Computing partial hypergraphs of bounded width

Nabil Adrar, Philippe Jégou, Cyril Terrioux

Computing Partial Hypergraphs of Bounded Width*

Nabil Adrar Philippe Jégou Cyril Terrioux
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{firstname.name}@univ-amu.fr

Abstract

In this paper, we are interested in the computation of partial hypergraphs whose width is bounded by an integer k. Given a hypergraph $H = (V, E)$ the task is to find a "partial" hypergraph $H' = (V, E')$ (with $E' \subseteq E$) for which the width of its 2-section is at most $k-1$. Several criteria can be considered for the optimality of the result. The first one is related to the maximum number of hyperedges to include in H' (maximum criteria). Another one is based on the fact that for all $E'' \subseteq E$ with $E' \subsetneq E''$, the width is then strictly greater than $k-1$ (maximal criteria). Unfortunately, each one of these tasks is NP-hard.

So, as this task can be useful in practice for the processing of graphical models (e.g. constraint networks, cost function networks, Bayesian networks, Markov random fields, ...), we propose a polynomial-time algorithm that finds such partial hypergraphs but with no guarantee with respect to optimality. Nevertheless, we show that for an important class of hypergraphs, it can be optimal. Finally, we present experiments performed on a large benchmark containing more than 11,000 instances from different communities. These experiments allow us to evaluate the efficiency of this algorithm from two points of view: first the proximity of the optimality of the result, then its time efficiency in practice.

Keywords: Partial hypergraph, Bounded treewidth

1 Introduction

It is well known that many problems can be represented by graphs, or even beyond, by hypergraphs. To mention only a few domains, it can be for example constraint networks, in the sense of CSP (Constraint Satisfaction Problem), with valuation [1] or not [2, 3], and more generally, graphical models in the sense of cost function networks, Bayesian networks, or Markov Random Fields [4, 5]. On another level, one can also evoke the conjunctive queries in relational databases [6]. In addition to the interest of representing these problems by their structure, this approach makes it possible to use at the same time many works stemming from the (algorithmic) graph theory, both in terms of properties that can be exploited and results in terms of complexity bounds, or even the whole set of algorithms available for the manipulation of these graphs or hypergraphs. And at the level of the most exploited theoretical results to help in the treatment of these graphical models, there is the work around the exploitation of structural properties. Among the different notions developed in this framework, we find mainly the notion of tree-decomposition [7] as well as its numerous variants and extensions which have been proposed in the literature for years while constituting today an extremely active field of work [8, 9, 10, 11, 12]. In this field, tree-decomposition has been exploited both theoretically to provide complexity bounds as well as to define tractable classes using the notion of bounded width [13, 14]. On a practical level, this notion allows offering to process and solving methods that are often very efficient (see [15]), as long as the structure of the processed instances has good properties such as a small width (see [16]). On another level, it has been known for a long time that conjunctive queries in relational databases can be processed in polynomial time when the structure of the concerned relations is defined by an α-acyclic hypergraph [6]. In fact, the link between acyclicity and tree-decomposition is very strong since the notion of treewidth is intended in particular as a kind of measure of the cyclicity of a graph, and beyond, of a hypergraph. But the link is even stronger formally. Indeed, it is well known that if we consider a tree-decomposition, then the hypergraph defined by the set of vertices and taking as hyperedges the bags of the decomposition is an α-acyclic hypergraph.

However, while tree-decomposition provides a very useful tool on the theoretical level, there are often difficulties on the practical level. On the one hand, the calculation of an optimal tree-decomposition, i.e. whose width is equal to the treewidth, is an NP-hard problem [17] if this treewidth is not bounded by a constant. On the other hand, depending on the instances, as soon as the treewidth is too large, it becomes difficult to exploit the solving methods based on the exploitation of the structure by this kind of approach.

Therefore, alternative approaches have been proposed to get around this type of difficulty. About the treatment of acyclic hypergraphs, Hirata et al. were interested in such a question [18]. They studied several questions including the

*The final publication is available at https://doi.org/10.1016/j.dam.2022.12.025.

problem called "Spanning connected Acyclic Subhypergraph" which consists, given a hypergraph H, in determining if there exists a partial α-acyclic hypergraph that is connected and covering all the vertices of H. They show that this problem is NP-complete. They also studied the so-called "Maximum Acyclic Subhypergraph Problem", itself NP-complete. In this problem, given a hypergraph H and an integer k, the question is whether there exists a partial sub-hypergraph (subsets of vertices and hyperedges) of k hyperedges that is α-acyclic. Because of these two negative results, they became interested in the problem which consists, given a hypergraph H, in calculating a maximum α-acyclic partial subhypergraph in the sense of the set of hyperedges. They propose for this purpose a linear time algorithm based on the algorithm of Tarjan and Yannakakis [19] that deals with the question of the recognition of α-acyclic hypergraphs and chordal graphs.

Within the framework of CSPs, Jégou and Terrioux have been interested in the search sub-networks of binary CSPs of limited width in order to propose new filtering techniques based on the internal structural properties of the constraint networks [20]. This approach was essentially guided by the study and exploitation of substructures in terms of filtering on the basis of a problem that is relaxed because it contains only a subset of the constraints of the problem to be solved. Such an approach gave interesting experimental results for this task. To this end, they dealt with the question of the calculation of sub-networks by introducing the notion of w-PST which corresponds to "partial spanning tree-decompositions" of width w. This question not being central in their work, the calculation was limited to a heuristic constructing from the input graph, a partial graph constituted by a partial k-tree [21].

It is in the Markov random fields framework that the issue seems to have been most studied. But in a way, as with the work on CSPs, the value of the work carried out is to be assessed in terms of its contribution to the field of application, i.e. the practical efficiency of the treatments of Markov random fields. Karger and Srebro [22] are interested in the problem called "Maximal Hypertree problem" which, given an integer k, a set of vertices V, and a weight function with real values on hyperedges of size at most $k+1$, seeks to find a hypertree H of treewidth at most k which minimizes the sum of the weights of the hyperedges of H. In fact, hypertree is to be understood as tree-decomposition. They show (the proofs are to be found in [23]) that the (associated decision) problem restricted to graphs is NP-complete. Moreover, the associated optimization problem belongs to the class max-SNP-hard. They also specify that even with unit weights, this problem remains NP-hard. Beyond this theoretical study, their main contribution consists in giving the first approximation algorithms for the problem, achieving a polynomial-time constant-factor approximation for any fixed treewidth objective.

More recently, still in the context of Markov random fields, several works have taken an interest in this question, such as [24] or [25]. In [25], Fix et al. study the problem called "Maximum Bounded-Treewidth Subgