.65E+33	<b>5.76E+33</b>
tokyo_bay_30	<b>6.35E+33</b>	6.33E+33	6.05E+33	6.05E+33	6.30E+33
lagos_nigeria_50	<b>3.26E+34</b>	3.15E+34	2.19E+34	2.18E+34	2.77E+34
mexico_city_50	2.88E+33	<b>2.92E+34</b>	2.63E+34	2.57E+34	2.71E+34
paris_50	<b>9.02E+34</b>	8.85E+34	7.12E+34	7.24E+34	7.85E+34
rio_de_janeiro_50	3.94E+33	<b>3.87E+34</b>	3.05E+34	3.04E+34	3.83E+34
tokyo_bay_50	<b>3.32E+34</b>	3.22E+34	1.65E+34	1.78E+34	1.98E+34
lagos_nigeria_100	1.86E+35	<b>1.87E+35</b>	1.68E+35	1.68E+35	8.55E+34
mexico_city_100	1.98E+35	1.97E+35	<b>2.11E+35</b>	2.03E+35	1.58E+35
paris_100	<b>4.73E+35</b>	3.13E+34	4.32E+35	2.91E+35	4.32E+35
rio_de_janeiro_100	<b>4.76E+35</b>	4.14E+35	2.28E+35	1.83E+35	3.39E+35
tokyo_bay_100	1.23E+35	1.23E+35	1.89E+35	1.89E+35	<b>2.77E+35</b>
lagos_nigeria_145	2.98E+36	<b>3.78E+36</b>	2.32E+36	2.32E+36	9.30E+35
mexico_city_150	1.52E+36	1.11E+36	2.27E+36	<b>2.28E+36</b>	3.42E+35
paris_150	2.60E+36	2.83E+36	1.11E+36	1.11E+36	<b>2.87E+36</b>
rio_de_janeiro_150	<b>6.59E+35</b>	4.81E+34	4.00E+35	4.00E+35	4.61E+34
tokyo_bay_150	<b>1.16E+36</b>	8.68E+35	8.36E+35	8.36E+35	1.12E+36
mexico_city_200	2.66E+36	2.05E+35	<b>4.28E+36</b>	<b>4.28E+36</b>	1.06E+36
paris_200	0.00E+00	0.00E+00	0.00E+00	0.00E+00	<b>1.60E+36</b>
rio_de_janeiro_200	<b>0.00E+00</b>	<b>0.00E+00</b>	<b>0.00E+00</b>	<b>0.00E+00</b>	<b>0.00E+00</b>
tokyo_bay_200	0.00E+00	0.00E+00	0.00E+00	0.00E+00	<b>2.46E+35</b>



# Partitioning a Map into Homogeneous Contiguous Regions: A Branch-And-Bound Approach Using Decision Diagrams

Nicolas Golenvaux ✉ 

Institute for Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), Université catholique de Louvain, Louvain-la-Neuve, Belgium

Xavier Gillard ✉ 

Institute for Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), Université catholique de Louvain, Louvain-la-Neuve, Belgium

Siegfried Nijssen ✉ 

Institute for Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), Université catholique de Louvain, Louvain-la-Neuve, Belgium

Pierre Schaus ✉ 

Institute for Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM), Université catholique de Louvain, Louvain-la-Neuve, Belgium

---

## Abstract

Regionalization is a crucial spatial analysis technique used for partitioning a map divided into zones into  $k$  continuous areas, optimizing the similarity of zone attributes within each area. This technique has a variety of applications in fields like urban planning, environmental management, and geographic information systems. The REDCAP algorithm is a well-known approach for addressing the regionalization problem. It consists of two main steps: first, it generates a spatially contiguous tree (SCT) representing the neighborhood structure of the set of spatial objects using a contiguity-constrained hierarchical clustering method. Second, it greedily removes  $k - 1$  edges from the SCT to create  $k$  regions. While this approach has proven to be effective, it may not always produce the most optimal solutions. We propose an alternative method for the second step, an exact dynamic programming (DP) formulation for the  $k-1$  edges removal problem. This DP is solved using a multi-valued decision diagram (MDD)-based branch and bound solver leading to a more optimal solution. We compared our proposed method with the REDCAP state-of-the-art technique on real data and synthetic ones, using different instances of the regionalization problem and different supervised and unsupervised metrics. Our results indicate that our approach provides higher quality partitions than those produced by REDCAP at acceptable computational costs. This suggests that our method could be a viable alternative for addressing the regionalization problem in various applications.

**2012 ACM Subject Classification** Computing methodologies → Discrete space search

**Keywords and phrases** Regionalization, Redcap, Skater, Multivalued Decision Diagrams

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.45

**Category** Short Paper

## 1 Introduction

Spatial analysis plays a crucial role in comprehending and managing intricate spatial relationships [13]. A fundamental challenge in spatial analysis involves determining homogeneous regions based on similarity computed from shared attributes.

Given a geographical map divided into zones that partition the space, each zone is associated with a set of attributes (e.g., population density, land use, socio-economic factors, etc.) as represented on Figure 1a where the colors represent the attributes. The regionalization



© Nicolas Golenvaux, Xavier Gillard, Siegfried Nijssen, and Pierre Schaus; licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 45; pp. 45:1–45:10

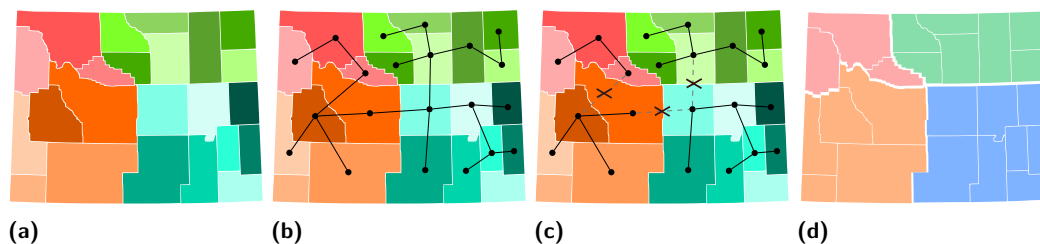
Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problem studied in this paper, involves grouping these zones into  $k$  contiguous areas (also referred to as regions), optimizing the similarity of attributes within each area. The contiguity constraint requires that each grouping forms a single, connected area without any isolated parts or exclaves.

Solving the regionalization problem can be computationally complex, even for a moderate number of zones and areas, as it often entails searching for an optimal solution within a large combinatorial search space. The state-of-the-art method called REDCAP [12] involves solving the problem in a two-step approach as illustrated on Figure 1. In the first step, a spatially contiguous (spanning) tree is created using a hierarchical clustering strategy. In the second step,  $k - 1$  edges are greedily deleted from the tree. The final contiguous areas are defined by the  $k$  remaining connected components of the tree, ensuring both attribute similarity and contiguity.



**Figure 1** Steps for solving a regionalization problem to cluster four areas: (a) Input of the problem where colors represent the attributes of each zone. (b) Create a spatially contiguous tree that connects all the zones. (c) Remove three edges from the tree. (d) The output areas are then formed.

## 2 Related Work

The regionalization problem has been a well-studied combinatorial problem since the 1970s [5]. Regionalization methods can be broadly classified as spatially implicit or spatially explicit models, depending on how they represent the spatial contiguity constraints of the formed regions [9]. Implicit methods initially apply traditional or non-spatial clustering methods to obtain a preliminary solution, which is then adjusted to enforce spatial constraints [16, 17]. Conversely, explicit models enforce spatial contiguity constraints from the outset [9].

Exact methods provide an optimal guarantee for the solutions. However, they are considered computationally intensive and still limited to small problems [8], meaning they are suited for situations with a low number of zones and regions. In contrast, heuristic approaches, which are more scalable, do not guarantee optimal solutions.

Among heuristic approaches with spatially explicit constraints, tree-based methods such as SKATER [14, 2] and REDCAP [12] are widely used and have been demonstrated to generate near-optimal partitions with acceptable computational costs [1, 6]. Both SKATER and REDCAP employ a two-step approach that first constructs a spatially contiguous tree connecting all the zones, then greedily splits it to create the desired number of regions.

Certain spatially explicit regionalization methods, such as those described by [7] and [18], do not require a predetermined number of regions. Instead, these methods aim to identify underlying regions while imposing constraints related to these regions.

It is worth noting that the regionalization problem can be viewed as a variant of the optimal graph partitioning problem [4]. The main difference is that, in graph partitioning, the dissimilarities between non-adjacent pairs of nodes are not considered by the objective function.

### 3 Proposed Approach

We start by stating formally the problem. Let us denote by  $V = \{1, 2, \dots, n\}$  the zones of our map or more generally the zones. Let  $G = (V, E)$  represent the contiguity graph of the map, where edges  $(i, j) \in E$  exist if and only if zones  $i$  and  $j$  share a common border on the map.  $G$  must be connected. Let  $P = \{V_1, V_2, \dots, V_k\}$  represent a partition (areas) of the zones  $V$  into  $k$  regions, with  $1 \leq k \leq n$ . A partition  $P$  is considered feasible if all areas are disjoint, cover the original set of zones, and the induced subgraphs  $G(V_u)$  are connected for all  $u \in \{1, 2, \dots, k\}$ . Let  $(a_{i,1}, a_{i,2}, \dots, a_{i,m})$  denote the  $m$  numerical attributes of zone  $i$ . The quadratic distance or dissimilarity between zones based on the set of numerical attributes is  $d_{i,j} = \sum_{l=1}^m (a_{i,l} - a_{j,l})^2$ . The *heterogeneity* of a region  $V_u$  is defined as  $h(V_u) = \frac{1}{|V_u|} \sum_{i < j, i, j \in V_u} d_{i,j}$ . It can be shown that the heterogeneity can equivalently be computed as the sum of squared distances to the mean of attributes  $h(V_u) = \sum_{i \in V_u} \sum_{l=1}^m (a_{i,l} - \bar{a}_l)^2$  with  $\bar{a}_l = \frac{1}{|V_u|} \sum_{i \in V_u} a_{i,l}$ . The regionalization problem is to find a feasible  $k$ -partition minimizing the overall heterogeneity  $H(P) = \sum_{u=1}^k h(V_u)$ .

#### 3.1 The REDCAP Two-Step Approach

REDCAP [12] is a state-of-the-art method for solving the regionalization problem. This approach consists of two consecutive steps. First, it identifies a spanning tree,  $T$ , of graph  $G$  (also referred to as a spatially contiguous tree (SCT) in this context) using a hierarchical clustering approach. Second, it identifies  $k - 1$  edges that partition the tree into a forest of  $k$  subtrees, each of which constitutes the final cluster of the regionalization problem.

**Step1.** Starting initially with a set of  $n$  clusters  $C$ , each one containing one of the zones  $C = \{c_1 = \{1\}, c_2 = \{2\}, \dots, c_n = \{n\}\}$ , the hierarchical clustering approach merges at each step, the two closest contiguous clusters  $c_I$  and  $c_J$  until one single cluster regroups all the zones. Two clusters  $c_I$  and  $c_J$  are considered contiguous if there is an edge in the connectivity graph  $G$  linking two zones from each cluster  $c_I$  and  $c_J$ . The distance between two clusters  $c_I$  and  $c_J$  denoted as  $D(c_I, c_J)$  and can be computed with different variants. The variant that generally yields the best results is called full-order complete linkage (Full-Order-CLK) defined as  $D(c_I, c_J) = \max_{i \in c_I, j \in c_J} d_{i,j}$ . Initially empty, one edge is thus added to the SCT  $T$  each time two clusters  $c_I$  and  $c_J$  are merged. This edge  $e \in E$  is the one of the original connectivity graph with minimal cost i.e.  $\operatorname{argmin}_{(i,j) \in E | i \in c_I, j \in c_J} d_{i,j}$ . At the end of the procedure,  $T$  contains  $n - 1$  edges connecting all the nodes. Overall, the computational complexity for building  $T$  using the aforementioned method is  $O(n^2 \log n)$ .

**Step2.** The second step of REDCAP [12] to obtain  $k$  homogeneous regions is to identify  $k - 1$  edges to remove from  $T$  as illustrated in Figure 1c. The remaining components form the final  $k$  regions. Due to the inherent complexity of finding an optimal solution for the second-step tree partitioning problem [14], this problem is solved in REDCAP using a greedy heuristic. Let us denote by  $F = \{T_1, T_2, \dots, T_k\}$  the spanning forest obtained after the removal of  $k - 1$  edges from  $T$ . The set of nodes and edges of each tree  $T_u$  are denoted by  $V_u$  and  $E_u$ .

At each iteration, one edge is taken out, splitting one tree of the forest into two trees. Notice that a subtree can possibly contain a single node in case a leaf-edge is removed. The edge that results in the greatest decrease in heterogeneity (or in other words, the highest *homogeneity gain*) is chosen to be eliminated.

For a tree  $T_u = (V_u, E_u)$ , the homogeneity gain  $h_g(e)$  obtained by the removal of an edge  $e \in E_u$ , dividing the tree into two trees  $T_{u_1}$  and  $T_{u_2}$  is defined as  $h_g(e) = h(V_u) - h(V_{u_1}) - h(V_{u_2})$  where  $V_{u_1}, V_{u_2}$  are the nodes in the corresponding sub-trees  $T_{u_1}$  and  $T_{u_2}$ . The complexity of this greedy algorithm is  $O(k \cdot n^2)$  but as  $k$  is usually much smaller than  $n$ , it can be ignored. We propose to replace this second-step greedy algorithm for the tree partitioning problem by an exact formulation using dynamic programming and MDD-based optimization as explained in the next section.

### 3.2 Edge Removal using MDD-based Optimization

We express the problem of the optimal removal of  $k - 1$  edges from the spanning tree  $T = (V_T, E_T)$  to minimize the heterogeneity as a Dynamic Programming Problem. We use a sequence of  $k - 1$  decision variables  $x_i$  representing the edges that are successively removed from the SCT. The domain of each of these variables is the set of edges  $E_T$  of the tree. The search-space can be described as Layered Transition Diagram, also called Multivalued Decision Diagrams (MDD) [3]. Let us denote by  $F_i$  the set of possible forests obtained by removing exactly  $i$  edges from  $T$ . This set of forests  $\bigcup_{0 \leq i \leq k-1} F_i$  constitutes the state space and the corresponding nodes of the MDD. A state (forest) is denoted by  $f = (V_f, E_f)$ . Since we always remove edges on the transitions, but not nodes, the set of nodes of each forest of the state space remains the one of the original contiguity graph  $V_f = V, \forall f \in F_i, \forall 0 \leq i \leq k - 1$ . Let us now describe the important nodes, the transition function and cost functions for the MDD:

- The set of state-spaces  $F = \{F_0, \dots, F_{k-1}\}$  forms the layers, where  $F_i$  corresponds to all the states formed by removing exactly  $i$  edges from  $T$ .
- The root of the MDD is denoted as  $r$  and corresponds to the state  $f^0 \in F_0$ , with  $f^0 = T$ , and its initial value is  $v_r = -h(V_T)$ , representing the heterogeneity of the entire original map.
- The terminal states are denoted by  $t$  and regroup every state  $f^{k-1} \in F_{k-1}$ .
- The set  $\tau$  of transition functions s.t.  $\tau_i : F_i \times E \rightarrow F_{i+1}$  for  $i = 0, \dots, k - 2$  taking the system from one state  $f^i$  to the next state  $f^{i+1}$  based on the edge removed.
- The set  $c$  of transition cost functions  $c_i : F_i \times E \rightarrow \mathbb{R}$  s.t.  $c_i(f, e)$  is the homogeneity gain  $h_g(e)$  of making the decision for  $x_i$  to remove the edge  $e$  from the forest  $f$  at the level  $i$ . The objective function is then to maximize  $v_r + \sum_{i=0}^{k-2} c_i(f^i, x_i)$  so that  $f^{i+1} = \tau_i(f^i, x_i)$  and  $x_i \in E_T, \forall i \in \{0, \dots, k - 2\}; f^i \in F_i, \forall i \in \{0, \dots, k - 1\}$ . The optimal solution can be obtained by searching the longest path from the root  $r$  to one of the terminal nodes  $t$ .

### 3.3 Branch-and-Bound with MDD

The number of states in the MDD augments rapidly with  $k$  and  $n$  ( $n - 1$  choose  $k - 1$ ). For such a situation, Bergman et al. [3] have introduced a branch-and-bound (BnB) framework to explore the state space of the MDD without generating it completely upfront and keeping the memory requirement limited. In BnB based on MDDs, relaxed and restricted MDDs, obtained by limiting the width of the MDDs, are used to efficiently explore and prune the solution space. A relaxed MDD is obtained by state-merging. It is an over-approximation of the solution space, where some infeasible solutions might be included. The optimal path of a relaxed MDD provides an upper-bound, which can be used to prune the search space. A restricted MDD, on the other hand, is an under-approximation of the solution space. It is obtained by discarding the less promising states. Some feasible solutions might be excluded but it can nevertheless be used to get lower bounds (similarly to a beam-search). MDD based

BnB enqueues nodes of the original MDD in the queue. When a node is popped, a restricted and a relaxed MDD are compiled from this node to hopefully improve the incumbent solution and prune the search by upper-bounding. This combination of dynamic compilation of restricted and relaxed MDD allows for a more effective exploration of the solution space of the MDD and helps to find the optimal solution in a computationally efficient manner. The nodes can also be pruned by computing a (cheap) upper-bound [11]. We describe next the state-merging procedure and the cheap upper-bound for the optimal edge removal problem.

### 3.3.1 State Merging

As opposed to restricted MDD, a relaxed MDD encodes a superset of the solutions of the original MDD and thus leads to an upper-bound. Its construction is limited to a given width by applying a problem-specific merge operator. In the context of the edge-removal problem, the merge operator applied to two nodes at a same level simply consists in taking the union of the edges in the two forests:  $\text{merge}(f_A = (V, E_{f_A}), f_B = (V, E_{f_B})) = (V, E_{f_A} \cup E_{f_B})$ . The costs of the arcs leading to the merged nodes remain unchanged. Notice that in a relaxed MDD, it is no longer true that the forests  $f^i$  at level  $i$  have exactly  $i + 1$  components. One can be convinced that this merging definition correctly includes a superset of the possible paths. It also guarantees an upper-bound on the optimal homogeneity gain that would be obtained without compression. This is a direct consequence of the following property.

► **Lemma 1.** *For a tree  $T = (V, E)$  and a super-tree of  $T$  denoted  $T' = (V', E')$  with  $V \subseteq V'$  and  $E \subseteq E'$ , let  $e$  be an edge present in both  $E$  and  $E'$ . The homogeneity gain of this edge removal in  $T$  denoted  $h_g(e)$  is lower than  $h'_g(e)$  i.e. when this edge is removed from  $T'$ .*

**Proof.** Assuming  $e$  connects the two sub-trees  $T_1 = (V_1, E_1)$  and  $T_2 = (V_2, E_2)$  of  $T$  and  $T'_1 = (V'_1, E'_1)$  and  $T'_2 = (V'_2, E'_2)$  of  $T'$ . The homogeneity gain in  $T$  is  $h_g(e) = h(V) - h(V_1) - h(V_2)$  which can equivalently be computed as  $h_g(e) = \sum_{i \in V_1} \sum_{j \in V_2} d_{i,j}$ . Similarly  $h'_g(e) = \sum_{i \in V'_1} \sum_{j \in V'_2} d_{i,j}$ . Therefore  $h'_g(e) - h_g(e) = \sum_{i \in V'_1 \setminus V_1} \sum_{j \in V'_2 \setminus V_2} d_{i,j} \geq 0$ . ◀

### 3.3.2 Cheap Upper-Bound

To efficiently compute an upper-bound for a state forest  $f^i$  at level  $i$ , one can assume that the  $k - i - 1$  remaining edges that will be removed induce sub-trees that are perfectly homogeneous (null heterogeneity). The upper-bound on the total homogeneity gain starting from  $f^i$  can then be calculated as :  $\sum_{l=1}^{\min(k-i-1, |f^i|)} h(V_l^*)$  where  $V_1^*, \dots, V_{k-i-1}^*$  are the  $k - i - 1$  trees of  $f^i$  having the highest heterogeneity value. Notice that the count of trees in  $f^i$  may be fewer than  $k - i - 1$ . Should this occur, the formula accounts for the heterogeneity of all present trees in the state.

## 4 Experiments

To assess the effectiveness of our MDD-based method for partitioning spatially contiguous trees, we carried out experiments that compare our approach to REDCAP using both real-life datasets and synthetic datasets with known ground-truth regions.

**Real-life datasets.** We use a set of 5 real-life regionalization datasets varying in size, geometry and number of attributes. Since comparing the absolute value of heterogeneity is meaningless, we use the *Rescaled Overall Heterogeneity*  $H_r$ . That corresponds to the ratio between the overall heterogeneity of the MDD approach and the one of REDCAP.



- **Economic and demographic indicators in NUTS zones:** We collected economic and demographic data on the NUTS areas of Europe to create several datasets that can be used as instances of the regionalization problem. Using the Eurostat database, we gathered data on the density, median age, average GDP per inhabitant, and migration rate for each NUTS-1, NUTS-2, and NUTS-3 European zones in 2019. We constructed one regionalization dataset for each level of NUTS classification, which we refer to as *Ecodemo NUTS1*, *Ecodemo NUTS2*, and *Ecodemo NUTS3*. After removing the unconnected NUTS zones, we are left with 94 zones for the *Ecodemo NUTS1* dataset, 236 for the *Ecodemo NUTS2* dataset, and 1,155 for the *Ecodemo NUTS3* dataset, each having four attributes.
- **Education in Belgium:** We collected data on the level of education in each municipality in Belgium for the year 2017 from the StatBel Open Data. Using this data, we created a regionalization dataset named *Education BE*, where the Belgian municipalities serve as the zones, and their attributes include the share of low, medium, and highly educated inhabitants living in their respective territories. This dataset comprises a total of 563 zones, each having three attributes.
- **USA Ecoregions:** Ecoregions are geographic regions of ecological systems based on vegetation, climate conditions, and land cover [15]. They are frequently used in conservation ecology for planning urban and agricultural development while preserving biodiversity. We use the same dataset as [1] to evaluate our regionalization model. The dataset gathers climatic and land-cover measurements from 1994 in 186 zones of the USA territory. Once the isolated zones are removed, our *Ecoregions USA* dataset includes 172 zones, each one having 15 different ecological attributes.

**Synthetic Maps.** We evaluate the performance of regionalization models in recovering the original regions on synthetic maps using supervised-learning metrics. Our synthetic maps are generated following the methodology proposed in [1], and are parameterized by the number of zones (cells of a square grid), number of regions, region fuzziness, and region geometries.

We created 3 different classes of synthetic maps obtained for different settings of the parameters of the generation process. Each class describes a different level of complexity for regionalization methods. The family *A* regroups maps with 100 zones divided in 5 regions with simple concentric geometries and well-delimited attribute values. The family *B* consists of synthetic maps of 400 zones distributed in 10 regions, with more complex and different geometries, and less pronounced frontiers between regions due to their more similar mean values. Finally, the family *C* comprises maps with 900 zones divided in 20 regions having more complex geometries and more diverse sizes but similar fuzziness than family *B*. An example of map for each family is represented on Figure 2.

In the case of synthetic datasets, we have access to the original partition, or the ground-truth. Consequently, the optimization problem can be reframed as a machine learning problem, with the goal of recovering the original partition. By comparing the assigned regions with the ground truth, we can calculate various machine learning metrics to assess the performance of the regionalization models. We use the pairwise comparison to evaluate the regionalization algorithms. Each pair of zones is labeled as positive if they belong in the same original region and as negative if they come from different ones. One can then compute the classical True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN) rates and evaluate the *precision*, *recall* and *F<sub>1</sub> score* metrics to evaluate the regionalization methods on the synthetic maps. In addition to these 3 supervised metrics, we also calculate the ratio between the overall heterogeneity of the partition generated by the regionalization algorithm with the one of ground-truth partition. We name this metric the *True Overall Heterogeneity Ratio H<sub>T</sub>*.



■ **Figure 2** Examples of ground-truth regions for each synthetic map class. From left to right, we have an example of the regions' geometry of a synthetic map from family *A*, family *B* and family *C*. The zones belonging to the same region are colored in the same color.

■ **Table 1** Comparison of MDD and REDCAP approaches on real-life regionalization problems. The table presents the rescaled overall heterogeneity for each dataset and partition size ( $k \in \{5, 10, 15, 20\}$ ).

Dataset	Number of regions $k$			
	5	10	15	20
Ecodemo NUTS1	1.00	0.98	0.96	0.965
Ecodemo NUTS2	0.989	0.97	0.958	0.935
Ecodemo NUTS3	0.997	1.00	1.00	1.00
Education BE	0,956	0.861	0.887	0.916
Ecoregions USA	1.00	0.997	0.963	0.953

## 5 Results

For each regionalization problem, we construct the corresponding SCT using REDCAP's hierarchical clustering. In the second step, we compare the greedy edge removal of REDCAP with the one computed using MDD-based optimization. We employ the DDO solver [10] with a width of 50 for both restricted and relaxed DDs and set a timeout of 100 seconds to identify the final  $k$  regions. Before the regionalization process, we normalize the attributes of the zones within a range between 0 and 1 using a MinMax scaler.

### 5.1 Real-life Instances

We assessed the performance of the MDD approach and REDCAP in generating 5, 10, 15, and 20 regions for each dataset described. The comparison between the two methods on the real-life regionalization problems is presented in Table 1. The table provides insights into the quality of the methods' partitions by displaying the rescaled overall heterogeneity for each dataset and for the partition sizes  $k = 5, 10, 15, 20$ .

Regardless of the requested number of regions, our experimental results show that the MDD approach consistently matches or outperforms REDCAP in terms of overall heterogeneity for each regionalization dataset. This indicates that the partitions generated by the MDD approach are of higher quality. However, the degree of difference varies depending on the dataset. For instance, in the case of the Education BE dataset, the MDD approach reduces the heterogeneity of the partition by almost 15% for 10 regions compared to REDCAP. In contrast, for the Ecodemo NUTS3 dataset, the two algorithms produce similar partitions for all values of  $k$ . Although the MDD approach only achieved optimality for the Ecodemo NUTS1 dataset when  $k = 5$ , it outperformed REDCAP for all datasets.

■ **Table 2** Number of seconds taken by the MDD approach to find the first solution for each real-life regionalization problem. The time at which it discovers the best solution is presented in parentheses in the cases where it is not equivalent to the first solution founded.

Dataset	Number of regions $k$			
	5	10	15	20
Ecodemo NUTS1	<1	<1	<1	<1
Ecodemo NUTS2	<1	<1	2	2 (28)
Ecodemo NUTS3	6	13	19	21
Education BE	2 (18)	3 (85)	5 (97)	6
Ecoregions USA	<1	<1	<1 (21)	2 (24)

■ **Table 3** Comparison of our MDD approach with REDCAP on synthetic datasets

Metric	A		B		C	
	MDD	REDCAP	MDD	REDCAP	MDD	REDCAP
Precision	<b>0,988</b>	0,944	<b>0,966</b>	0,912	<b>0,956</b>	0,954
Recall	<b>0,989</b>	0,962	<b>0,949</b>	0,891	<b>0,951</b>	0,908
$F_1$ Score	<b>0,988</b>	0,952	<b>0,956</b>	0,903	<b>0,954</b>	0,928
$H_T$	<b>0,99</b>	1,19	<b>0,978</b>	1,108	<b>1,015</b>	1,078

Regarding performance, there is a significant difference between the MDD approach and REDCAP. REDCAP takes, on average, between 0.01 and 0.3 seconds to produce a partition depending on the dataset and the number of regions requested, while the MDD approach is allowed to use up to 100 seconds to obtain the best possible solution. However, for the most part, the first partition discovered by the MDD approach is also the best one obtained within the 100-seconds timeframe. Table 2 presents the amount of time the MDD approach searched before finding the first valid partition for each regionalization problem. It is noteworthy that this first partition found by the MDD approach has always a lower or equal overall heterogeneity than the partition found by REDCAP. Additionally, if this first solution is not the best one found within the 100-seconds timeframe, Table 2 reports in parentheses the time taken by the MDD approach to discover the partition with the lowest overall heterogeneity.

## 5.2 Synthetic Maps

We evaluated both the REDCAP and MDD approaches on various synthetic maps of different sizes and complexities (families A, B, and C). For each family, we generated and assessed 20 maps using both methods. Table 3 displays the mean Precision, Recall, F1 Score, and True Overall Heterogeneity Ratio for each method, computed using the ground-truth values. The MDD approach was able to prove the optimal solution for the edge removal problem only for instances belonging to family A.

We can see that using the MDD-based approach for the second step of REDCAP improves the solution in all metrics for the three synthetic map families. Comparing the results of the MDD approach, it can be seen that the precision and recall are lower for families B and C. Both of these families share a characteristic in that their region boundaries are more fuzzy than those of family A. This suggests that our method encounters more difficulty in recovering the initial partition when the attributes between two neighboring regions are more similar, i.e. when the delimitations between regions are less pronounced. The number of zones, the number of regions and their geometric complexity seem to have a lesser impact on the capacity of our model to recover the original regions.

Moreover, for families A and B, the MDD approach generates regions with a lower overall heterogeneity than the ground-truth partition on average. This implies that the original partition is not always the optimal one in terms of overall heterogeneity. Thus, it can be concluded that for the first two families, the regions generated by the MDD approach deviate from the ground-truth ones simply because it discovered partitions with lower overall heterogeneity than the original ones.

## 6 Conclusion

In this paper, we have proposed a novel approach for the regionalization problem, an essential clustering task in a variety of spatial analysis domains. We have improved upon the second step of the well-established REDCAP algorithm by introducing an exact dynamic programming formulation for the edge removal problem solved using a multi-valued decision diagram (MDD)-based branch and bound solver. We have provided comprehensive experiments on both real-life datasets and synthetic ones to illustrate the efficacy of our method. Our comparison with the REDCAP algorithm using a wide range of supervised and unsupervised metrics demonstrated that our approach consistently produces partitions of higher quality.

---

### References

- 1 Orhun Aydin, Mark Janikas, Renato Assunção, and Ting-Hwan Lee. A quantitative comparison of regionalization methods. *Int. J. Geogr. Inf. Sci.*, 35:1–29, 2021. doi:10.1080/13658816.2021.1905819.
- 2 Orhun Aydin, Mark V Janikas, Renato Assuncao, and Ting-Hwan Lee. Skater-con: Unsupervised regionalization via stochastic tree partitioning within a consensus framework using random spanning trees. In *Proceedings of the 2nd ACM SIGSPATIAL international workshop on AI for geographic knowledge discovery*, pages 33–42, 2018.
- 3 David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016. doi:10.1287/ijoc.2015.0648.
- 4 Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. *Recent advances in graph partitioning*. Springer, 2016.
- 5 Andrew D Cliff and Peter Haggett. On the efficiency of alternative aggregations in region-building problems. *Environment and Planning A*, 2(3):285–294, 1970.
- 6 Diep Dao and Jean-Claude Thill. Detecting Attribute-Based Homogeneous Patches Using Spatial Clustering: A Comparison Test. In *Information Fusion and Geographical Information Systems*, pages 37–54. January 2018. doi:10.1007/978-3-319-59539-9\_4.
- 7 Juan C. Duque, Luc Anselin, and Sergio J. Rey. THE MAX-P-REGIONS PROBLEM\*. *Journal of Regional Science*, 52(3):397–419, August 2012. doi:10.1111/j.1467-9787.2011.00743.x.
- 8 Juan C Duque and Richard L Church. A new heuristic model for designing analytical regions. In *North American Meeting of the International Regional Science Association, Seattle*, 2004.
- 9 Juan Carlos Duque, Raúl Ramos, and Jordi Suriñach. Supervised Regionalization Methods: A Survey. *International Regional Science Review*, 30(3):195–220, July 2007. Publisher: SAGE Publications Inc. doi:10.1177/0160017607301605.
- 10 X. Gillard, P. Schaus, and V. Coppé. Ddo, a generic and efficient framework for mdd-based optimization. International Joint Conference on Artificial Intelligence (IJCAI-20); DEMO track, 2020.
- 11 Xavier Gillard, Coppé Vianney, Pierre Schaus, and Ciré André. Improving the filtering of branch-and-bound mdd solvers. In *CPAIOR*, 2021.
- 12 D. Guo. Regionalization with dynamically constrained agglomerative clustering and partitioning (REDCAP). *International Journal of Geographical Information Science*, 22(7):801–823,

## 45:10 Partitioning a Map into Homogeneous Regions: A BnB Approach Using DD



July 2008. Publisher: Taylor & Francis \_eprint: <https://doi.org/10.1080/13658810701674970>. doi:10.1080/13658810701674970.

- 13 Paul A Longley, Michael F Goodchild, David J Maguire, and David W Rhind. *Geographic information science and systems*. John Wiley & Sons, 2015.
- 14 Assuncao Martins, Marcos Neves, Gilberto Câmara, and Domingos Da Costa Freitas. Efficient Regionalization Techniques for Socio-Economic Geographical Units Using Minimum Spanning Trees. *International Journal of Geographical Information Science*, 20:797–811, August 2006. doi:10.1080/13658810600665111.
- 15 James M Omernik, Shannen S Chapman, Richard A Lillie, Robert T Dumke, et al. Ecoregions of wisconsin. *Transactions of the Wisconsin Academy of Sciences, Arts and Letters*, 88:77–103, 2000.
- 16 Stan Openshaw. A regionalisation program for large data sets. *Computer Applications*, 3(4):136–147, 1973.
- 17 Stan Openshaw. Classifying and regionalizing census data. *Census users' handbook*, pages 239–270, 1995.
- 18 Ran Wei, Sergio Rey, and Elijah Knaap. Efficient regionalization for spatially explicit neighborhood delineation. *International Journal of Geographical Information Science*, 35(1):135–151, 2021. doi:10.1080/13658816.2020.1759806.

# Constraint Programming to Improve Hub Utilization in Autonomous Transfer Hub Networks

Chungjae Lee   

H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

Wirattawut Boonbandansook  

H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

Vahid Eghbal Akhlaghi   

H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

Kevin Dalmeijer<sup>1</sup>   

H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

Pascal Van Hentenryck   

H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

---

## Abstract

The Autonomous Transfer Hub Network (ATHN) is one of the most promising ways to adapt self-driving trucks for the freight industry. These networks use autonomous trucks for the middle mile, while human drivers perform the first and last miles. This paper extends previous work on optimizing ATHN operations by including transfer hub capacities, which are crucial for labor planning and policy design. It presents a Constraint Programming (CP) model that shifts an initial schedule produced by a Mixed Integer Program to minimize the hub capacities. The scalability of the CP model is demonstrated on a case study at the scale of the United States, based on data provided by Ryder System, Inc. The CP model efficiently finds optimal solutions and lowers the necessary total hub capacity by 42%, saving \$15.2M in annual labor costs. The results also show that the reduced capacity is close to a theoretical (optimistic) lower bound.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming

**Keywords and phrases** Constraint Programming, Autonomous Trucking, Transfer Hub Network

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.46

**Category** Short Paper

**Funding** This research was partly funded through a gift from Ryder and partly supported by the NSF AI Institute for Advances in Optimization (Award 2112533).

**Acknowledgements** Special thanks to the Ryder team for their invaluable support, expertise, and insights.

## 1 Introduction

It is widely believed that autonomous trucks will revolutionize the freight industry, and many companies have started exploring its potential [5, 6, 7, 11, 26, 27]. Some of the major players describe the *transfer hub business model* to be the most likely implementation for

---

<sup>1</sup> Corresponding author



autonomous trucking [21, 25, 28]. An Autonomous Transfer Hub Network (ATHN) is a network of autonomous truck ports (*transfer hubs*) that leverages the strengths of humans and automation in their most effective roles. Autonomous trucks handle the monotonous middle mile to transport goods between the transfer hubs, while humans handle the complex first and last miles through local cities and deal with customer contacts. According to Roland Berger [21], implementing a transfer hub model can lower operational costs by 22% to 40%. A case study in the Southeast of the United States by Ryder System, Inc. and the Socially Aware Mobility lab [22] reports savings from 27% to 40%, supporting earlier estimates. The authors model optimizing ATHN operations as a scheduling problem, and a Constraint Programming (CP) model is used to minimize the empty miles [4]. Subsequent work by [13] presents a column-generation approach and a bespoke network-flow model to quickly find high-quality solutions. These findings are combined by [14] into a flow-based Mixed Integer Programming (MIP) framework that can solve large-scale instances over a long time horizon (e.g., a month) optimally in reasonable time. These capabilities also allow for detailed analyses of the benefits and costs of ATHNs.

This paper builds on previous work by introducing a framework for optimizing both ATHN operations and transfer hub capacity utilization. Capacity planning plays a critical role in an actual implementation of the ATHN for operational scheduling. Furthermore, hub capacity has an impact on the number of essential personnel required, which is a crucial factor to consider in labor planning and policy design. Prior research on ATHN operations primarily focused on routing and scheduling, but neglected the capacity considerations involved. Including capacity constraints in a scalable way is not obvious. The CP method in [4] could be extended to incorporate hub capacity constraints by modeling a Resource-Constrained Project Scheduling Problem [9, 12, 20] at every hub and adding cumulative constraints [1, 24] for hub capacity. However, the poor performance reported in [4] makes it unlikely this method will provide good solutions on the national level. The MIP framework from [14] does handle large scale systems, but does not support a cumulative constraint. Alternatives typically require sophisticated modeling techniques and solution methods such as Time-expanded Networks or Dynamic Discretization Discovery [2, 3, 10, 15, 23, 29, 30], which are not obvious to scale either.

The method presented in this work optimizes ATHN operations and improves hub utilization even for large-scale systems by combining the strengths of MIP and CP. The MIP model is used to generate routes and an initial schedule, while the CP model is used to shift the schedule and minimize the required hub capacities. A case study based on real data is conducted to demonstrate the effectiveness of the new methodology on an ATHN system spanning the United States for a four-week horizon. The proposed CP model efficiently finds optimal solutions and lowers the necessary total hub capacity by 42%. This reduction in capacity may save \$15.2M per year in labor cost. Furthermore, it is shown that this is close to the best possible savings for any initial schedule. This paper also includes a sensitivity analysis and provides operational insights for future implementation of the ATHN framework. The remainder of this paper is organized as follows. Section 2 presents the MIP and CP methodology. Section 3 describes the data and experimental settings used in the case study. Results and sensitivity analysis are presented in Section 4. Finally, Section 5 provides conclusions and suggests directions for future research.

## **2** Methodology

This paper uses the methods by [14] to design an ATHN and to optimize the routes of the autonomous vehicles with a MIP. The resulting solution minimizes the cost of the system, but does not take into account the necessary capacity at each of the hubs, which may lead

to low hub utilization. To address this issue, this paper introduces a CP model to shift the schedule in such a way that the original time windows remain satisfied, and the necessary hub capacities are minimized.

## 2.1 ATHN Design and Operations

The input data for the design and optimization of ATHN consists of a set of loads with origins, destinations, and release times. Following [14], K-means clustering is used to determine hub locations  $H$ , and loads are assigned to hubs according to a hub-assignment rule that minimizes the total driving distance (with autonomous miles discounted by a factor  $\gamma$ ).

To optimize the ATHN operations, first define a set of tasks  $T = \{1, 2, \dots\}$ , where each task  $t \in T$  corresponds to moving a load from origin hub  $h_t^+ \in H$  to destination hub  $h_t^- \in H$  with an autonomous truck. Each task  $t \in T$  is associated with a desired pickup time  $p_t$  and a flexibility  $\Delta \geq 0$ , resulting in a pickup time window of  $[p_t - \Delta, p_t + \Delta]$ . A *task graph*  $G = (V, A)$  is introduced to find the optimal sequence of tasks for every vehicle. The set  $V = T \cup \{0\} \cup \{|T| + 1\}$  consists of vertices that correspond to the tasks, together with a source node 0 and a sink node  $|T| + 1$ . Choosing an arc  $a \in A$  indicates that the corresponding tasks are performed sequentially by the same vehicle. An arc between two tasks  $t, t' \in T$  represents loading at  $h_t^+$ , moving freight from  $h_t^+$  to  $h_t^-$ , unloading at  $h_t^-$ , and relocating from  $h_t^-$  to  $h_{t'}^+$  to be ready for the next task. Each arc is associated with a corresponding time  $\tau_a$ . The time for loading or unloading is given by a parameter  $\sigma$ , and OpenStreetMap times are used for driving and relocation [17].

The cost is calculated in two parts:  $d_t$  is the *direct cost*, which represents the cost of serving task  $t \in T$  directly with a conventional truck without using any of the hubs. This cost is taken to be the total distance for delivery and empty return. Note that the current non-autonomous system corresponds to using only direct trips. The second component is a *cost differential*  $c_a$  associated with each arc  $a \in A$ . For arc  $a = (t, t')$  this represents the difference in cost to switch task  $t$  from conventional to autonomous delivery, including the relocation cost from  $h_t^-$  to  $h_{t'}^+$ . That is,  $d_t$  is the direct trip cost, and  $d_t + c_{tt'}$  is the cost to serve task  $t \in T$  autonomously and relocate to the start of task  $t'$ . The autonomous middle miles are discounted by a factor  $\alpha \in [0, 1]$  and it is assumed that a fraction  $\beta \in [0, 1]$  of the implied first/last miles are empty. Additional details on how  $\tau_a$  and  $c_a$  are calculated are provided by [14].

Let  $y_a$  be a binary variable that indicates that arc  $a \in A$  is selected, and let  $x_t$  be the start time of task  $t \in T$ . For convenience, let  $\delta_v^+$  and  $\delta_v^-$  denote the out-arcs and in-arcs of vertex  $v \in V$ , respectively. For a given number of autonomous trucks  $K$ , MIP (1) asks for a set of at most  $K$  routes from source to sink that cover different tasks. Objective (1a) minimizes the total cost. If a task is not covered it means it is served by conventional means and the cost is that of a direct trip  $d_t$ . If the task is covered, the cost differential  $c_{tt'}$  is added to calculate the autonomous cost. Constraints (1b) enforce that every task is performed at most once. Constraints (1c) are flow conservation constraints, and Constraint (1d) limits the number of vehicles to  $K$ . Constraints (1e) are Miller-Tucker-Zemlin constraints [16] that ensure sufficient time passes between subsequent tasks, where  $M$  is a sufficiently large constant. These constraints also eliminate cycles. Finally, the variables and their domains are given by Equations (1f) and (1g). The MIP Model (1) is solved with a blackbox solver after applying the acceleration techniques detailed in [14]. The arc-flows ( $y$ -variables) are translated into a set of routes, which are represented as sequences of tasks, by tracing the flows from the source node 0 to the sink node  $|T| + 1$ . Note that for given optimal routes, the start times ( $x$ -variables) are typically not unique. For consistent analysis, the start times are shifted to as early as possible in post processing.



$$\begin{aligned}
 \min \quad & \sum_{t \in T} d_t + \sum_{a \in A} c_a y_a, & (1a) \\
 \text{s.t.} \quad & \sum_{a \in \delta_t^+} y_a \leq 1 & \forall t \in T, & (1b) \\
 & \sum_{a \in \delta_t^+} y_a = \sum_{a \in \delta_t^-} y_a & \forall t \in T, & (1c) \\
 & \sum_{a \in \delta_0^+} y_a \leq K, & (1d) \\
 & x_{t'} \geq x_t + \tau_{tt'} - M(1 - y_{tt'}) & \forall t, t' \in T, (t, t') \in A, & (1e) \\
 & x_t \in [p_t - \Delta, p_t + \Delta] & \forall t \in T, & (1f) \\
 & y_a \in \mathbb{B} & \forall a \in A. & (1g)
 \end{aligned}$$

■ **Figure 1** Mixed Integer Program for Optimizing ATHN Operations.

## 2.2 Minimizing Required Hub Capacities

The routes and schedule obtained from the MIP do not take hub utilization into account. Therefore, it may happen that many trucks are loading and unloading at the same hub at the same time. To address this issue, a CP model is introduced to shift the schedule to minimize the necessary loading/unloading capacity at the hubs, while satisfying the original time windows and maintaining the same route (sequence of tasks) for every vehicle.

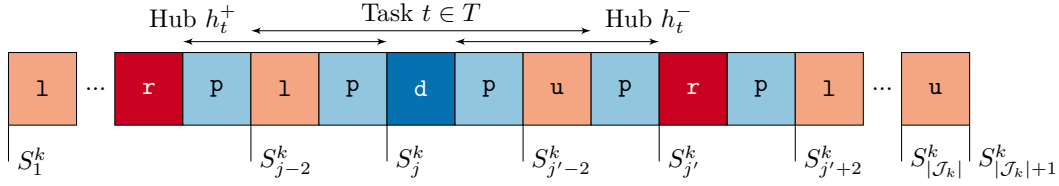
The CP model is based on  $\mathcal{K}$ ; the set of routes obtained from MIP (1). Every route  $k \in \mathcal{K}$  is split into an sequence of jobs  $\mathcal{J}_k = \{1, 2, \dots\}$ . For notational convenience, the jobs are numbered sequentially by the order in which they are performed, rather than using the original task numbers. Figure 2 provides a visualization, which will serve as a running example. Each job  $j \in \mathcal{J}_k$  has up to four properties:

- $\text{type}(j) \in \{1, \text{d}, \text{u}, \text{r}, \text{p}\}$ ; type of job: load, drive, unload, relocate, park, respectively.
- $\text{duration}(j)$  (only for types 1, d, u, r); duration of this job.
- $\text{task}(j) \in T$  (only for types 1, d, u); task associated with this job.
- $\text{hub}(j) \in H$  (only for types 1, u, p); hub associated with this job.

Each route is modeled with the same repeating sequence of loading at the origin hub (1), waiting before driving (p), driving (d), waiting at the destination hub before unloading (p), unloading (u), waiting before relocating to the next task if any (p), relocating (r), and waiting at the origin hub of the next task until loading (p). Note that jobs of type 1, d, u, r have a fixed duration, while the duration of p jobs is flexible and can be zero. When no relocation is necessary (the previous destination is equal to the next origin), the r job is still defined with duration zero for convenience. Every 1, d, u job is trivially associated with an original task  $t \in T$ . Jobs 1, u, p for which the vehicle is standing still are associated with a hub as described above.

### CP Model

The CP Model (2) is based on variables  $S_j^k$  that indicate the start time of job  $j \in \mathcal{J}_k$  in route  $k \in \mathcal{K}$ , and variables  $C_h$  that indicate the necessary loading/unloading capacity at hub  $h \in H$ . For convenience,  $S_{|\mathcal{J}_k|+1}^k$  is defined to represent the time at which the final job



■ **Figure 2** Jobs  $\mathcal{J}_k$  for Route  $k \in \mathcal{K}$  (jobs  $j, j' \in \mathcal{J}_k$  are used as examples in the main text).

$$\min \sum_{h \in H} C_h, \quad (2a)$$

$$\text{s.t. } I_j^k = \text{Interval}([S_j^k, S_{j+1}^k]) \quad \forall k \in \mathcal{K}, j \in \mathcal{J}_k, \quad (2b)$$

$$\text{Cumulative}([I_j^k | k \in \mathcal{K}, j \in \mathcal{J}_k, \text{type}(j) \in \{1, u\}, \text{hub}(j) = h], C_h) \quad \forall h \in H, \quad (2c)$$

$$S_{j+1}^k = S_j^k + \text{duration}(j) \quad \forall k \in \mathcal{K}, j \in \mathcal{J}_k, \text{type}(j) \in \{1, d, u, r\}, \quad (2d)$$

$$\text{dom}(S_j^k) = [\underline{s}_j^k, \bar{s}_j^k] \quad \forall k \in \mathcal{K}, j \in \mathcal{J}_k. \quad (2e)$$

■ **Figure 3** Constraint Programming Model for Minimizing Required Hub Capacities.

$u$  is completed (see Figure (2)). Objective (2a) minimizes the total necessary hub capacity. To calculate the hub capacity, the model first defines an interval variable  $I_j^k$  for every job (Equation (2b)), which implicitly enforces  $S_j^k \leq S_{j+1}^k$ . Equation (2c) defines a cumulative constraint for every hub  $h \in H$  to collect the intervals of the jobs with type 1 and  $u$  at that hub, and to assign the necessary hub capacity to  $C_h$ . Note that this cumulative constraint has a variable as the capacity. It is also worth mentioning that Constraints (2c) span all the vehicles. Constraints (2d) ensure the correct job duration when a duration is defined ( $p$  jobs are flexible). Finally, Equation (2e) defines the domains of the  $S$ -variables, where constants  $\underline{s}_j^k, \bar{s}_j^k$  remain to be defined. This paper focuses on loading/unloading capacity, but note that CP Model (2) is easily adapted to other objectives, such as minimizing parking space.

To ensure that the time flexibility  $\Delta$  is respected, the domains of the  $S$ -variables need to be defined accordingly. For job  $j \in \mathcal{J}_k$  of route  $k \in \mathcal{K}$  and  $\text{type}(j) = 1$ , the domain is defined around the desired pickup time of the task to match the MIP:  $\text{dom}(S_j^k) = [p_{\text{task}(j)} - \Delta, p_{\text{task}(j)} + \Delta]$ . This domain is translated to the following  $d$  and  $u$  jobs to make sure that the flexibility is not exceeded until the task is complete. With slight abuse of notation, this gives the translated domains  $\text{dom}(S_j^k) = \text{dom}(S_{j-2}^k) + \text{duration}(j - 2)$  for jobs of  $\text{type}(j) \in \{d, u\}$  (see Figure 2).

### Redundant Bounds and Constraints

Non-trivial bounds for jobs of type  $r$  and  $p$  are not strictly necessary, but it is straightforward to derive the following:

$$\text{dom}(S_j^k) = \begin{cases} [\underline{s}_{j-2}^k + \text{duration}(j - 2), \bar{s}_{j+2}^k - \text{duration}(j)] & \text{if } \text{type}(j) = r, \\ [\underline{s}_{j-1}^k + \text{duration}(j - 1), \bar{s}_{j+1}^k] & \text{if } \text{type}(j) = p. \end{cases} \quad (3)$$

Relocation can only start after unloading is completed, and has to start to be in time for the next loading (see job  $j'$  in Figure 2). In a similar way, parking can only start after the previous job is completed, and parking for duration zero is possible until the upper bound of the next job.

■ **Table 1** Baseline Parameter Values for the Case Study.

Parameter	Value
$n$	6842 loads
$ H $	100 transfer hubs
$\gamma$	40% discount for autonomous mileage during hub-assignment
$\beta$	25% first/last-mile inefficiency
$\alpha$	25% discount for autonomous mileage
$\Delta$	1 hour pickup-time flexibility
$\sigma$	30 minutes autonomous truck loading/unloading time
$K$	100 autonomous trucks

For the current Objective (2a), the following observation is used to reduce the search space: To minimize the loading/unloading capacity, it does not matter when relocation takes place between the jobs  $u$  and  $1$ . It is therefore possible without loss of generality to impose that relocation starts immediately after unloading:

$$S_j^k = S_{j-1}^k \quad \forall k \in \mathcal{K}, j \in \mathcal{J}_k, \text{type}(j) = \mathbf{r}. \quad (4)$$

### 3 Case Study

This paper conducts a realistic case study based on data from Ryder System, Inc. (Ryder), one of the largest transportation and logistics companies in North America. Ryder has provided a dataset that is representative for its dedicated transportation business in the US, reducing the scope to orders that are strong candidates for automation. Following [14], the case study focuses on orders that are *challenging* in the sense that they would currently induce an empty return trip. Every order represents a load with an origin, a destination, and a scheduled release time. The hubs are chosen based on data from October to December 2019, while the experiments are based on 6842 loads in the first four weeks of October.

#### Experimental Settings

The parameter settings are taken from [14] and summarized in Table 1. All methods from Section 2 are implemented in Python 3.9. The MIP models are solved with Gurobi 9.5.2, and the CP models are solved with CP-SAT 9.6 [19]. All time-related data is rounded to the nearest minute to facilitate the integer domains that are required by CP-SAT. The experiments are conducted on a Linux machine with dual Intel Xeon Gold 6226 CPUs on the PACE Pheonix cluster [18]. Each experiment is assigned to use at most 24 cores and 192GB of RAM. If Gurobi runs out of memory, the experiment is repeated on a machine with 384GB of RAM, and if that fails, the number of cores is halved until the solver terminates successfully. CP-SAT did not encounter memory issues at any point. The Gurobi time limit is set to three hours, except for the sensitivity analysis for flexibility  $\Delta$ , which is given 12 hours to obtain better solutions and is warm started with a MIP start. The CP-SAT solver is warm started with the solution obtained by the MIP, which is provided as a hint.

### 4 Results

Figure 4 summarizes the results of using the CP model to minimize the necessary hub capacity for loading and unloading trucks. “Before CP” shows the required capacity if the MIP solution were implemented immediately, while “After CP” shows the results after applying the CP model. The ATHN problem was solved to optimality for all instances except

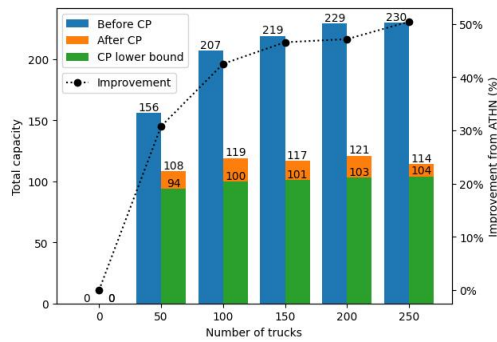


Figure 4 Capacity Before and After CP.

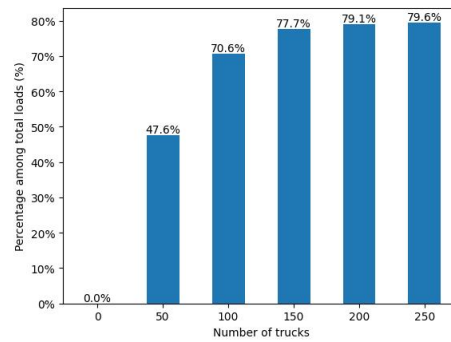
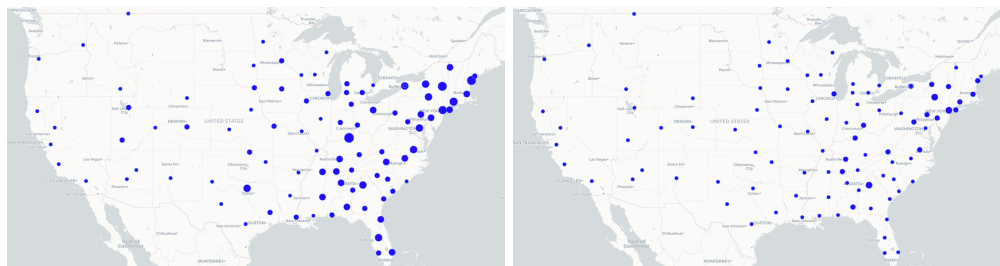


Figure 5 Loads Served Autonomously.



(a) Before CP Optimization.

(b) After CP Optimization.

Figure 6 Capacity Reduction for the 100 Truck Case (circle area proportional to hub capacity).

when  $K = 50$ , which remained at a 0.05% optimality gap within the given time limit. For the CP problem, all instances were solved to optimality in under 30 seconds. The figure shows that the CP model is highly effective in reducing the total hub capacity for all instances, reducing the necessary capacity by 31% up to 50%. For the base case of  $K = 100$  the CP model can reduce capacity by 42%. Note that this may correspond to significant monetary savings: If each unit of loading/unloading capacity requires the assistance of a mechanic around the clock (three shifts of \$57,557 per year [8]), the CP model reduces the annual labor cost by \$15.2 million. Figure 4 also shows that the resulting capacity is close to the lower bound for any ATHN solution, which is obtained by removing the time constraints between subsequent tasks, i.e., treat every task as if it is the only task on the route.

As the number of trucks increases, the ATHN starts serving more loads autonomously, as shown by Figure 5. Without the CP model, this leads to a substantial increase in hub capacity as more loads are added to the system. The CP model completely mitigates this effect, and is able to maintain an almost stable hub capacity as the workload increases. This reveals a surprising robustness to accommodate new orders that the CP model is able to exploit. When making investment decisions and hiring personnel to operate the hubs, this is a very desirable property. The maps in Figure 6 demonstrate that the necessary capacity is reduced throughout the system, and the peaks in the South and Northeast have decreased significantly. The capacity at the hub near Louisville, Kentucky for example was reduced from 7 to 2.

Interestingly, only a small portion of the jobs needs to be rescheduled to obtain the substantial savings in hub capacity. E.g., in the 100-truck case only 13% of loadings were rescheduled. Figure 7 provides histograms for the absolute size of the shift for the loading and unloading times that were shifted. It can be seen that a majority of these jobs were

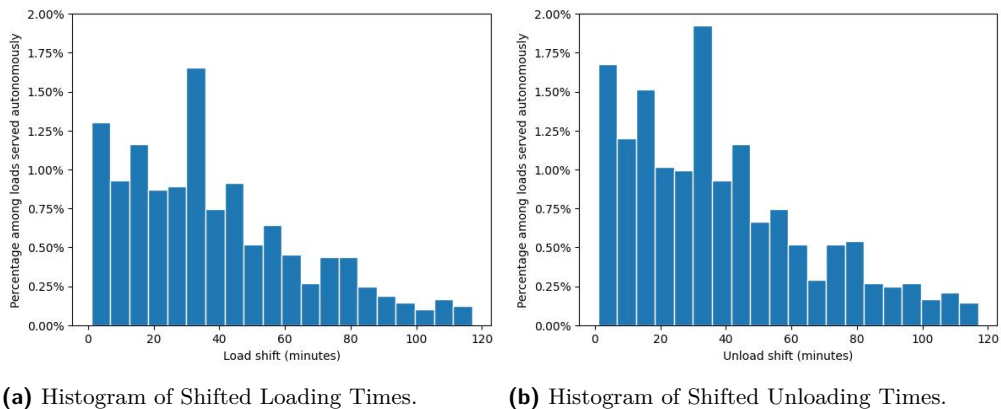


Figure 7 Shifted Schedule Times for the 100 Truck Case.

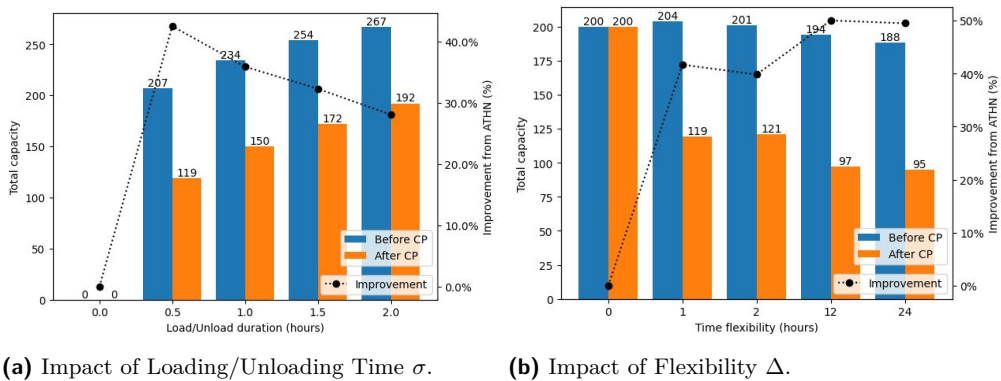


Figure 8 Sensitivity Analysis Loading/Unloading Time and Flexibility for the 100 Truck Case.

moved by no more than 30 minutes, which shows that the ATHN solutions are sufficiently flexible to be adjusted without propagating delays through the schedule. The fact that the CP model is easy to solve and only makes small modifications also makes it an attractive tool at the operational level: if order details are changed or delays are encountered, the CP model can quickly be re-solved to avoid causing overlap in loading and unloading at the hubs.

### Sensitivity Analysis

The loading/unloading time  $\sigma$  and the flexibility  $\Delta$  are two significant factors that affect the ATHN. Figure 8 summarizes how these parameters impact the total hub capacity and the CP model’s performance. The ATHN is solved for loading/unloading time from zero to two hours, and for no flexibility up to one day of flexibility. Note that loading/unloading time up to two hours may be realistic if the vehicle is inspected every time it leaves or enters a hub.

For  $\sigma$ , the ATHN solver found optimal solutions for all instances and the CP problem found optimal schedules within 15 seconds. As  $\sigma$  increases, the total hub capacity increases because jobs overlap more frequently. The CP model is not able to fully compensate for this effect, but can still improve the hub capacity by at least 28%.

For  $\Delta$ , optimal solutions were found for  $\Delta \in \{0h, 1h\}$ , a near-optimal solution within 0.05% optimality gap was found for  $\Delta = 2h$ , while  $\Delta \in \{12h, 24h\}$  remained at a 3% optimality gap. The CP solver took 7 minutes for  $\Delta = 12h$  and 13 minutes for  $\Delta = 24h$

■ **Table 2** CP Solving Time Speedup compared to Omitting Redundant Bounds and Constraints.

Redundant Bounds	Redundant Constraints	Number of Trucks				
		50	100	150	200	250
✗	✗	1.0x	1.0x	1.0x	1.0x	1.0x
✓	✗	1.1x	1.4x	1.2x	0.9x	0.8x
✗	✓	3.9x	6.8x	7.4x	3.9x	5.4x
✓	✓	3.5x	5.5x	6.3x	3.1x	5.5x

to find optimal schedules, which suggests that the CP computation time is more sensitive to changes in  $\Delta$  than in  $\sigma$ . This is explained by the fact that increasing the flexibility significantly increases the search space. Again, the CP model shows substantial improvement over the initial solution with improvements ranging from 39% to 50% for the case study. When the flexibility increases, the necessary capacity goes down even before CP is applied, but the CP model is able to benefit more from the additional freedom.

### Impact of Redundant Bounds and Constraints

Table 2 shows the impact of the redundant bounds and constraints introduced in Section 2. These results are based on the baseline parameter settings for different numbers of trucks. It can be seen that only adding redundant bounds can both speed up or slow down the solver. For example, the solver becomes 1.4 times faster for  $K = 100$  trucks, but a factor 0.8 slower for  $K = 250$  trucks. The main benefit comes from adding redundant constraints, which speeds up the solver by up to 7.4 times for  $K = 150$  trucks. When the redundant bounds and constraints are combined, performance may be improved further, as is the case for  $K = 250$  trucks, but it appears that only including redundant constraints is the most efficient setting for these experiments.

## 5 Conclusion

The Autonomous Transfer Hub Network (ATHN) is one of the most promising ways to adapt self-driving trucks for the freight industry. This paper proposes a framework for optimizing ATHN operations with respect to hub utilization. To accomplish this, a MIP model generates autonomous vehicle routes and an initial schedule, which is then optimized using a CP model to reduce the required hub capacity for loading and unloading trucks. Results from the Ryder case study demonstrate the effectiveness of this approach, with the CP model reducing the total capacity by at least 42% while requiring only minor schedule modifications. This may save \$15.2M per year in labor cost and is close to the lowest possible capacity for any initial schedule. Sensitivity analysis on loading/unloading duration and flexibility provides practical insights for ATHN operations and systematically shows the benefit of the CP model. Future work may explore alternative objective functions, such as minimizing parking space. Another interesting direction is to attempt to jointly optimize routes and hub capacity directly.

### References

- 1 Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993. doi:10.1016/0895-7177(93)90068-A.

- 2 Natasha Boland, Mike Hewitt, Luke Marshall, and Martin Savelsbergh. The Continuous-Time Service Network Design Problem. *Operations Research*, 65(5):1303–1321, 2017. doi:10.1287/opre.2017.1624.
- 3 Natasha Boland and Martin Savelsbergh. Perspectives on integer programming for time-dependent models. *TOP*, 27:147–173, 2019. doi:10.1007/s11750-019-00514-4.
- 4 Kevin Dalmeijer and Pascal Van Hentenryck. Optimizing Freight Operations for Autonomous Transfer Hub Networks. *arXiv:2110.12327*, 2021. arXiv:2110.12327.
- 5 FleetOwner. TuSimple among autonomous truck companies to join Self-Driving Coalition. <https://www.fleetowner.com/>, 2021. URL: <https://www.fleetowner.com/technology/autonomous-vehicles/article/21152006/tusimple-among-autonomous-truck-companies-to-join-selfdriving-coalition>.
- 6 Forbes. Plus Partners With IVECO To Develop Automated Trucks For Global Deployment. <https://www.forbes.com/>, 2021. URL: <https://www.forbes.com/sites/richardbishop1/2021/04/12/plus-partners-with-iveco-to-develop-automated-trucks-for-global-deployment>.
- 7 FreightWaves. Gatik, Isuzu to partner on autonomous truck platform. <https://www.freightwaves.com/>, 2021. URL: <https://www.freightwaves.com/news/gatik-isuzu-to-partner-on-autonomous-truck-platform>.
- 8 Glassdoor. How much does a Truck Mechanic make? [https://www.glassdoor.com/Salaries/truck-mechanic-salary-SRCH\\_K00,14.htm](https://www.glassdoor.com/Salaries/truck-mechanic-salary-SRCH_K00,14.htm), 2023. Accessed: 2023-04-28.
- 9 Sönke Hartmann and Dirk Briskorn. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 297(1):1–14, 2022. doi:10.1016/j.ejor.2021.05.004.
- 10 Edward He, Natasha Boland, George Nemhauser, and Martin Savelsbergh. An exact algorithm for the service network design problem with hub capacity constraints. *Networks*, 80(4):572–596, 2022. doi:10.1002/net.22128.
- 11 Heavy Duty Trucking. Daimler’s Redundant Chassis for Autonomous-Truck Operation. <https://www.truckinginfo.com/>, 2021. URL: <https://www.truckinginfo.com/10157338/daimlers-redundant-chassis-for-autonomous-truck-operation>.
- 12 Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, 25(4):279–302, 1998. doi:10.1016/S0305-0548(97)00055-5.
- 13 Chungjae Lee, Kevin Dalmeijer, and Pascal Van Hentenryck. Optimization Models for Autonomous Transfer Hub Networks. *arXiv:2201.06137*, 2022. arXiv:2201.06137.
- 14 Chungjae Lee, Kevin Dalmeijer, Pascal Van Hentenryck, and Peibo Zhang. Optimizing Autonomous Transfer Hub Networks: Quantifying the Potential Impact of Self-Driving Trucks. *arXiv:2305.03119*, 2023. arXiv:2305.03119.
- 15 Luke Marshall, Natasha Boland, Martin Savelsbergh, and Mike Hewitt. Interval-Based Dynamic Discretization Discovery for Solving the Continuous-Time Service Network Design Problem. *Transportation Science*, 55(1):29–51, 2021. doi:10.1287/trsc.2020.0994.
- 16 Clair E. Miller, Albert W. Tucker, and Richard A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM*, 7(4):326–329, 1960. doi:10.1145/321043.321046.
- 17 OpenStreetMap. Planet dump retrieved from <https://planet.osm.org>, 2021. URL: <https://www.openstreetmap.org>.
- 18 PACE. *Partnership for an Advanced Computing Environment (PACE)*, 2017. URL: <http://www.pace.gatech.edu>.
- 19 Laurent Perron and Vincent Furnon. Google OR-Tools v9.6. <https://developers.google.com/optimization/>, 2023.
- 20 A. Alan B. Pritsker, Lawrence J. Waiters, and Wolfe Philip M. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science*, 16(1):93–108, 1969. doi:10.1287/mnsc.16.1.93.

- 21 Roland Berger. Shifting up a gear – Automation, electrification and digitalization in the trucking industry. <https://www.rolandberger.com/>, 2018. URL: [https://www.rolandberger.com/publications/publication\\_pdf/roland\\_berger\\_trucking\\_industry.pdf](https://www.rolandberger.com/publications/publication_pdf/roland_berger_trucking_industry.pdf).
- 22 Ryder System, Inc. and Socially Aware Mobility Lab. The Impact of Autonomous Trucking: A Case-Study of Ryder’s Dedicated Transportation Network. *Ryder Newsroom*, 2021. URL: <https://newsroom.ryder.com/news/news-details/2021/Ryder-Teams-Up-with-Georgia-Tech-for-Industrys-First-Data-Driven-Study-on-Impact-of-Autonomous-Trucking/>.
- 23 Yannick Oskar Scherr, Mike Hewitt, Bruno Albert Neumann-Saavedra, and Dirk Christian Mattfeld. Dynamic discretization discovery for the service network design problem with mixed autonomous fleets. *Transportation Research Part B: Methodological*, 141:164–195, 2020. doi:10.1016/j.trb.2020.09.009.
- 24 Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Explaining the cumulative propagator. *Constraints*, 16:250–282, 2011. doi:10.1007/s10601-010-9103-2.
- 25 Mohsen Shahandasht, Binaya Pudasaini, and Sean Logan McCauley. Autonomous Vehicles and Freight Transportation Analysis. Technical report, The University of Texas at Arlington, 2019. doi:10.13140/RG.2.2.28484.78726.
- 26 TechCrunch. Aurora and Volvo partner to bring autonomous long-haul trucks to North America. <https://www.techcrunch.com/>, 2021. URL: <https://techcrunch.com/2021/03/30/aurora-and-volvo-partner-to-bring-autonomous-long-haul-trucks-to-north-america/>.
- 27 Transport Topics. Navistar, TuSimple Partner to Launch Self-Driving Trucks in 2024. <https://www.ttnews.com/>, 2020. URL: <https://www.ttnews.com/articles/navistar-tusimple-partner-launch-self-driving-trucks-2024>.
- 28 Steve Viscelli. Driverless? Autonomous Trucks and the Future of the American Trucker. Center for Labor Research and Education, University of California, Berkeley, and Working Partnerships USA, 2018. URL: <http://driverlessreport.org/>.
- 29 Duc Minh Vu, Mike Hewitt, Natasha Boland, and Martin Savelsbergh. Dynamic Discretization Discovery for Solving the Time-Dependent Traveling Salesman Problem with Time Windows. *Transportation Science*, 54(3):703–720, 2020. doi:10.1287/trsc.2019.0911.
- 30 Haotian Wu, Ian Herszterg, Martin Savelsbergh, and Yixiao Huang. Service Network Design for Same-Day Delivery with Hub Capacity Constraints. *Transportation Science*, 57(1):273–287, 2023. doi:10.1287/trsc.2022.1155.





# A New Approach to Finding $2 \times n$ Partially Spatially Balanced Latin Rectangles

Renee Mirka ✉

Cornell University, Ithaca, NY, USA

Laura Greenstreet ✉

Cornell University, Ithaca, NY, USA

Marc Grimson ✉

Cornell University, Ithaca, NY, USA

Carla P. Gomes ✉

Cornell University, Ithaca, NY, USA

---

## Abstract

Partially spatially balanced Latin rectangles are combinatorial structures that are important for experimental design. However, it is computationally challenging to find even small optimally balanced rectangles, where previous work has not been able to prove optimality for any rectangle with a dimension above size 11. Here we introduce a graph-based encoding for the  $2 \times n$  case based on finding the minimum-cost clique of size  $n$ . This encoding inspires a new mixed-integer programming (MIP) formulation, which finds exact solutions for the  $2 \times 12$  and  $2 \times 13$  cases and provides improved bounds up to  $n = 20$ . Compared to three other methods, the new formulation establishes the best lower bound in all cases and establishes the best upper bound in five out of seven cases.

**2012 ACM Subject Classification** Applied computing → Operations research

**Keywords and phrases** Spatially balanced Latin squares, partially spatially balanced Latin rectangles, minimum edge weight clique, combinatorial optimization, mixed integer programming, imbalance, cliques

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.47

**Category** Short Paper

**Funding** This material is based upon work partially supported by the National Science Foundation under Awards CCF-1522054 and CCF-2007009 and by AFOSR DURIP grant FA9550-21-1-0316, AFOSR MURI grant FA9550-18-1-0136, and AFOSR grant FA9550-20-1-0421.

## 1 Introduction

Latin squares and rectangles are combinatorial objects represented by  $n \times n$  or  $k \times n$  grids with entries assigned from  $\{1, 2, \dots, n\}$  such that no entry is repeated in a row or column. They are important structures for experimental design. For example, in agronomic field experiments, experts are interested in applying  $n$  treatments consisting of  $n$  fertilizers in different orderings, which can be achieved by designing the treatment sequences following a Latin square. Arbitrary Latin squares are not hard to generate. However, geometric imbalance due to some treatments occurring closer together more frequently can bias experimental results [21]. This motivates the use of *spatially balanced* Latin squares and rectangles which require additional structure to capture the notion of distance between any two treatments in the square or rectangle and are more computationally challenging to construct.

A large body of previous work has focused on spatially balanced Latin squares, including introducing streamlining constraints [8], using stochastic optimization [11, 10, 12, 6], and applying local search methods [22]. While spatially balanced Latin squares have been extensively studied [9, 16, 17, 21], there has been much less work on spatially balanced Latin



© Renee Mirka, Laura Greenstreet, Marc Grimson, and Carla P. Gomes;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 47; pp. 47:1–47:11

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

rectangles [4], despite rectangular conditions occurring more frequently in practice. For example, in the case of fertilizer treatments, the number of treatments is often less than the number of fertilizers resulting in a rectangular structure.

The main previous work investigating spatially balanced Latin rectangles established the non-existence of perfectly balanced Latin rectangles for an infinite family of sizes and shifted focus to constructing *partially* spatially balanced Latin rectangles (PSBLRs) [4]. Diaz et al. introduce and experimentally compare three approaches to generating PSBLRs: an assignment-based mixed-integer program (MIP), a constraint satisfaction program (CP), and a random-restart hill climbing local search. Using these approaches, Diaz et al. were able to find the provably optimal imbalance for rectangles up to a size of  $2 \times 11$  and provide bounds up to  $12 \times 12$ .

In this work, we focus on the  $2 \times n$  case of constructing PSBLRs, introducing a new graph-based encoding based on a reduction to a minimum-edge weight clique problem. The maximum/minimum edge-weight clique problem (MEWC) is a generalization of the classic max clique problem, where given an input graph  $G = (V, E)$  instead of finding the largest complete subgraph the goal is to find the subgraph with the largest/smallest sum of weighted edges. The MEWC problem has many applications including in experimental design [3], molecular biology [20], and materials discovery [1]. Multiple approaches have been developed for MEWC problems including linear and quadratic mixed-integer programs [5, 7, 18], branch-and-cut [5, 15, 19], and heuristic methods [2, 14]. Several benchmarks have been developed for MEWC, though less than half the instances have been solved to optimality [13].

For our experiments, we use a straight-forward MIP formulation of MEWC to emphasize the benefits of the reduction instead of advanced techniques developed for MEWC. This new MIP formulation for the problem of finding  $2 \times n$  PSBLRs finds optimal solutions for the previously unsolved cases of  $n = 12$  and  $13$  and provides new bounds for  $n = 14 - 20$ . Further, we demonstrate the new clique-based MIP formulation outperforms the previous assignment-based MIP formulation and A\* search and finds comparable solutions to local search while providing lower bounds.

## 2 Preliminaries

► **Definition 1** (Latin Rectangle). *Let  $k$  and  $n$  be positive integers with  $k \leq n$  and  $R$  be a  $k \times n$  matrix. Then  $R$  is a Latin rectangle if every entry of  $R$  contains a number in  $[n] = \{1, 2, \dots, n\}$  and no number is repeated in any row or column.*

In order to discuss partially spatially balanced Latin rectangles, we first introduce the notion of distance between two symbols in a Latin rectangle and imbalance:

► **Definition 2** (Imbalance). *For two symbols  $u, v \in [n]$  and  $i \leq k$ , the distance between  $u$  and  $v$  in row  $i$ , denoted  $d_i(u, v)$ , is the absolute value of the difference of the indices of the positions of  $u$  and  $v$  in row  $i$ . The overall distance between  $u$  and  $v$  is then  $d(u, v) = \sum_{i \leq k} d_i(u, v)$ .*

*The spatial imbalance of a Latin rectangle,  $R$ , is defined by*

$$\mathbb{I}(R) = \sum_{i,j} \left| d(i, j) - \frac{k(n+1)}{3} \right|.$$

► **Definition 3** (Spatially Balanced Latin Rectangle). *We say that a  $k \times n$  rectangle  $R$  is spatially balanced if all distances  $d(u, v)$  are the same. For brevity, we denote this as SBLR( $k, n$ ).*

Note that in this case, Diaz et al. show that each distance must be exactly  $\frac{k(n+1)}{3}$  and thus the imbalance of the rectangle is 0; see [4] for a short proof of this proposition.

► **Proposition 4.** *If there exists a solution for  $SBLR(k, n)$  then the distance between any pair of symbols is equal to  $k(n + 1)/3$ .*

As a corollary,  $k \equiv_3 0$  or  $n \equiv_2 2$  are necessary conditions for the existence of  $SBLR(k, n)$  as  $k(n + 1) \equiv_3 0$  must hold. Diaz et al. have further shown the non-existence of SBLRs of size  $2 \times n$  for  $n \neq 2$  and  $3 \times n$  for  $n \neq 3$ , as well as experimentally demonstrated that no perfectly spatially balanced rectangles with  $k \neq n$  exist up to size  $7 \times 7$  [4]. In practice, it is useful to minimize the imbalance even if it is not possible to reduce it to zero. This motivates the definition of the object of our study- partially spatially balanced Latin rectangles (PSBLRs):

► **Definition 5 (Partially Spatially Balanced Latin Rectangle).** *A  $k \times n$  Latin rectangle  $R$  is partially spatially balanced if  $\mathbb{I}(R)$  is minimized. In particular,  $R$  is partially spatially balanced if for any  $k \times n$  Latin rectangle  $R'$ ,  $\mathbb{I}(R) \leq \mathbb{I}(R')$ .*

### 3 Graph Encoding

In this section, we describe a new graph-based encoding for the problem of constructing PSBLRs. For the  $2 \times n$  Latin rectangles we consider, we will always assume the first row is ordered and given by  $1 \ 2 \ \dots \ n$ , as any other solution can be transformed into a solution in this form through relabelling symbols. This reduces the problem to selecting a single imbalance-minimizing derangement, or permutation with no fixed points, for the second row of the rectangle. It also simplifies the computation of the distance between any two symbols  $u$  and  $v$ . Particularly, for  $1 \leq u < v \leq n$ , we have  $d(u, v) = v - u + |i_v - i_u|$  where  $i_v, i_u$  are the indices of  $v, u$  in the second row, respectively. For fixed  $u, v$  and  $n$ ,  $d(u, v)$  is solely determined by the choices for  $i_u$  and  $i_v$ . We exploit this through the construction of a graph which encodes how these choices affect the imbalance.

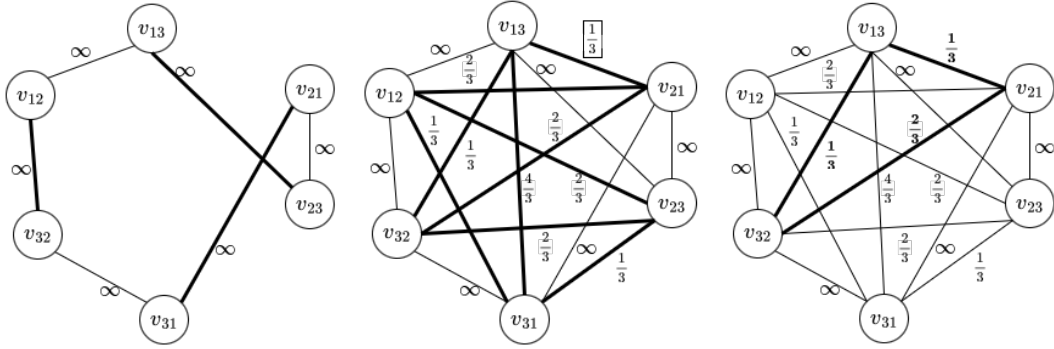
The graph encoding is as follows. For a given  $n$ , construct a complete graph  $G = (V, E)$  with  $V = \{v_{ij} : 1 \leq i, j \leq n, i \neq j\}$  and  $E = \{(u, v) : u, v \in V\}$ . Intuitively, a vertex  $v_{ij}$  represents a Latin rectangle where  $j$  is the index of  $i$  in the second row. We include a cost on each edge  $(v_{st}, v_{qr})$  which represents how much is contributed to the imbalance from  $d(s, q)$  if  $t$  and  $r$  are the indices of  $s$  and  $q$  in the second row, respectively. In particular, for  $e = (v_{st}, v_{qr}) \in E$  with  $s \neq q$  and  $t \neq r$ , let

$$\begin{aligned} c_e &= \left| d(s, q) - \frac{2(n+1)}{3} \right| \\ &= \left| |s - q| + |i_s - i_q| - \frac{2(n+1)}{3} \right| \\ &= \left| |s - q| + |t - r| - \frac{2(n+1)}{3} \right|. \end{aligned}$$

If  $s = q$  or  $t = r$ , the cost on the edge is  $\infty$ , since the assignments these vertices represent are not valid for a Latin rectangle. See Figure 1 for an example when  $n = 3$ .

To complete the problem encoding, we need to discern how to recover a PSBLR from this complete graph. The key observation is that every  $n$ -clique where all edges have finite cost corresponds to a valid Latin rectangle. The finiteness of the costs ensures that the assignment rules of the Latin rectangle are obeyed. Furthermore, the sum of the costs on the edges of an  $n$ -clique is exactly equal to the imbalance of its corresponding Latin rectangle. As such, to find a PSBLR, it suffices to find an  $n$ -clique whose sum of edge costs is minimal. Figure 1 illustrates an optimal clique corresponding to an optimal Latin rectangle for  $n = 3$ :

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{bmatrix}$$



**Figure 1** (Left) The graph encoding after including all edges with infinite costs between vertices  $v_{sq}$  and  $v_{sr}$  for  $s = 1, 2, 3$  (thin edges) and between vertices  $v_{sq}$  and  $v_{r_q}$  for  $q = 1, 2, 3$  (bold edges). (Center) The graph encoding after including all remaining edges with finite costs (bold edges) between the remaining pairs of vertices. The boxed value is calculated as  $|(2 - 1) + (3 - 1) - 8/3| = 1/3$ . (Right) A 3-clique (bold edges) in the graph encoding when  $n = 3$  corresponding to a  $2 \times 3$  PSBLR.

#### 4 MIP Formulation

Now that we have reduced the problem of finding a  $2 \times n$  PSBLR to finding a minimum-cost  $n$ -clique, we can model the problem through a new integer program:

$$\begin{aligned}
 & \text{minimize } \sum_{e \in E} c_e x_e \\
 & \text{subject to } \sum_{v \in V} z_v = n, & (1) \\
 & x_e \geq z_v + z_u - 1, & \forall e = (u, v) \in E, & (2) \\
 & x_e \in \{0, 1\}, & \forall e \in E, \\
 & z_v \in \{0, 1\} & \forall v \in V.
 \end{aligned}$$

In this model, we have binary variables  $z_v$  and  $x_e$  for each vertex  $v$  and edge  $e$  in the graph representing whether the vertex or edge is included in the  $n$ -clique. While (1) guarantees that we select  $n$  vertices, (2) requires that any edges between two selected vertices are included in the cost; together these ensure the model selects an  $n$ -clique.

Note that the size of this model is polynomial in  $n$  with  $n(n-1) + n(n-1)(n(n-1)-1)/2$  variables and  $n(n-1)(n(n-1)-1)/2 + 1$  constraints. Furthermore, by construction, an optimal solution will be a PSBLR. However, the presence of so many binary variables makes finding the optimal solution a cumbersome computational task.

The following observation allows us to partially relax the integer program and significantly reduce the number of binary variables:

**► Observation 6.** For each edge  $e \in E$ ,  $x_e$  will be 0 or 1 if  $z_v$  is binary for all  $v \in V$ . In other words, we can relax the restriction that  $x_e$  is binary for all  $e \in E$  by replacing it with  $x_e \geq 0$  for all  $e \in E$  and still guarantee a binary solution.

The relaxed mixed integer program only requires  $n(n-1)$  binary variables, one for each vertex, where the remaining edge variables are continuous.

## 5 Methods

We compared the min-cost  $n$ -clique MIP formulation, hereafter referred to as the clique formulation, to an assignment-based MIP formulation, A\* search, and a local search approach.

In the assignment based MIP formulation, we again assume the first row is fixed to 1 2 ...  $n$ . Thus, we only need to introduce binary variables indicating whether value  $i$  occurs in position  $j$  of the second row,  $i, j \in \{1, 2, \dots, n\} \equiv [n], i \neq j$ . Note that these exactly correspond to the vertex variables in the clique formulation,  $z_v = z_{v_{ij}}$  for  $v \in \{v_{ij} : i, j \in [n], i \neq j\}$ . As the imbalance is dependent on pairwise positions in the second row, we also introduce binary variables that are 1 if a pair of variables is included in the solution. Note that these correspond exactly to the edge variables in the clique formulation,  $x_e$  for  $e \in \{(u, v) : u, v \in V\}$ , where the pairwise cost corresponds to  $c_e$ . Finally, we introduce constraints ensuring each value appears exactly once in the second row (3) and each position in the second row is assigned exactly one value (4):

$$\begin{aligned} & \text{minimize } \sum_{e \in E} c_e x_e \\ & \text{subject to } \sum_{j \in [n], j \neq i} z_{v_{ij}} = 1, & \forall i \in [n] & \quad (3) \\ & \sum_{i \in [n], i \neq j} z_{v_{ij}} = 1, & \forall j \in [n] & \quad (4) \\ & x_e \geq z_v + z_u - 1, & \forall e = (u, v) \in E, \\ & x_e \in \{0, 1\}, & \forall e \in E, \\ & z_v \in \{0, 1\} & \forall v \in V. \end{aligned}$$

Similar to the clique formulation, we can partially relax the integer program by allowing the  $x_e$  variables to assume continuous values, reducing the number of binary variables to  $n(n-1)$ .

Both MIP formulations were implemented in Gurobi and CPLEX using their respective Python APIs. As previous work has established optimal solutions up to  $n = 11$ , we replicated previous results and further tested instances with  $n = 12 - 20$ . For each  $n$ , both models were tested for 6 hours running on a Intel Xeon 6154 processor and allowed 32GB of memory in three configurations: a single instance running on a single thread, a single instance running on 32 threads, and 8 instances running on a total of 32 threads.

For A\* search, we again fix the first row and only consider positions in the second row. Starting with an empty second row, we add all valid placements of the first digit to a priority queue, assigning each node a cost based on the partial imbalance of the filled digits and an admissible heuristic for the imbalance due to the unplaced digits. We repeatedly evaluate the first node in the queue until we reach a node with all digits filled which is guaranteed to be an optimal solution. As a heuristic, for each unplaced digit we calculated the minimum increase to the imbalance that would result from placing it in one of the remaining positions, accounting only for interactions with already placed digits. We then summed these values across all unplaced digits.

A\* search was implemented in C++ and was also tested for 6 hours each on instances of size  $n = 12 - 20$  on an Intel Xeon 6154 processor. As A\* search is memory intensive, runs were allocated 128GB of memory, compared to 32GB for the other methods. If A\* search does not complete, the value of the node at the front of the queue can be used as a lower bound on the solution value. However if the algorithm does not run to completion, no feasible solutions are found, resulting in no upper bound on the imbalance.

We implemented local search using a random walk from a random initial derangement, where at each step the position of two random digits were swapped, maintaining feasibility. The random walk was implemented in C++ and tested for 6 hours each on instances of size  $n = 12 - 20$  on an Intel Xeon 6154 processor and allocated 32GB of memory. Unlike the other methods, a random walk cannot prove optimality or provide a lower bound on the solution.

In addition to testing methods to bound the optimal solution, we performed several tests to better characterize the solution space. First, for  $n = 2 - 10$  we brute force computed all solutions to determine the number of optimal solutions. We were inspired to characterize the number of solutions due to the observation that mirroring either the first or second row results in a rectangle with the same imbalance. For example if we have a solution  $R$ , we can construct a rectangle  $R'$  with the second row mirrored by setting  $R'_{2i} = n + 1 - R_{2i}$  for  $i \in [n]$ . This results in the same imbalance:

$$\begin{aligned} \mathbb{I}(R') &= \sum_{i,j} \left| |R_{1i} - R_{1j}| - |(n+1 - R_{2i}) - (n+1 - R_{2j})| - \frac{2(n+1)}{3} \right| \\ &= \sum_{i,j} \left| |R_{1i} - R_{1j}| - |R_{2i} - R_{2j}| - \frac{2(n+1)}{3} \right| \\ &= \mathbb{I}(R) \end{aligned}$$

However, it is not guaranteed that  $R'$  is a Latin rectangle, as the rearrangement may not respect the column constraint. Similarly, we can create a rectangle,  $R^\dagger$ , with the first row mirrored by setting  $R^\dagger_{1i} = n + 1 - R_{1i}$  for  $i \in [n]$ . While this does not result in a rectangle with the first row ordered as  $1 \ 2 \ \dots \ n$ , we can relabel the variables so the first row goes from  $1 \ 2 \ \dots \ n$  which results in the second row having the labels  $R^\dagger_{2i} = R_{2(n+1-i)}$ . While this could potentially allow up to four solutions from a single solution, there is no guarantee that these solutions satisfy the column constraints. As multiple solutions are not guaranteed, we cannot use methods like streamlining. However, the presence of multiple solutions may make it difficult for the MIP formulations to prove optimality.

Further, we sought to characterize the distribution of feasible solutions. For  $n = 2 - 10$  we were able to compute the distribution exactly. For  $n = 11 - 16$ , we randomly sampled a million solutions to approximate the distribution. The distribution of solutions impacts all four methods, where having many solutions with nearly optimal imbalance will benefit a random walk, while it will make it difficult for the MIP formulation and A\* search to prove optimality.

## 6 Results

While CPLEX had better runtimes up to  $n = 10$ , only Gurobi was able to prove optimality for  $n = 12$  and 13 and Gurobi consistently found better bounds for  $n = 14 - 20$  (see Tables 3-5). Thus, for the remainder of the results we report the performance of the Gurobi model for both MIP formulations. In general, running the model using a single instance and a single thread performed better than running a single instance with multiple threads or multiple instances, with the only notable exception being for  $n = 20$ , where running with multiple instances found the smallest upper bound of 816 (See Tables 6-7).

While both MIP formulations were able to prove optimality for  $n = 12$ , only the clique formulation was able to prove optimality for  $n = 13$  within the 6 hours allotted (See Table 8). While local search is not able to prove optimality, it found the optimal solution for both

$n = 12$  and  $13$ . For all  $n = 2 - 20$ , the clique formulation resulted in the largest lower bound (See Table 1). The results were more varied for the upper bound, where the random walk found the lowest upper bound for  $n = 14$  and  $17$ , the clique found the lowest upper bound for  $n = 18, 19$ , and  $20$ , and both methods found the same upper bound for  $n = 15$  and  $16$  (See Table 2). For  $n = 16$ , the assignment formulation matched the upper bound found by the clique formulation and random walk.

A\* was only able to run to completion for  $n = 12$ . For  $n = 13 - 15$ , A\* ran out memory, despite being allotted 128GB where all other methods used under 32GB. For  $n > 15$ , A\* was only able to reach configurations with at most seven fixed values, resulting in a weak lower bound on the optimal solution.

■ **Table 1** Lower bound by method for  $n = 12 - 20$ . Bold values indicate the best bound, i.e. the largest lower bound, for each  $n$ . All methods were allotted 6 hours.

	$n$								
	12	13	14	15	16	17	18	19	20
Clique MIP	<b>168</b>	<b>218.66</b>	<b>191</b>	<b>252.66</b>	<b>273.66</b>	<b>252</b>	<b>402.66</b>	<b>465.33</b>	<b>155</b>
Assignment MIP	<b>168</b>	<b>218.66</b>	131	141.33	133	91	121	138.66	47
A* search	<b>168</b>	191	166	169	164.66	134	159.33	158.66	106
Random walk	-	-	-	-	-	-	-	-	-

■ **Table 2** Upper bound by method for  $n = 12 - 20$ . Bold values indicate the best bound, i.e. the smallest upper bound, for each  $n$ . All methods were allotted of 6 hours.

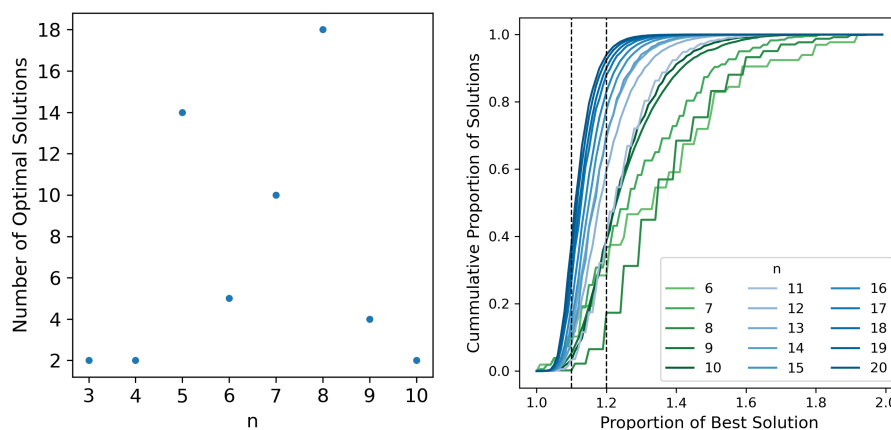
	$n$								
	12	13	14	15	16	17	18	19	20
Clique MIP	<b>168</b>	<b>218.66</b>	272	<b>345.33</b>	<b>427.33</b>	522	<b>617.33</b>	<b>732</b>	<b>816</b>
Assignment MIP	<b>168</b>	<b>218.66</b>	272	346	<b>427.33</b>	526	621.33	<b>732</b>	886
A* search	<b>168</b>	-	-	-	-	-	-	-	-
Random walk	<b>168</b>	<b>218.66</b>	<b>268</b>	<b>345.33</b>	<b>427.33</b>	<b>508</b>	623.33	742.66	866

For the exploratory tests, for all  $n = 2 - 10$  there are multiple optimal solutions (See Figure 2). The number of solutions did not show any obvious patterns, such as a monotonic increase or alternation between odd and even digits. For  $n = 6 - 10$  where we were able to compute the distribution of solutions exactly, up to 10% of solutions were within 10% of optimal and as many as 40% of solutions were within 20% of optimal (See Figure 2). For  $n = 11 - 20$  we looked at the distribution over a million random solutions instead of the full solution space, so the best solution likely does not represent the optimal solution. However, the trend of a large proportion of the cumulative distribution being within 20% of the best solution continued, where at the high-end of the range over 40% of solutions were within 10% of the best solution and over 90% were within 20% of the best solution.

## 7 Discussion

Both the clique and assignment formulation were able to prove optimality for the previously unsolved case of  $n = 12$  and the clique formulation was further able to prove optimality for  $n = 13$ . For  $n = 14 - 20$ , the clique formulation consistently found the best lower-bounds, outperforming the assignment formulation and A\* in all seven cases. While the clique formulation and a random walk were comparable in establishing upper bounds, finding the





■ **Figure 2** (Left) Number of optimal solutions by  $n$ . (Right) Cumulative distribution of solutions as proportion of the best solution. For  $n = 6 - 10$  (green), distributions represent all solutions. For  $n = 11 - 20$  (blue), distributions are over a million random solutions. Dashed lines indicate solutions that are 10% and 20% greater than the best solution respectively.

best upper bound in five and four cases respectively, the clique formulation appeared to perform better for large  $n$ , establishing the best upper bound for  $n = 18 - 20$ . Thus, the clique formulation outperformed each of the other three methods individually, as well as performed better than using a combination of other methods, such as using  $A^*$  to establish lower bounds and a random walk to establish upper bounds.

While the clique formulation outperformed the other methods, no method was able to prove the optimality of a solution above  $n = 13$ . The exploratory results shed some light on why it is difficult to find even relatively small PSBLRs. First, the potential for mirror solutions as well as the presence of multiple solutions for all  $n = 3 - 10$  make it challenging for the MIP formulations to prove optimality, as potentially many branches of the branch-and-bound tree include optimal solutions and must be extensively explored before they can be pruned. Further, while there are likely multiple solutions, the possibility of a mirror rectangle violating a column constraint means we cannot guarantee multiple solutions and use techniques like streamlining constraints to guide the model towards a single optimal solution.

The large portion of the cumulative distribution close to the optimal solution further hampers the MIP formulations, making it difficult to prune branches that include near-optimal solutions. This large number of near-optimal solutions also hampers  $A^*$  search, where a very tight heuristic is needed to allow for any significant amount of pruning, where experimentally we found that most nodes could generally only be pruned once all but one or two values had been assigned. While a random walk would benefit from multiple optimal solutions, a random walk or other local-search based methods cannot establish optimality.

## 8 Conclusion

Spatially and partially spatially balanced Latin squares and rectangles are important combinatorial structures used for experimental design. While spatially balanced Latin squares have been extensively studied, relatively little work has considered spatially balanced Latin rectangles, despite them occurring more frequently in practice.

We introduce a new graph-based encoding for a  $2 \times n$  Latin rectangle which inspires a new MIP formulation based on the MEWC problem. This new formulation found optimal solutions previously unsolved  $2 \times 12$  and  $2 \times 13$  cases and outperforms an assignment-based MIP formulation, A\* search, and a random-walk based local-search, establishing improved bounds up to  $n = 20$ . Given the success of our straight-forward MIP implementation for MEWC, a direction for future work is to explore whether more advanced MEWC methods are able to find optimal PSBLRs for  $n > 13$ .

Further, our exploratory results help characterize what make finding PSBLRs computationally challenging. The potential, but not guarantee, of multiple optimal solutions makes it difficult for mathematical programming methods to establish optimality and prevents the use of standard methods for handling multiple solutions, such as streamlining. The large number of near-optimal solutions makes it difficult for informed search-based methods to prune any significant amount of solutions, requiring near brute-force exploration of the search space. While simple to encode, this problem is quite challenging, making it an ideal benchmark for future search and optimization methods and we encourage further exploration of the problem.

---

## References

---

- 1 Luis A Agapito, Marco Fornari, Davide Ceresoli, Andrea Ferretti, Stefano Curtarolo, and Marco Buongiorno Nardelli. Accurate tight-binding hamiltonians for two-dimensional and layered materials. *Physical Review B*, 93(12):125137, 2016.
- 2 Bahram Alidaee, Fred Glover, Gary Kochenberger, and Haibo Wang. Solving the maximum edge weight clique problem via unconstrained quadratic programming. *European Journal of Operational Research*, 181(2):592–597, 2007.
- 3 Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4):1054–1068, 1986.
- 4 Mateo Díaz, Ronan Le Bras, and Carla Gomes. In search of balance: The challenge of generating balanced latin rectangles. In *Proceedings of Integration of AI and OR Techniques in Constraint Programming: 14th International Conference*, pages 68–76. Springer, 2017.
- 5 G Dijkhuizen and Ulrich Faigle. A cutting-plane approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 69(1):121–130, 1993.
- 6 Stefano Ermon, Carla Gomes, Ashish Sabharwal, and Bart Selman. Low-density parity constraints for hashing-based discrete integration. In *Proceedings of International Conference on Machine Learning*, pages 271–279. PMLR, 2014.
- 7 U Faigle, R Garbe, K Heerink, and B Spieker. Lp-relaxations for the edge-weighted subclique problem. In *Operations Research'93: Extended Abstracts of the 18th Symposium on Operations Research held at the University of Cologne September 1–3, 1993*, pages 157–160. Springer, 1994.
- 8 Carla Gomes and Meinolf Sellmann. Streamlined constraint reasoning. In *Proceedings of Principles and Practice of Constraint Programming: 10th International Conference*, pages 274–289. Springer, 2004.
- 9 Carla Gomes, Meinolf Sellmann, Cindy Van Es, and Harold Van Es. The challenge of generating spatially balanced scientific experiment designs. In *Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 387–394. Springer, 2004.
- 10 Carla P Gomes, Joerg Hoffmann, Ashish Sabharwal, and Bart Selman. Short xors for model counting: from theory to practice. In *Proceedings of Theory and Applications of Satisfiability Testing–SAT 2007: 10th International Conference*, pages 100–106. Springer, 2007.

- 11 Carla P Gomes, Ashish Sabharwal, and Bart Selman. Model counting: A new strategy for obtaining good bounds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 10, pages 1597538–1597548, 2006.
- 12 Carla P Gomes, Willem Jan van Hoeve, Ashish Sabharwal, and Bart Selman. Counting csp solutions using generalized xor constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 204–209, 2007.
- 13 Seyedmohammadhossein Hosseinian, Dalila BMM Fontes, Sergiy Butenko, Marco Buongiorno Nardelli, Marco Fornari, and Stefano Curtarolo. The maximum edge weight clique problem: formulations and solution approaches. *Optimization Methods and Applications: In Honor of Ivan V. Sergienko's 80th Birthday*, pages 217–237, 2017.
- 14 Seyedmohammadhossein Hosseinian, DBMM Fontes, and Sergiy Butenko. A quadratic approach to the maximum edge weight clique problem. In *XIII Global Optimization Workshop GOW*, volume 16, pages 125–128, 2016.
- 15 Marcel Hunting, Ulrich Faigle, and Walter Kern. A lagrangian relaxation approach to the edge-weighted clique problem. *European Journal of Operational Research*, 131(1):119–131, 2001.
- 16 Ronan Le Bras, Carla Gomes, and Bart Selman. From streamlined combinatorial search to efficient constructive procedures. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26-1, pages 499–506, 2012.
- 17 Ronan Le Bras, Andrew Perrault, and Carla P Gomes. Polynomial time construction for spatially balanced latin squares. Technical report, Cornell University, 2012.
- 18 Anuj Mehrotra. Cardinality constrained boolean quadratic polytope. *Discrete Applied Mathematics*, 79(1-3):137–154, 1997.
- 19 Michael M Sørensen. New facets and a branch-and-cut algorithm for the weighted clique problem. *European Journal of Operational Research*, 154(1):57–70, 2004.
- 20 Etsuji Tomita, Tatsuya Akutsu, and Tsutomu Matsunaga. *Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics*. IntechOpen, 2011.
- 21 HM Van Es and CL Van Es. Spatial nature of randomization and its effect on the outcome of field experiments. *Agronomy Journal*, 85(2):420–428, 1993.
- 22 Pascal Van Hentenryck and Laurent Michel. Differentiable invariants. In *Proceedings of Principles and Practice of Constraint Programming: 12th International Conference*, pages 604–619. Springer, 2006.

## A Supplementary Results

■ **Table 3** Runtime in seconds for single instances of the clique formulation run in Gurobi and CPLEX using a single thread for  $n = 8 - 13$ . The dash indicates that the model was not able to prove optimality in 6 hours (21,600 s). For all  $n$  less than 8, both models ran in under one second.

	n					
	8	9	10	11	12	13
CPLEX	<b>1.10</b>	<b>4.48</b>	<b>28.76</b>	675.16	-	-
Gurobi	1.15	4.99	51.13	<b>171.22</b>	<b>614.22</b>	<b>4750.96</b>

	n						
	14	15	16	17	18	19	20
CPLEX	132	119.29	97.54	88	47.79	40.06	45.06
Gurobi	<b>191</b>	<b>252.66</b>	<b>273.66</b>	<b>252</b>	<b>402.66</b>	<b>465.33</b>	<b>155</b>

■ **Table 4** Comparison of the greatest lower bound found by single instances of the clique formulation run in Gurobi and CPLEX using a single thread for 6 hours for  $n = 14 - 20$ .

	n						
	14	15	16	17	18	19	20
CPLEX	<b>272</b>	354	437.33	526	631.33	762	878
Gurobi	<b>272</b>	<b>345.33</b>	<b>427.33</b>	<b>522</b>	<b>617.33</b>	<b>732</b>	<b>876</b>

■ **Table 5** Comparison of the smallest upper bound found by single instances of the clique formulation run in Gurobi and CPLEX using a single thread for 6 hours for  $n = 14 - 20$ .

		n						
Instances	Threads/Inst.	14	15	16	17	18	19	20
1	1	191	<b>252.66</b>	273.66	<b>252</b>	<b>402.66</b>	<b>465.33</b>	<b>155</b>
8	1	159.69	185.28	203.11	198.56	-	389.50	92.5
1	32	<b>201.57</b>	249.79	<b>273.82</b>	240.90	185.75	371.95	28

■ **Table 6** Comparison of the greatest lower bound found by three Gurobi configurations run for 6 hours for  $n = 14 - 20$ . The dashed cell indicates a run that failed due to numerical instability.

		n						
Instances	Threads/Inst.	14	15	16	17	18	19	20
1	1	272	<b>345.33</b>	<b>427.33</b>	<b>522</b>	<b>617.33</b>	<b>732</b>	886
8	1	<b>270</b>	358.66	442.66	530	-	747.33	<b>816</b>
1	32	276	347.33	430.66	<b>522</b>	631.33	766	900

■ **Table 7** Comparison of the smallest upper bound found by three Gurobi configurations run for 6 hours for  $n = 14 - 20$ . The dashed cell indicates a run that failed due to numerical instability.

	n					
	8	9	10	11	12	13
Assignment MIP	1.09	6.23	59.02	678.39	3790.28	-
Clique MIP	1.15	4.99	51.13	171.22	<b>614.22</b>	<b>8573.43</b>
A*	<b>0.07</b>	<b>0.78</b>	<b>8.72</b>	<b>98.70</b>	1299.66	-

■ **Table 8** Runtime in seconds for single instances of the clique and assignment formulations run in Gurobi using a single thread and A\* search for  $n = 8 - 13$ . The dash indicates that the model was not able to prove optimality in 6 hours (21,600 s). For all  $n$  less than 8, all models finished in under one second. The random walk is not included in this table as the method does not prove optimality.



# Proven Optimally-Balanced Latin Rectangles with SAT

Vaidyanathan Peruvemba Ramaswamy ✉ 🏠 

Algorithms and Complexity Group, TU Wien, Austria

Stefan Szeider ✉ 🏠 

Algorithms and Complexity Group, TU Wien, Austria

---

## Abstract

Motivated by applications from agronomic field experiments, Díaz, Le Bras, and Gomes [CPAIOR 2015] introduced Partially Balanced Latin Rectangles as a generalization of Spatially Balanced Latin Squares. They observed that the generation of Latin rectangles that are optimally balanced is a highly challenging computational problem. They computed, utilizing CSP and MIP encodings, Latin rectangles up to  $12 \times 12$ , some optimally balanced, some suboptimally balanced.

In this paper, we develop a SAT encoding for generating balanced Latin rectangles. We compare experimentally encoding variants. Our results indicate that SAT encodings perform competitively with the MIP encoding, in some cases better. In some cases we could find Latin rectangles that are more balanced than previously known ones. This finding is significant, as there are many arithmetic constraints involved. The SAT approach offers the advantage that we can certify that Latin rectangles are optimally balanced through DRAT proofs that can be verified independently.

**2012 ACM Subject Classification** Hardware → Theorem proving and SAT solving; Theory of computation → Constraint and logic programming; Mathematics of computing → Combinatoric problems; Software and its engineering → Constraints; Theory of computation → Integer programming; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Enumeration; Mathematics of computing → Solvers; Computing methodologies → Theorem proving algorithms; Applied computing → Agriculture

**Keywords and phrases** combinatorial design, SAT encodings, certified optimality, arithmetic constraints, spatially balanced Latin rectangles

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.48

**Category** Short Paper

**Supplementary Material** *Software (Script, Encodings and Models)*: <https://doi.org/10.5281/zenodo.8151905>

**Funding** Supported by the Austrian Science Funds (FWF) within the project P36420, and the Vienna Science and Technology Fund (WWTF) within the project ICT19-065.

**Acknowledgements** This work was carried out in part while the second author visited the Simons Institute for the Theory of Computing, University of Berkeley, within the program *Extended Reunion: Satisfiability*.

## 1 Introduction

A *Latin square* is an  $n \times n$  array filled with  $n$  different symbols, each occurring exactly once in each row and exactly once in each column. The notion goes back to the 18th century and Leonard Euler, who studied the problem of generating Latin squares. More recently, additional constraints have been added, making the combinatorial design problem even more challenging. An interesting additional constraint asks for a Latin square to be spatially balanced, i.e., any two symbols  $u, v$  have the same distance. The distance of  $u$  and  $v$  is



© Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 48; pp. 48:1–48:10

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

defined as the sum of their distances over all the rows. A polynomial-time algorithm is known that constructs for any given  $n$  such that  $2n + 1$  is prime a spatially balanced Latin square [10].

Díaz, Le Bras, and Gomes [3] introduced *spatially balanced Latin rectangles*, which are arrays with  $n$  columns and  $k$  rows filled with  $n$  symbols, such that no symbol occurs more than once in any row or column, and again the distance of any pair of symbols is the same. As it turned out that spatially balanced Latin rectangles only exist for a relatively restricted set of combinations of  $n$  and  $k$ , Díaz et al. defined a notation of *imbalance* and asked for Latin rectangles where the imbalance is minimal. In a spatially balanced  $k \times n$  Latin rectangle, the distance between any two symbols is  $k(n + 1)/3$ . Therefore, the imbalance of a pair of symbols  $i, j$  is naturally defined as  $\mathbb{I}(u, v) = |d(u, v) - k(n + 1)/3|$ . The *imbalance* of a Latin rectangle  $L$  is then  $\mathbb{I}(L) = \sum_{u < v} \mathbb{I}(u, v) = \sum_{u < v} |d(u, v) - k(n + 1)/3|$ . See Figure 1b for a sample working out of this quantity. Latin squares are of general importance for the design of agronomic experiments [9]. More specifically, however, the balanced version is essential to the design of bias-free agronomic experiments, as they avoid unintentional patterns introduced due to spatial auto-correlation [13].

Díaz et al. computed Latin rectangles up to  $12 \times 12$ , trying to minimize the total imbalance. They used CSP, local search, and MIP methods, with the latter being the most efficient. In some cases, they could obtain optimal imbalance; in other cases, upper bounds. They conclude that finding Latin rectangles with minimum imbalance seems to be a very challenging computational problem and suggest it as an ideal benchmark for different search and optimization approaches.

In this paper, we follow this insight and consider the research question of whether propositional satisfiability (SAT) solvers [4] are competitive in finding optimally balanced Latin rectangles. This question is particularly interesting as the definition of a partially balanced Latin rectangle entails many arithmetic constraints, including addition, subtraction, and absolute values, which adds to the challenge. A SAT approach that is competitive with MIP brings the added value of certification: one can certify that a Latin rectangle is optimally balanced through a DRAT proof that can be checked independently. This provides an additional layer of confidence to the results.

In brief, the SAT encoding developed by us represents the position of each symbol on a row by a set of propositional variables. The assignment of these variables is then propagated by means of auxiliary variables and clauses through the rest of the encoding. This propagation terminates at a set of variables which represent the total imbalance of the Latin rectangle expressed in unary. Bounding these variables effectively bounds the imbalance. Special care must be taken to encode the absolute value arithmetic and the nested summations with the auxiliary variables and clauses.

From our experiments, we observe that the SAT approach is able to replicate 44 out of the 63 upper bounds shown by the MIP approach and in 14 cases even find a tighter upper bound. From the infeasibility side as well, the SAT approach is comparable to the MIP approach and is able to confirm almost all the optimality results with the added bonus of producing DRAT proofs with some minor overhead.

## 2 Preliminaries

We denote the set of integers  $\{1, \dots, n\}$  by  $[n]$ . For positive integers  $k \leq n$ , a  $k \times n$  *Latin rectangle* is a matrix  $L$  with  $k$  rows and  $n$  columns with elements from  $[n]$  such that  $L(i, j) \neq L(i', j')$  whenever  $i = i' \wedge j \neq j'$  or  $i \neq i' \wedge j = j'$ .

Consider a  $k \times n$  Latin rectangle  $L$  and  $i, j \in [n]$ . The *row  $i$  distance* between  $u, v$ , if  $L(i, c_u) = u$  and  $L(i, c_v) = v$ , is

$$d_i(u, v) := |c_u - c_v|. \tag{1}$$

The *distance* between  $u$  and  $v$  in  $L$  is

$$d(u, v) := \sum_{i \in [k]} d_i(u, v), \tag{2}$$

and the *imbalance* of the pair  $u, v$  is

$$\mathbb{I}(u, v) := |d(u, v) - k(n + 1)/3|. \tag{3}$$

The imbalance of a Latin rectangle  $L$  is the sum of the pairwise imbalances, i.e.,

$$\mathbb{I}(L) := \sum_{u < v} \mathbb{I}(u, v). \tag{4}$$

We are interested in finding Latin rectangles with low imbalance. A Latin rectangle  $L$  is *optimally balanced* (or partially spatially balanced as Díaz et al. [3] call it) if  $\mathbb{I}(L)$  is minimum and *suboptimally balanced* otherwise. See Figure 1a for an example.

1	2	3	4	5
2	5	1	3	4
4	1	5	2	3

$u, v$	$d(u, v)$	$ d(u, v) - Z $	$u, v$	$d(u, v)$	$ d(u, v) - Z $
1, 2	5	1	2, 4	9	3
1, 3	6	0	2, 5	5	1
1, 4	6	0	3, 4	6	0
1, 5	6	0	3, 5	6	0
2, 3	5	1	4, 5	6	0

(a)  $3 \times 5$  optimally balanced Latin rectangle with imbalance 6. (b) Breakdown (for each pair) of the total imbalance, where  $Z = k(n + 1)/3 = 6$ . For illustration,  $d(1, 2) = d_1(1, 2) + d_2(1, 2) + d_3(1, 2)$  which is  $1 + 2 + 2 = 5$ .

■ **Figure 1** Example of an optimally balanced  $3 \times 5$  Latin rectangle.

### 3 Encoding

In this section, we describe the details of the SAT encoding. We describe the clauses that are conjuncted together to form the CNF formula  $\mathcal{F}(k, n, t)$ . This formula is satisfiable if and only if there exists a  $k \times n$  Latin rectangle  $L$  such that  $\mathbb{I}(L) \leq t$ . For the sake of convenience, we sometimes shorten  $\mathcal{F}(n, k, t)$  to  $\mathcal{F}$ , and use  $\Delta_n$  to denote all ordered pairs  $(u, v)$  such that  $u < v \in [n]$ . We also use  $Z := k(n + 1)/3$ , assuming that 3 divides  $k(n + 1)$ . We address the case of indivisibility later. Further, we make extensive use of higher-level cardinality constraints written as  $\{v_i \mid i \in [\ell]\}_{\leq k}$ ,  $\{v_i \mid i \in [\ell]\}_{=k}$ , or  $\{v_i \mid i \in \ell\}_{\geq k}$  to encode the condition that only at most  $k$ , exactly  $k$ , or at least  $k$  variables, respectively, from the set  $\{v_1, \dots, v_\ell\}$  are set to true. Each higher-level constraint can then be rewritten as a conjunction of several elementary clauses utilizing auxiliary variables.

The core variables in  $\mathcal{F}(n, k, t)$  are  $p(i, u, j)$  for  $u, j \in [n], i \in [k]$  which are true if and only if the column where element  $u$  appears in the  $i$ th row is  $j$ . The following cardinality constraint encodes the condition that each cell of the matrix contains exactly one element:

$$\bigwedge_{i \in [k], j \in [n]} \{p(i, u, j) \mid u \in [n]\}_{=1}$$



## 48:4 Proven Optimally-Balanced Latin Rectangles with SAT

The following cardinality constraints encode the requirement that each element appears exactly once within a row:

$$\bigwedge_{i \in [k], u \in [n]} \{p(i, u, j) \mid j \in [n]\}_{\geq 1} \wedge \{p(i, u, j) \mid j \in [n]\}_{\leq 1}$$

Lastly, the following cardinality constraint encodes the requirement that no element appears twice within the same column:

$$\bigwedge_{u, j \in [n]} \{p(i, u, j) \mid i \in [k]\}_{\leq 1}$$

Then, to capture Equation (1), we introduce the auxiliary variables  $c(i, u, v, d)$  which are true if  $d_i(u, v) \geq d$  and constraint them using the following three sets of clauses:

$$\bigwedge_{\substack{i \in [k] \\ (u, v) \in \Delta_n \\ j_1 \neq j_2 \in [n]}} (p(i, u, j_1) \wedge p(i, v, j_2)) \rightarrow c(i, u, v, |j_1 - j_2|)$$

$$\bigwedge_{\substack{i \in [k] \\ (u, v) \in \Delta_n \\ j_1 \neq j_2 \in [n]}} (p(i, u, j_1) \wedge p(i, v, j_2)) \rightarrow \neg c(i, u, v, |j_1 - j_2| + 1)$$

$$\bigwedge_{\substack{i \in [k] \\ (u, v) \in \Delta_n \\ d \in [n-1]}} c(i, u, v, d) \rightarrow c(i, u, v, d - 1)$$

Then we represent the inner sum  $d(u, v) = \sum_i d_i(u, v)$  from Equation (2) using the two sets of auxiliary variables;  $s_g(u, v, p)$  which when falsified enforces the condition  $d(u, v) \leq p$  and similarly  $s_l(u, v, p)$  which when falsified enforces the condition  $\sum_i d_i(u, v) \geq p$ . This is achieved using the following conditional cardinality constraints:

$$\bigwedge_{\substack{i \in [k] \\ (u, v) \in \Delta_n \\ k \leq p \leq k(n-1)}} \neg s_g(u, v, p) \rightarrow \{c(i, u, v, d) \mid d \in [n-1]\}_{\leq p}$$

$$\bigwedge_{\substack{i \in [k] \\ (u, v) \in \Delta_n \\ k \leq p \leq k(n-1)}} \neg s_l(u, v, p) \rightarrow \{c(i, u, v, d) \mid d \in [n-1]\}_{\geq p}$$

Recall from Equations (3) and (4), that our final goal is to bound the quantity  $\mathbb{I}(L) = \sum_{(u, v) \in \Delta_n} |d(u, v) - Z|$ , where  $Z = k(n+1)/3$ . The redundancy in the form of two sets of variables  $s_g$  and  $s_l$  is to allow us to later deal with the expression  $|d(u, v) - Z|$  and help count up from  $Z$  in either direction. We thus introduce auxiliary variables which encode the sign of the term  $d(u, v) - Z$ .  $s_0(u, v)$  is true if and only if  $d(u, v) = Z$ ;  $s_+(u, v)$  is true if  $d(u, v) > Z$ , false if  $d(u, v) < Z$ , and undefined otherwise. The following sets of clauses encode these auxiliary variables:

$$\bigwedge_{(u, v) \in \Delta_n} s_0(u, v) \rightarrow \neg s_g(u, v, Z) \wedge \neg s_l(u, v, Z)$$

$$\bigwedge_{(u, v) \in \Delta_n} (\neg s_0(u, v) \wedge s_+(u, v)) \rightarrow \neg s_l(u, v, Z + 1)$$

$$\bigwedge_{(u,v) \in \Delta_n} (\neg s_0(u,v) \wedge \neg s_+(u,v)) \rightarrow \neg s_g(u,v, Z-1)$$

Then, using the auxiliary variables  $f(u,v,q)$ , we represent the condition  $|d(u,v) - Z| \leq q$ . We encode these variables using the following set of clauses:

$$\bigwedge_{\substack{(u,v) \in \Delta_n \\ 1 \leq q \leq k(n-1) - Z}} (\neg s_0(u,v) \wedge s_+(u,v) \wedge s_g(u,v, Z+q)) \rightarrow f(u,v,q)$$

$$\bigwedge_{\substack{(u,v) \in \Delta_n \\ 1 \leq q \leq Z-k}} (\neg s_0(u,v) \wedge \neg s_+(u,v) \wedge s_l(u,v, Z-q)) \rightarrow f(u,v,q)$$

And finally, we express the bound  $\mathbb{I}(L) \leq t$  by adding the cardinality constraint:

$$\{ f(u,v,q) \mid (u,v) \in \Delta_n, 1 \leq q \leq \max(k(n-1) - Z, Z-k) \}_{\leq t}$$

### Symmetry Breaking

We refer to the encoding so far as the *base version* and denote the formula by  $\mathcal{F}(n,k,t)$ . We optionally add two sets of *symmetry-breaking* clauses which reduce the size of the search space by identifying equivalent solutions. These constraints are identical to those mentioned by Díaz et al. [3], however, in our approach, we selectively enable or disable these constraints depending on whether we are seeking a SAT instance (i.e., upper bound) or an UNSAT instance (i.e., infeasibility). The following set of clauses fixes the first row to be the identity permutation:

$$\bigwedge_{u \in [n]} p(1, u, u)$$

We refer to the encoding with these additional clauses as  $\mathcal{F}'(n,k,t)$ . Similarly, the following set of clauses forces the first column to follow a strictly increasing order, i.e., a lexicographic order:

$$\bigwedge_{(u,v), i \in [k-1]} p(i, v, 1) \rightarrow \neg p(i+1, u, 1)$$

We refer to the encoding with both these sets of additional clauses as  $\mathcal{F}''(n,k,t)$ . These two sets of symmetry breaking clauses enforce that the Latin rectangle is in its *reduced form* [3].

### Non-integral Instances

In the description of the encoding above, we made the assumption that 3 divides  $k(n+1)$ . In this case, we do not need to deal with fractional  $\frac{1}{3}$  values. We call these the *integral instances* and all other instances *non-integral instances*. To deal with the non-integral instances, we modify our encoding by multiplying the distances by 3. To this end, we use  $Z = k(n+1)$  instead of  $Z = k(n+1)/3$ . We use  $3|j_1 - j_2|$  instead of  $|j_1 - j_2|$  and we also multiply the bounds for  $d$ ,  $p$ , and  $q$  by 3.

## 4 Experimental Evaluation

To analyze the performance of our proposed encodings, we implemented a method for generating the encodings in Python 3.10.6 with help from the PySAT 0.1.7 library [8] for handling some of the higher-level cardinality constraints [12, 1, 5]. We then ran the SAT solver

`kissat` [2] with default settings on the generated encodings. We ran all our experiments on a 10-core Intel Xeon E5-2640 v4, 2.40 GHz CPU, with each process having access to 8GB RAM.

`kissat`, like other state-of-the-art CDCL solvers, is capable of producing a proof in the DRAT (Deletion Resolution Asymmetric Tautology) format for formulas which it claims to be unsatisfiable. Similar to how a mathematical proof of a theorem follows a sequence of smaller lemmas, a DRAT proof also consists of lemmas (in the form of a set of literals) and lemma deletion instructions. Due to the elementary nature of the proof format, it is straightforward for a CDCL solver to emit a DRAT proof with minimal overhead. This proof can then be independently verified using tools such as DRAT-trim [6, 15]. DRAT-trim takes as input the original CNF formula and the DRAT proof produced by the SAT solver and can verify, in time comparable to original proof producing time, that the proof indeed holds.

There are several ways of translating the higher-level cardinality constraints into elementary CNF clauses. Thus, we conducted a preliminary investigation to identify the most promising translations from those provided by the PySAT library<sup>1</sup>. We also tested the three versions of the encoding with different levels of symmetry breaking  $\mathcal{F}$ ,  $\mathcal{F}'$ ,  $\mathcal{F}''$  in combination with the different cardinality encoding types. From this investigation, we found that  $\mathcal{F}'$  combined with the `bitwise` cardinality encoding [12] and  $\mathcal{F}''$  combined with the `ladder` cardinality encoding [1] were the two best performing encodings for SAT and UNSAT instances, respectively. Hence, we stuck with these two combinations for the main experiments. In this preliminary investigation, we worked with instances which were already known to be either SAT or UNSAT. In the main experiments, typically, we don't know beforehand if the instance is SAT or UNSAT, thus we try both these encodings.

This paper focuses on the obtained bounds and their verification rather than on exact running times; the results reported by Díaz et al. serve as a basic comparison of the runtime performance with other methods. We conducted cursory experiments with a CSP model formulated with MiniZinc [11] for a direct and reproducible comparison on the same hardware. We tested on the solvers Gecode 6.3.0 and Chuffed 0.11.0, the former outperforming the latter. On small rectangles, up to around  $5 \times 7$ , the CSP approach is significantly faster than our SAT approach, but its performance quickly deteriorates as the size grows. For example, the CSP approach fails to replicate known bounds for  $6 \times 8$  within 10 hours even though, on the other hand, our approach produced solutions in under an hour.

Apart from the prototyping experiments, we conducted two main experiments rigorously. The first was for showing upper bounds and consequently generating Latin rectangles matching those bounds, and the second was to show lower bounds and consequently generating DRAT proofs of infeasibility. We worked with the same set of instances as Díaz et al. and used a timeout of 24 hours. We used our proposed encodings to replicate their upper bounds, produce verifiable proofs for their lower bounds, and also in some cases improve their upper bounds. The results of these experiments are summarized in Table 1. Table 2 shows some attributes of the proofs generated by our method for a representative set of instances. Figure 2 shows some of the Latin rectangles that witness the new upper bounds shown by us. In the supplementary material, we include the script used to generate the encodings and some generated encodings along with their corresponding models (if satisfiable) or DRAT proofs (if unsatisfiable).

---

<sup>1</sup> List of cardinality encoding types: `pysat.card.EncType`

■ **Table 1** Table of results. The top row indicates the value of  $k$  and the left column indicates the value of  $n$ . Grey cells indicate values claimed optimal by Díaz et al. using CSP/MIP. The cells with bold text indicate upper bounds that we could replicate. Cells with a ‘\*’ indicate lower bounds that we could confirm and certify with a DRAT proof (not applicable for rectangles with minimum imbalance 0). Green cells indicate an earlier upper bound (strike-through text) that we could improve.

	2	3	4	5	6	7	8	9	10	11	12
4	<b>2.6*</b>	<b>4*</b>	<b>5.3*</b>								
5	<b>8*</b>	<b>6*</b>	<b>8*</b>	<b>0</b>							
6	<b>16*</b>	<b>12*</b>	<b>13.3*</b>	<b>16*</b>	<b>0</b>						
7	<b>28*</b>	<b>22*</b>	<b>22.6*</b>	<b>22.6*</b>	<b>20*</b>	<b>18.6*</b>					
8	<b>40*</b>	<b>36*</b>	<b>32</b>	<b>30</b>	<b>24</b>	<b>28</b>	<b>0</b>				
9	<b>65.3*</b>	<b>56</b>	<del>56.6</del> 56.0	<del>56</del> 54	<del>52</del> 48	<del>66</del> 64.6	<del>60</del> 58.6	<b>0</b>			
10	<b>92*</b>	<del>86</del> 84	<del>92</del> 91.3	66.6	<del>102</del> 96	<del>100</del> 99.3	99.3	80	40		
11	<b>124*</b>	<del>120</del> 118	<del>122</del> 118	<del>122</del> 120	<del>126</del> 124	<del>136</del> 132	132	128	110	<b>0</b>	
12	168	158	162.6	170.6	120	183	184.6	178	174.6	147.3	<b>0</b>

■ **Table 2** Encoding size for some representative instances along with time required to prove optimality and the size of the generated DRAT proof. All these proofs were generated by running the SAT solver with the `--unsat` preset on  $\mathcal{F}''$  combined with the `ladder` cardinality encoding. Notice that the encoding size for the non-integral instance  $4 \times 6$  is much bigger than that of the integral instance  $6 \times 7$ .

$k \times n$	#variables	#clauses	Unsat Time	Proof Overhead	Proof Size
$3 \times 5$	2364	5750	<1 s	<1 s	20 KB
$4 \times 6$	73184	147903	45 s	2 s	11 MB
$6 \times 7$	37809	84906	1 hr 12 min	1 hr 45 min	1.6 GB

1	2	3	4	5	6	7	8	9	10
4	10	2	8	7	5	1	6	3	9
7	3	8	1	4	9	2	10	6	5

1	2	3	4	5	6	7	8	9	10	11
2	6	5	7	11	8	1	3	10	9	4
3	5	11	8	9	2	10	4	1	6	7
5	1	9	10	7	3	6	2	11	4	8

(a)  $3 \times 10$  rectangle with imbalance 84 (32 min). (b)  $4 \times 11$  rectangle with imbalance 118 (102 min).

1	2	3	4	5	6	7	8	9	10
5	8	6	1	9	3	10	4	2	7
6	5	9	3	2	7	4	10	1	8
7	6	1	5	10	2	8	9	4	3
9	1	7	10	4	5	6	3	8	2
10	3	5	7	1	8	9	2	6	4

1	2	3	4	5	6	7	8	9	10	11
2	11	7	6	4	5	1	9	8	3	10
3	7	9	5	10	2	6	1	11	4	8
5	1	11	10	7	3	8	6	4	2	9
6	10	1	3	8	7	2	11	5	9	4
7	4	10	1	9	11	3	2	6	8	5

(c)  $6 \times 10$  rectangle with imbalance 96 (40 min). (d)  $6 \times 11$  rectangle with imbalance 124 (5.5 hr).

■ **Figure 2** Examples of Latin rectangles witnessing new upper bounds along with the time required by the SAT solver to find these solutions in parentheses. All these rectangles were found by the version of the encoding that combines  $\mathcal{F}''$  with the `ladder` cardinality encoding. Note that, no lower bounds are known for these sizes.

## 5 Conclusion

This work presented a SAT approach to the balanced Latin rectangle problem treating complex arithmetic expressions by translating them into propositional logic. Our approach is competitive with the prior CSP/MIP-based techniques with a slight advantage. On the one hand, the SAT approach can find tighter upper bounds for realizing Latin rectangles. On the other hand, our approach performs comparably to the MIP approach when it comes to proving the infeasibility of particular Latin rectangles. This finding is significant, as propositional logic is often considered inferior to MIP when handling complex arithmetic expressions. A key advantage of our SAT approach is its ability to produce DRAT proofs of optimality that can be independently verified. The proof generation causes only some minor computational overhead. On a more technical level, our work shows how to handle non-trivial nested summations in a SAT encoding by chaining ladders and cardinality counters. We observe that it is sometimes beneficial to construct two different encodings tailored towards SAT and UNSAT instances in problems where one is interested in both upper and lower bounds. For instance, these encodings can use different levels of symmetry-breaking and different types of cardinality encodings.

We see many potential avenues for future work. One could use techniques such as cubing [7] to tackle the instances for which proving optimality is still challenging. The cubing can be performed by either fixing the values of the first column, fixing the imbalance of certain pairs to be 0, or bounding the number of pairs with 0 imbalance. We can enumerate all unique (up to isomorphism) Latin rectangles of a particular size and imbalance by extending the SAT approach to an incremental solving approach. Lastly, another natural direction is to apply Pseudo-Boolean solving to this problem. Pseudo-Boolean solving can also produce cutting planes proofs for verification [14] and might be even better suited to handle the arithmetic constraints involved in this problem.

## References

- 1 Carlos Ansótegui and Felip Manyà. Mapping problems with finite-domain variables to problems with boolean variables. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 1–15. Springer Verlag, 2004. doi:10.1007/11527695\_1.
- 2 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020. URL: [https://tuhat.helsinki.fi/ws/files/142452772/sc2020\\_proceedings.pdf](https://tuhat.helsinki.fi/ws/files/142452772/sc2020_proceedings.pdf).
- 3 Mateo Díaz, Ronan Le Bras, and Carla P. Gomes. In search of balance: The challenge of generating balanced Latin rectangles. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 68–76. Springer, 2017. doi:10.1007/978-3-319-59776-8\_6.
- 4 Johannes K. Fichte, Daniel Le Berre, Markus Hecher, and Stefan Szeider. The silent (r)evolution of SAT. *Communications of the ACM*, 66(6):64–72, June 2023. doi:10.1145/3560469.
- 5 Ian P. Gent and Peter Nightingale. A new encoding of alldifferent into SAT. In *International Workshop on Modelling and Reformulating Constraint Satisfaction*, volume 3, pages 95–110, 2004.
- 6 Marijn Heule, Warren A. Hunt Jr., Matt Kaufmann, and Nathan Wetzler. Efficient, verified checking of propositional proofs. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *Lecture Notes in Computer Science*, pages 269–284. Springer Verlag, 2017. doi:10.1007/978-3-319-66107-0\_18.
- 7 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving very hard problems: Cube-and-conquer, a hybrid SAT solving method. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4864–4868. ijcai.org, 2017. doi:10.24963/ijcai.2017/683.
- 8 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018. doi:10.1007/978-3-319-94144-8\_26.
- 9 Marcus Jones, Richard Woodward, and Jerry Stoller. Increasing precision in agronomic field trials using latin square designs. *Agronomy Journal*, 107(1):20–24, 2015. doi:10.2134/agronj14.0232.
- 10 Ronan LeBras, Carla P. Gomes, and Bart Selman. From streamlined combinatorial search to efficient constructive procedures. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012. doi:10.1609/aaai.v26i1.8147.
- 11 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007. doi:10.1007/978-3-540-74970-7\_38.
- 12 Steven David Prestwich. CNF encodings. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 75–97. IOS Press, 2009.
- 13 H.M. Van Es and C.L. Van Es. Spatial nature of randomization and its effect on the outcome of field experiments. *Agronomy Journal*, 85(2):420–428, 1993.
- 14 Marc Vinyals, Jan Elffers, Jesús Giraldez-Cru, Stephan Gocht, and Jakob Nordström. In between resolution and cutting planes: A study of proof systems for pseudo-boolean SAT

solving. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 292–310. Springer Verlag, 2018. doi:10.1007/978-3-319-94144-8\_18.

- 15 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing – SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer Verlag, 2014. doi:10.1007/978-3-319-09284-3\_31.

# Enumerative Level-2 Solution Counting for Quantified Boolean Formulas

Andreas Plank ✉ 🏠 

Institute for Symbolic Artificial Intelligence, JKU Linz, Austria

Sibylle Möhle ✉ 🏠 

Max Planck Institute for Informatics, Saarbrücken, Germany

Martina Seidl ✉ 🏠 

Institute for Symbolic Artificial Intelligence, JKU Linz, Austria

---

## Abstract

We lift the problem of enumerative solution counting to quantified Boolean formulas (QBFs) at the second level. In contrast to the well-explored model counting problem for SAT ( $\#SAT$ ), where models are simply assignments to the Boolean variables of a formula, we are now dealing with tree (counter-)models reflecting the dependencies between the variables of the first and the second quantifier block. It turns out that enumerative counting on the second level does not give the complete model count. We present the – to the best of our knowledge – first approach of counting tree (counter-)models together with a counting tool that exploits state-of-the-art QBF technology. We provide several kinds of benchmarks for testing our implementation and illustrate in several case studies that solution counting provides valuable insights into QBF encodings.

**2012 ACM Subject Classification** Theory of computation → Automated reasoning

**Keywords and phrases** QBF, Second-Level Model Counting

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.49

**Category** Short Paper

**Supplementary Material** *Software:* <https://qcount.pages.sai.jku.at/12count>

**Funding** This work has been supported by the LIT AI Lab funded by the State of Upper Austria.

## 1 Introduction

Over the last decade, solvers for quantified Boolean formulas (QBFs) have become appealing tools to handle PSPACE-hard problems as found in many applications from, for example, artificial intelligence and formal verification (see [21] for a survey). Much progress has been made in the theory and practice of solving [3, 18], partly relying on generalizations of techniques from SAT, but partly being enabled by new genuine QBF techniques. However, some aspects of QBFs are hardly explored yet. One of these is counting the number of solutions of a formula. The problem of counting the number of solutions of a given QBF is also known as  $\#QBF$  [13]. In contrast to  $\#SAT$  [9], the counting problem of propositional logic, which is used in many application domains including probabilistic reasoning [7, 19], verification of neural networks [1, 16] and the analysis of software vulnerability [5, 24],  $\#QBF$  has mainly been studied theoretically [13, 10, 2].

Only recently, some work has been presented dealing with counting the solutions of a QBF at the outer-level [22]. Given a true QBF with first quantifier block  $\exists X$ , this work is concerned with the problem of counting the number of assignments to the variables in  $X$  leading to a QBF that is true. The authors also consider the dual problem, i.e., given a false QBF starting with quantifier block  $\forall Y$ , how many assignments of the variables  $Y$  result in a false QBF? In order to answer these questions, they presented an enumerative approach that



© Andreas Plank, Sibylle Möhle, and Martina Seidl;  
licensed under Creative Commons License CC-BY 4.0

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).

Editor: Roland H. C. Yap; Article No. 49; pp. 49:1–49:10

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



directly resulted from enumerative propositional model counting [6, 8]. Therefore, a solver is used to obtain one solution, which is then excluded from the formula to obtain a different solution. This process is repeated until no further solution is found.

In this work, we go one step further by dealing with solutions of true QBFs that start with quantifier prefix  $\forall X \exists Y$  and false QBFs that start with quantifier prefix  $\exists X \forall Y$ , i.e., in the first case, we count models and in the second case, we count counter-models. For both cases, the solutions are not simply propositional assignments as in outer-level counting, but tree models and counter-models capturing the dependencies between the variables of the first and second quantifier block. While tree (counter-)models are a very convenient way to describe QBF solutions, in practice, QBF solvers represent solutions by Boolean functions. We will use both views on QBF solutions to investigate to what extent enumeration-based counting can be lifted to the second level. Therefore, solutions are excluded by conjunctively/disjunctively adding (negated) functions to the formula until no further solutions are found. It turns out, that enumeration-based solution counting at the second level does not lead to the full model count, but still gives valuable insights about the solutions of a formula. We implemented the approach in a first prototype to evaluate how it works in practice. To this end, we build on state-of-the-art QBF solving technology like incremental solving and function extraction.

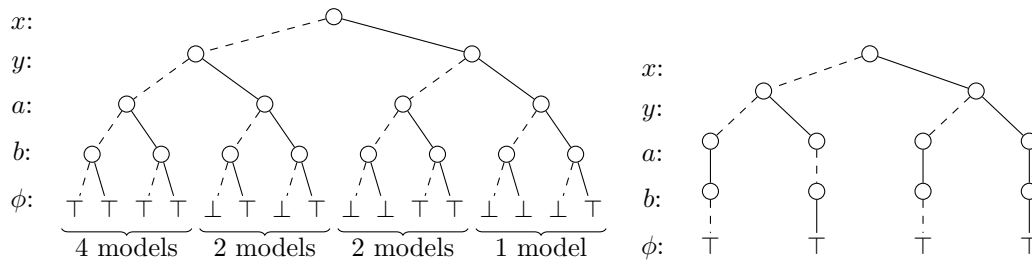
This paper is structured as follows. First, we introduce the necessary preliminaries in the next section. Then we present the level-2 counting problem by an example in Section 3 before we discuss enumeration-based solution counting in Section 4. In Section 5, evaluate our implementation on three different types of benchmarks, before we conclude in Section 6.

## 2 Preliminaries

We consider quantified Boolean formulas of the form  $\Pi.\phi$ , where  $\Pi = Q_1 X_1 \dots Q_n$  is called *quantifier prefix* (with  $Q_i \in \{\forall, \exists\}$ ,  $Q_i \neq Q_{i+1}$  for  $i \in \{1, \dots, n-1\}$  and  $X_1, \dots, X_n$  are pairwise disjoint, non-empty sets of Boolean variables). The *matrix*  $\phi$  is a propositional formula over variables  $X_i$  with standard Boolean connectives  $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$  and  $\oplus$  (XOR). The prefix  $\Pi$  induces an ordering on the variables:  $x_i <_{\Pi} x_j$  if  $x_i \in X_i$ ,  $x_j \in X_j$  and  $i < j$ . If prefix  $\Pi$  is clear from the context, we just write  $x_i < x_j$ . In this paper, we consider only *closed* formulas, i.e., every variable that occurs in matrix  $\phi$  also occurs in the prefix  $\Pi$ .

A QBF  $\Pi.\phi$  is in *prenex conjunctive normal form* (PCNF), if  $\phi$  is a conjunction of clauses. A *clause* is a disjunction of literals and a *literal* is a variable or a negated variable. If  $l$  is a literal, then  $\text{var}(l) = x$  if  $l = x$  or  $l = \neg x$ . As usual,  $\bar{l} = x$  if  $l = \neg x$  and  $\bar{l} = \neg x$  otherwise. For a QBF  $\varphi = Q_1 X_1 \dots Q_n X_n.\phi$ ,  $\text{var}(\varphi) = X_1 \cup \dots \cup X_n$ . An *assignment*  $\sigma$  of  $\varphi$  is a set of literals over (a subset of)  $\text{var}(\varphi)$  such that there is no  $l \in \sigma$  with  $\bar{l} \in \sigma$ . For an assignment  $\sigma$ ,  $\text{var}(\sigma) = \{\text{var}(l) \mid l \in \sigma\}$ . If  $\text{var}(\sigma) = \text{var}(\varphi)$ , then  $\sigma$  is a *full* assignment, else it is a *partial* assignment. A (partial) X-assignment is an assignment over a (sub-)set of variables  $X$ .

Given a propositional formula  $\phi$  and an assignment  $\sigma$ , then  $\phi_{\sigma}$  denotes the formula obtained when setting all variables  $x \in \text{var}(\sigma)$  to true if  $x \in \sigma$  and to false if  $\neg x \in \sigma$ , respectively. Based on this notation, the semantics of a QBF is defined as follows:  $\forall x \Pi.\phi$  is true iff  $\Pi.\phi_{\{x\}}$  and  $\Pi.\phi_{\{\neg x\}}$  are true. A QBF  $\exists x \Pi.\phi$  is true iff  $\Pi.\phi_{\{x\}}$  or  $\Pi.\phi_{\{\neg x\}}$  is true. For example, the QBF  $\forall x \exists y.(x \leftrightarrow y)$  is true, while the QBF  $\exists y \forall x.(x \leftrightarrow y)$  is false. A *model* for a QBF  $\varphi = \Pi.\phi$  with  $|\text{var}(\varphi)| = m$  is a tree of height  $m+1$  such that every node at level  $k \in \{1, \dots, m\}$  is labeled with a variable  $x_k$  in the order of the prefix, i.e., if variable  $x_j$  is at level  $j$  and variable  $x_k$  is at level  $k$  with  $j < k$  then  $x_j \leq_{\Pi} x_k$ . A node at level  $k$  has one child if  $x_k \in X_i$  and  $Q_i = \exists$ , and two children if  $Q_i = \forall$ . In the graphical representation, a dashed edge indicates that the parent variable is set to false and a solid edge indicates that it is set to true. Examples are shown in Figure 1. The path from the root to the leaves gives a full assignment under which  $\phi$  evaluates to true.



**Figure 1** The left tree shows the full assignment tree for the given QBF. Tree models (16 in total) are subtrees of the assignment tree such that universal nodes have two children, existential nodes have one child and all leaves are  $\top$ . An example of a tree model is shown on the right.

A counter-model is defined dually, with the difference that universal nodes have one child node, existential nodes have two child nodes, and the matrix evaluates to false under the respective assignments. An alternative representation of tree models that is practically more relevant are sets of Skolem functions, so-called Skolem sets. A *Skolem set*  $F$  of a true QBF  $\Phi$  is a set of Boolean functions such that for each each existential variable  $y$  of  $\Phi$ , there is a function  $f_y(x_1, \dots, x_n) \in F$  where  $x_1, \dots, x_n$  are the universal variables of  $\Phi$  with  $x_i < y$ . The elements of a Skolem set are called Skolem functions. If we take the matrix of  $\Phi$  and replace all the existential variables by their Skolem functions, we obtain a valid propositional formula over the universal variables of  $\Phi$ . A Skolem set for the true QBF  $\forall x \exists y. (x \leftrightarrow y)$  is  $\{f_y = x\}$ . If we replace  $y$  by  $f_y$ , we obtain the valid formula  $(x \leftrightarrow x)$ . If  $\sigma_X$  is an assignment to a set of universal variables  $X$ , then  $f_y(\sigma_X)$  denotes the value of  $y$  w.r.t. the assignment of the variables in  $X$ . For false QBFs and their tree counter-models, functions for the universal variables are defined dually. Such functions are called *Herbrand functions* and are collected in Herbrand sets. If we replace all universal variables by their Herbrand functions of a Herbrand set, we obtain an unsatisfiable formula.

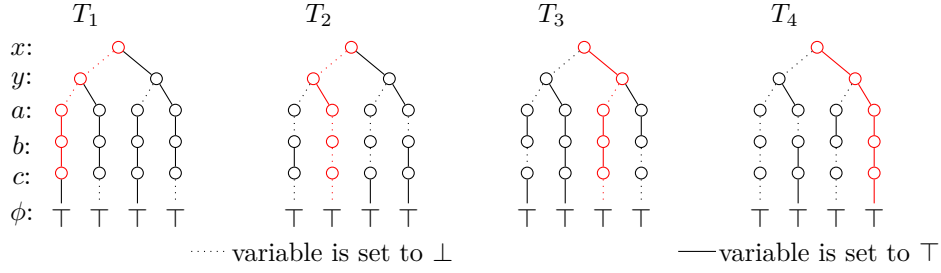
### 3 Counting at Level Two

In the following, we introduce the QBF level-2 counting problem by an example. Consider the true QBF

$$\Phi = \forall x, y \exists a, b. (\neg x \vee a) \wedge (\neg y \vee b).$$

On the left of Figure 1 we see the full assignment tree of  $\varphi = \Pi.\phi$ . Each complete path from the root to a leaf node represents a full assignment to the variables in  $\varphi$ . A tree model of  $\varphi$  is a subtree of this assignment tree such that nodes  $x$  and  $y$  have two children, nodes  $a$  and  $b$  have one child each, and the leaves are labeled by  $\top$ . An example of a tree model is shown on the right of Figure 1. We are concerned with the question of counting the number of tree models of a given true QBF. Alternatively, we could also ask the question how many different Skolem sets a true QBF has. Here it is important that syntactically different, but semantically equivalent Skolem sets are not counted multiple times. For example the Skolem functions  $f_a = \top$  and  $f_a = (x \vee \neg x)$  of our example are semantically equivalent even though they are syntactically different.

A direct approach to count tree models is to iterate over all assignments  $\sigma$  of the universal variables in the first quantifier block of a true QBF  $\varphi = \Pi.\phi$ . With a propositional model counter we can determine the number of satisfying assignments of  $\phi_\sigma$ . If  $S$  is the set of all assignments of the universal variables in the first quantifier block, then the total number



■ **Figure 2** Tree models of  $\forall x \forall y \exists a \exists b \exists c. (\neg x \vee a \vee c) \wedge (\neg y \vee b \vee \neg c)$ .

of tree models is  $\Pi_{\sigma \in S} \#SAT(\phi_\sigma)$ . Counting counter-models is defined dually, with the difference that the role of existential and universal quantifiers is exchanged, i.e., the formulas need to have a prefix starting with  $\exists X \forall Y$ . While this direct approach results in a complete solution counter, it is very inefficient to iterate over all assignments of the variables in the outermost quantifier block and solve a propositional counting problem for each assignment. In the following section, we therefore investigate if an enumeration-based approach relying on recent QBF solving technology is possible.

#### 4 Enumerative Model Counting for 2QBF

In this section, we formalize enumeration-based solution counting for the second level. For technical simplicity, we focus on true 2QBFs (formulas with quantifier prefix  $\forall X \exists Y$ ). To this end, we lift the idea of blocking clauses to blocking functions. Recall that a blocking clause in SAT is the negation of a model  $\sigma$  of a formula  $\phi$ . If  $\phi$  is enriched with  $\neg\sigma$ , then  $\sigma$  is excluded from the solution space. For QBFs, we introduce the notion of blocking Skolem set as follows.

► **Definition 1.** Let  $\Phi = \forall X \exists Y. \phi$  be a true QBF and let  $F$  be a Skolem set of  $\Phi$ . Then  $\neg\phi_F$  is a blocking Skolem set of  $\Phi$  where  $\phi_F = \bigwedge_{f_y \in F} (y \leftrightarrow f_y)$ .

In the following example, we now use blocking Skolem sets to exclude solutions.

► **Example 2.** Consider the true QBF  $\Phi = \forall x \forall y \exists a \exists b \exists c. (\neg x \vee a \vee c) \wedge (\neg y \vee b \vee \neg c)$ . We now incrementally add blocking Skolem sets  $\neg\psi_{F_i}$  until the formula becomes false. The resulting models are shown in Figure 2.

1. One solution of this formula is tree  $T_1$ , which is represented by Skolem set  $F_1 = \{f_a(x, y) = \top, f_b(x, y) = \top, f_c(x, y) = \neg y\}$ . To exclude  $F_1$ , we add  $\neg\psi_{F_1}$  to  $\Phi$  and obtain  $\Phi_1 = \forall x \forall y \exists a \exists b \exists c. (\neg x \vee a \vee c) \wedge (\neg y \vee b \vee \neg c) \wedge \neg\psi_{F_1}$ . Now  $T_1$  is no solution of  $\Phi_1$ .
2. A solution of  $\Phi_1$  is  $T_2$  with Skolem set  $F_2 = \{f_a(x, y) = \perp, f_b(x, y) = (x \leftrightarrow y), f_c(x, y) = x\}$ . To exclude  $F_2$  as well, we get  $\Phi_2 = \forall x \forall y \exists a \exists b \exists c. (\neg x \vee a \vee c) \wedge (\neg y \vee b \vee \neg c) \wedge \neg\psi_{F_1} \wedge \neg\psi_{F_2}$ .
3. Next, we get tree model  $T_3$  with Skolem set  $F_3 = \{f_a(x, y) = \top, f_b(x, y) = (x \oplus y), f_c(x, y) = \neg x\}$ . We exclude  $F_3$  from  $\Phi_2$  as before and obtain  $\Phi_3$ .
4. The next tree model we find is  $T_4$  with Skolem set  $F_4 = \{f_a(x, y) = x, f_b(x, y) = y, f_c(x, y) = y\}$ .
5. Finally, the QBF  $\Phi_4 = \forall x \forall y \exists a \exists b \exists c. \phi \wedge \neg\psi_{F_1} \wedge \neg\psi_{F_2} \wedge \neg\psi_{F_3} \wedge \neg\psi_{F_4}$  is false.

We found four different tree models, each with four branches. Hereby Tseitin transformation is used to add the blocking function to the PCNF formula before each solver call. From these four models, we can assemble more models by combining the red branches of

Figure 2. To calculate the full model count, all possible combinations of the paths need to be considered, resulting in  $4^4$  models in total (there are four choices for the branch  $\{\bar{x}, \bar{y}\}$ , four choices for the branch  $\{x, \bar{y}\}$  and so on). When we take a closer look, however, the count is not correct, as also  $\{\bar{x}, \bar{y}, a, b, \bar{c}\}$  is a model of the matrix of  $\Phi$ , but it has not been considered.

As the example above shows, blocking Skolem sets can indeed be used to exclude solutions, but they are sometimes too restrictive. To describe what is happening when adding blocking Skolem sets to a formula, we need to introduce the following definitions.

► **Definition 3.** Let  $\Phi = \forall X \exists Y. \phi$  be a true QBF and let  $T_1$  and  $T_2$  be tree models of  $\Phi$ . Then  $T_1$  and  $T_2$  are disjoint if there are no complete paths  $\sigma_1$  of  $T_1$  and  $\sigma_2$  of  $T_2$  with  $\sigma_1 = \sigma_2$ .

A complete path describes a full assignment (i.e., an assignment of all variables), such that the according branch of the tree model evaluates to true. The trees of Figure 2 are disjoint, because all their paths are different. If we want to characterize the models of a QBF in terms of Skolem sets, we get the following definition.

► **Definition 4.** Let  $\Phi = \forall X \exists Y. \phi$  be a true QBF and let  $F$  and  $G$  be Skolem sets of  $\Phi$ . Then  $F$  and  $G$  are called disjoint if for each full assignment  $\sigma_X$  of the universal variables  $X$  there exists an existential variable  $y \in Y$  such that  $f_y(\sigma_X) \neq g_y(\sigma_X)$  with  $f_y \in F$  and  $g_y \in G$ .

The notion of disjoint models is rather strong as it forbids to have to have common paths in the tree model representation. We therefore also introduce the notion of *different* models requiring that at least one path of two tree models is different. This notion is transferred to Skolem sets as follows.

► **Definition 5.** Let  $\Phi = \forall X \exists Y. \phi$  be a true QBF. Furthermore, let  $F$  and  $G$  be Skolem sets of  $\Phi$ . Then  $F$  and  $G$  are called different if there exists a full assignment  $\sigma_X$  of the universal variables  $X$  such that there is an existential variable  $y \in Y$  with  $f_y(\sigma_X) \neq g_y(\sigma_X)$ ,  $f_y \in F$ , and  $g_y \in G$ .

Obviously, any two disjoint models are different. The other direction does not hold. To count all models of a QBF, we need to count the different models. As the following lemma shows, using blocking Skolem sets excludes disjoint models only.

► **Lemma 6.** Let  $\Phi = \forall X \exists Y. \phi$  be a QBF and  $F$  be a Skolem set of  $\Phi$ . If  $G$  is a Skolem set of  $\Phi' = \forall X \exists Y. \phi \wedge \neg \phi_F$ , where  $\neg \phi_F$  is a blocking Skolem set as defined above, then  $F$  and  $G$  are disjoint.

**Proof.** Assume that  $F$  and  $G$  are not disjoint. Then there is an assignment  $\sigma_X$  such that for all  $y \in Y$  it holds that  $f_y(\sigma_X) = g_y(\sigma_X)$  for  $f_y \in F, g_y \in G$ . Now we extend the assignment  $\sigma_X$  to an assignment over  $X \cup Y$  as follows:  $\sigma = \sigma_X \cup \{y \mid f_y(\sigma_X) = \top, f_y \in F\} \cup \{\neg y \mid f_y(\sigma_X) = \perp, f_y \in F\} = \sigma_X \cup \{y \mid g_y(\sigma_X) = \top, g_y \in G\} \cup \{\neg y \mid g_y(\sigma_X) = \perp, g_y \in G\}$ . As  $G$  is a Skolem set of  $\Phi'$ ,  $\sigma$  has to satisfy  $\neg \phi_F$ . But by its construction,  $\sigma$  also satisfies  $\phi_F$ , leading to a contradiction. Hence,  $F$  and  $G$  have to be disjoint. ◀

As the following lemma shows, we can use the enumerative approach to calculate the maximum number of pairwise disjoint Skolem sets.

► **Proposition 7.** Let  $\Phi = \forall X \exists Y. \phi$  be a true QBF and let  $F_1, \dots, F_m$  be pairwise disjoint Skolem sets of  $\Phi$  such that

$$\Phi' = \forall X \exists Y. \phi' = \forall X \exists Y. (\phi \wedge \neg \phi_{F_1} \wedge \dots \wedge \neg \phi_{F_m})$$

is false. Then  $m$  is the maximum number of pairwise disjoint Skolem sets.

**Proof.** Since  $\Phi'$  is false, there is at least one assignment  $\sigma_X$  such that  $\phi'$  is unsatisfiable under  $\sigma_X$ . Assume there is a Skolem set  $F_{m+1}$  that is pairwise disjoint with  $F_1, \dots, F_m$ . Let  $\sigma = \sigma_X \cup \{x \mid f_x(\sigma_X) = \top, f_x \in F\} \cup \{\neg x \mid f_x(\sigma_X) = \perp, f_x \in F\}$ . Since  $F_{m+1}$  is a Skolem set of  $\Phi$ ,  $\phi$  is satisfied by  $\sigma$ . Further,  $\sigma$  satisfies  $\neg\phi_{F_i}$  with  $1 \leq i \leq m$ , because  $F_i$  and  $F_{m+1}$  are disjoint. Hence,  $\phi'$  is satisfied by  $\sigma$ . This contradicts the assumption that  $\phi'$  is unsatisfiable under  $\sigma_X$ .  $\blacktriangleleft$

The maximum number of pairwise disjoint models is determined by the assignments of the variables in the outermost universal quantifier block that, when the universal variables are assigned accordingly, lead to the fewest propositional models.

► **Lemma 8.** *Let  $\Phi = \forall X \exists Y. \phi$  be a true QBF and let  $S$  be the set of all full assignments of universal variables  $X$ . Then the maximum number of pairwise disjoint models of  $\Phi$  is  $\min(\{\#SAT(\phi_\sigma) \mid \sigma \in S\})$ .*

The proof of the lemma above follows directly from the construction of disjoint models. For example, the QBF  $\forall x, y \exists a, b. (\neg x \vee a) \wedge (\neg y \vee b)$  with the assignment tree shown in Figure 1 has only one disjoint model, because for the assignment  $\sigma = \{x, y\}$ , there is only one assignment to  $\{a, b\}$  that satisfies the matrix under  $\sigma$ . In contrast, the QBF  $\forall x \forall y \exists a \exists b \exists c. (\neg x \vee a \vee c) \wedge (\neg y \vee b \vee \neg c)$  from Example 2 has four disjoint models, because of the assignment  $\{x, y\}$ . In practical encodings, those assignments of the universal variables determine the number of disjoint models for which the assignment of the outermost variables not immediately falsifies the formula. The number of disjoint models gives us a lower bound for the full model count.

► **Proposition 9.** *Given a true QBF  $\Phi = \forall X \exists Y. \phi$  with  $|X| = n$  and  $m$  pairwise disjoint tree models. Then  $\Phi$  has at least  $m^{2^n}$  different tree models.*

If we know that a true QBF  $\Phi = \forall X \exists Y. \phi$  with  $|X| = n$  has  $m$  pairwise disjoint models, we can calculate the full model count as follows. Let  $\Phi' = \forall X \exists Y. \phi'$  with  $\phi' = \phi \wedge \neg\phi_{F_1} \wedge \dots \wedge \neg\phi_{F_m}$  be the false QBF obtained by enriching  $\Phi$  with the  $m$  blocking Skolem sets. Furthermore, let  $S$  be the set of assignments of variables  $X$  such that  $\phi'_\sigma$  is true  $\forall \sigma \in S$ . Then the full model count is

$$\prod_{\sigma \in S} \#SAT(\phi'_\sigma) * m^{2^n - |S|}$$

While the enumerative approach also works for true QBFs with prefix  $\forall X \exists Y \forall Z \Pi. \phi$  for counting disjoint partial Skolem sets that consider only functions of the set  $Y$ , it is not possible to obtain a full model count of partial Skolem sets in the  $Y$  variables by using a propositional model counter. Here, the approach needs to be applied recursively.

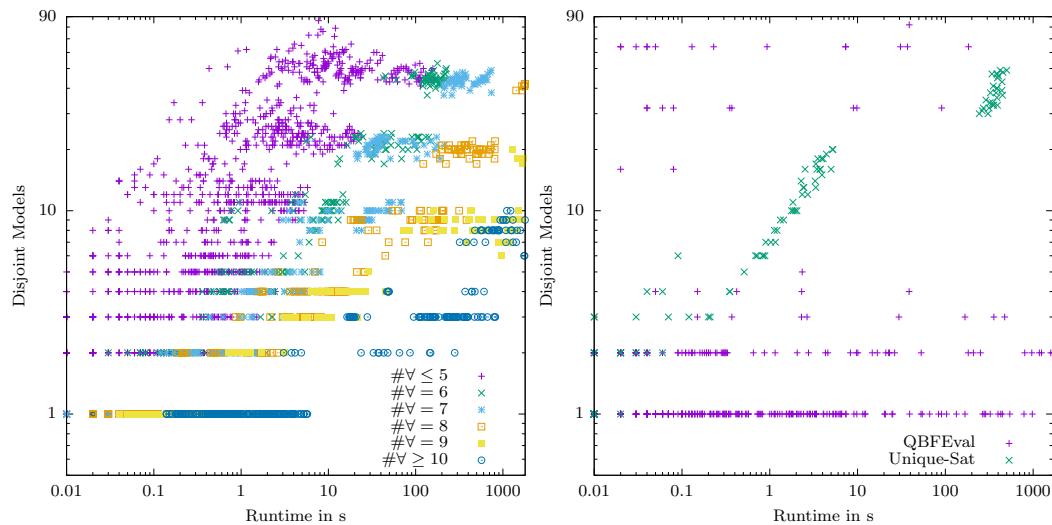
The enumerative approach for counting disjoint models also directly transfers to an enumerative approach for counting disjoint counter-models of false QBFs starting with the prefix  $\exists X \forall Y \exists Z$ .<sup>1</sup>

► **Proposition 10.** *Let  $\Phi = \exists X \forall Y \exists Z. \phi$  be a false QBF and let  $H_1, \dots, H_m$  be disjoint Herbrand sets such that*

$$\Phi' = \forall X \exists Y. (\phi \vee \phi_{H_1} \vee \dots \vee \phi_{H_m})$$

*is true with  $\phi_H = \bigwedge_{h_y \in H} (y \leftrightarrow h_y)$ . Then  $m$  is the maximum number of pairwise disjoint Herbrand sets over the variables from  $Y$ .*

<sup>1</sup> Note that we include the last quantifier block to deal with formulas in PCNF. Otherwise, the universal variables could be immediately removed.



■ **Figure 3** Runtime related to number of disjoint models for randomly generated formulas (left) and Unique-SAT/QBF Eval benchmarks (right).

## 5 Evaluation

We implemented the previously described approach in the tool `qCounter`.<sup>2</sup> As backend solver our tool relies on the QBF solver DepQBF 6.03 [15] which is able to produce Q-resolution proofs for true and false formulas from which Skolem and Herbrand functions can be extracted. We used the QBF certification framework QBF Cert [17] to obtain the functions in the Aiger format.<sup>3</sup> We implemented a small tool to convert the blocking functions of the variables from the second quantifier block. Therefore, the Skolem functions needed to be negated and appended conjunctively to the current QBF by using the incremental interface of DepQBF. To disjunctively add the Herbrand functions, an extra Tseitin transformation step is necessary to obtain a formula in PCNF. While this transformation introduces auxiliary variables we can avoid an exponential increase of the formula using this technique [23, 12]. For this we also used the incremental interface of DepQBF with its ability to add temporary clauses. To calculate the full model count as described above, we employed the propositional model counter Ganak [20] together with the SAT solver Lingeling [4].

Currently, there exist no standard benchmark sets which we could use to evaluate the correctness and performance of our implementation. To this end, we propose three different benchmark sets: (1) randomly generated formulas, (2) QBF encodings of the unique-SAT problem (does a given propositional formula have exactly one solution?) and (3) benchmarks from past QBF Evaluations with a suitable quantifier structure. Whereas the benchmarks from (1) and (2) are constructed in such a way that the number of models is known, this is not the case for the benchmarks from (3). Details follow below. All experiments were performed on a cluster of dual-socket AMD EPYC 7313 @  $16 \times 3.7\text{GHz}$  machines with 4GB memory limit and 1800 seconds as timeout.

<sup>2</sup> The tool, the benchmarks and the log-files are publicly available at <https://qcount.pages.sai.jku.at/l2count>

<sup>3</sup> <http://fmv.jku.at/aiger/>

## 5.1 Randomly Generated Formulas

We provide a generator that returns true 2-QBFs in PCNF with quantifier structure  $\forall X \exists Y$  such that the number of (disjoint) models is known. The parameters  $n$  (size of  $X$ ),  $m$  (size of  $Y$ ) have to be provided. For producing the clauses of the matrix, the generator iterates over all possible assignments of the universal variables  $X$  and excludes propositional models by randomly assigning the existential variables  $Y$  and building blocking clauses. For example, for a prefix  $\forall x_1, x_2 \exists y_1, y_2$  the clause  $(x_1 \vee \bar{x}_2 \vee y_1 \vee y_2)$  prohibits in any QBF model that in the branch where  $x_1$  is set to false and  $x_2$  is set to true, both  $y_1$  and  $y_2$  are both set to false. The number of disjoint models is determined by the branch with the smallest number of propositional models (see also Lemma 8).

We generated 3950 formulas with  $2 \leq |X|, |Y| \leq 11$  having up to 100 disjoint models each. For 2608 of these formulas, the number of disjoint models could be determined. The results are shown in the left plot of Figure 3. We cluster the results according to the size of  $X$ , indicating that not only the number of disjoint models impacts the runtime, but also the size of the first quantifier block.

## 5.2 Unique-SAT Encodings

The question whether a given propositional formula  $\phi$  has exactly one model can be encoded by a QBF  $\exists X \forall \psi$  as shown in [11]. In this QBF,  $X$  contains the variables of  $\phi$ ,  $Y$  is a copy of the variables of  $\phi$ , and  $\psi$  is constructed in such a way that the QBF is true iff  $\phi$  has exactly one model. The prefix fits for counting the counter-models at level-2 in case the QBF is false. The number of QBF counter-models can be determined based on the number of the models of  $\phi$ . If  $\phi$  is unsatisfiable and has  $n$  variables, then the number of disjoint counter-models equals the number of different counter-models and is  $2^{2^n}$ . If  $\phi$  has  $m$  models, then it has  $(m-1)$  disjoint models and  $(2^n)^{2^n-m} (m-1)^m$  different models.

We generated 497 Unique-SAT formulas based on true propositional formulas with 26 to 240 variables and 61 and 643 clauses by using the random generator from [14]. We considered only propositional formulas with more than one model in order to count disjoint counter-models of the resulting false QBFs. The performance of our tool is shown in the right plot of Figure 3. The number of disjoint counter-models enumerated by our tool could be quickly validated with a propositional model counter. In this way, we could check the results of our tool for false formulas.

## 5.3 Benchmarks from QBF Evaluations

As a final set of benchmarks, we considered formulas from the main tracks of QBF Eval 2022 and QBF Eval 2008.<sup>4</sup> From the QBF Eval 2022 set, we identified 18 true formulas and 102 false formulas with a suitable prefix structure that could be solved by plain DepQBF within a time limit of 1800 seconds. In a similar way, we selected two true formulas and 683 false formulas from the QBF Eval 2008 set. For the 2022 formulas, we could determine the number of disjoint models of 3 formulas and the number of disjoint counter-models of 53 formulas. For the 2008 formulas, we could determine the number of disjoint models for both formulas and the number of disjoint counter-models of 285 formulas. Interestingly, many of the formulas have only one disjoint model indicating that the search space is strongly

---

<sup>4</sup> <http://www.qbflib.org>

restricted for certain assignments of the variables in the outermost quantifier block. On the other hand, for some of the formulas we could find up to 82 disjoint (counter-)models. Details are shown in the right plot of Figure 3.

## 6 Conclusion

We considered the problem of counting level-2 (counter-)models for QBFs. It turned out that enumerating Skolem/Herbrand functions does not provide the full count of solutions as it is the case for propositional model counting and counting QBF models at the outer level. We characterized the subset of solutions which can be counted in an enumerative manner. This subset often connects to interesting solutions of a QBF encoding. We also provided the first practical implementation of an enumerative method for level-2 solution counting. Our approach cannot only be used for counting, but also for explicitly enumerating solutions which might also have some applications in better understanding and debugging QBF encodings. We provided several sets of benchmarks of true and false instances to evaluate our implementation.

We consider this work as an important first step to full solution counting, i.e., for QBFs with an arbitrary number of quantifier blocks. While a recursive application of our approach seems possible, it has to be expected that this approach will be inefficient. Hence, alternative ways need to be explored like a tight integration of solving and counting or exploiting approaches that process the prefix in a reverse order as it is for example done in certain preprocessing techniques.

---

## References

- 1 Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S. Meel, and Prateek Saxena. Quantitative verification of neural networks and its security applications. In *Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security*, pages 1249–1264. ACM, 2019.
- 2 Michael Bauland, Elmar Böhler, Nadia Creignou, Steffen Reith, Henning Schnoor, and Heribert Vollmer. Quantified constraints: The complexity of decision and counting for bounded alternation. *Electron. Colloquium Comput. Complex.*, TR05-024, 2005.
- 3 Olaf Beyersdorff, Janota Mikolás, Florian Lonsing, and Martina Seidl. Quantified Boolean Formulas. In *Handbook of Satisfiability*, volume 336, pages 1177–1221. IOS Press, 2021.
- 4 Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling and YaSAT Entering the SAT Competition 2018. In *Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions*, volume B-2018-1 of *Department of Computer Science Series of Publications B*, pages 13–14. University of Helsinki, 2018.
- 5 Fabrizio Biondi, Michael A. Enescu, Annelie Heuser, Axel Legay, Kuldeep S. Meel, and Jean Quilbeuf. Scalable approximation of quantitative information flow in programs. In *Proc. of Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 10747 of *LNCS*, pages 71–93. Springer, 2018.
- 6 Elazar Birnbaum and Eliezer L. Lozinskii. The good old Davis-Putnam procedure helps counting models. *J. Artif. Intell. Res.*, 10:457–477, 1999.
- 7 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. of Int. Joint Conf. on Artificial Intelligence*, pages 3569–3576. IJCAI/AAAI Press, 2016.
- 8 Olivier Dubois. Counting the number of solutions for instances of satisfiability. *Theor. Comput. Sci.*, 81(1):49–64, 1991.
- 9 Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In *Handbook of Satisfiability*, pages 993–1014. IOS Press, 2021.



- 10 Lane A. Hemaspaandra and Heribert Vollmer. The satanic notations: counting classes beyond  $\#P$  and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.
- 11 Hans Kleine Büning and Theodor Lettmann. *Propositional logic - deduction and algorithms*. Cambridge University Press, 1999.
- 12 Elias Kuitert, Sebastian Krieter, Chico Sundermann, Thomas Thüm, and Gunter Saake. Tseitin or Not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses. In *Proc. of the 37th IEEE/ACM Int. Conf. on Automated Software Engineering*. ACM, 2023.
- 13 Richard E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18(6):1087–1097, 1989.
- 14 Massimo Lauria, Jan Elffers, Jakob Nordström, and Marc Vinyals. CNFgen: A Generator of Crafted Benchmarks. In *Proc. of the 20th Int. Conf. on Theory and Applications of Satisfiability Testing*, volume 10491 of *LNCS*, pages 464–473. Springer, 2017.
- 15 Florian Lonsing and Uwe Egly. DepQBF 6.0: A search-based QBF solver beyond traditional QCDCL. In *Proc. of the 26th Conf. on Automated Deduction*, volume 10395 of *LNCS*, pages 371–384. Springer, 2017.
- 16 Nina Narodytska, Aditya A. Shrotri, Kuldeep S. Meel, Alexey Ignatiev, and João Marques-Silva. Assessing heuristic machine learning explanations with model counting. In *Proc. of the Int. Conf. on Theory and Applications of Satisfiability Testing*, volume 11628 of *LNCS*, pages 267–278. Springer, 2019.
- 17 Aina Niemetz, Mathias Preiner, Martina Seidl, and Armin Biere. Resolution-based certificate extraction for QBF - (tool presentation). In *Proc. of the 15th Int. Conference on Theory and Applications of Satisfiability Testing*, volume 7317 of *LNCS*, pages 430–435. Springer, 2012.
- 18 Luca Pulina and Martina Seidl. The 2016 and 2017 QBF solvers evaluations (QBFVAL’16 and QBFVAL’17). *Artif. Intell.*, 274:224–248, 2019.
- 19 Tian Sang, Paul Beame, and Henry A. Kautz. Performing Bayesian inference by weighted model counting. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence*, pages 475–482. AAAI Press / The MIT Press, 2005.
- 20 Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. Ganak: A scalable probabilistic exact model counter. In *Proc. of Int. Joint Conf. on Artificial Intelligence*, pages 1169–1176. Int. Joint Conf. on Artificial Intelligence Organization, 2019.
- 21 Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A survey on applications of quantified Boolean formulas. In *Proc. of the Int. Conf. on Tools with Artificial Intelligence*, pages 78–84. IEEE, 2019.
- 22 Ankit Shukla, Sibylle Möhle, Manuel Kauers, and Martina Seidl. Outercount: A first-level solution-counter for quantified boolean formulas. In *Proc. of the 15th Int. Conf on Intelligent Computer Mathematics*, volume 13467 of *LNCS*, pages 272–284. Springer, 2022.
- 23 G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer, 1983.
- 24 Ziqiao Zhou, Zhiyun Qian, Michael K. Reiter, and Yinqian Zhang. Static evaluation of noninterference using approximate model counting. In *Proc. of IEEE Symposium on Security and Privacy*, pages 514–528. IEEE Computer Society, 2018.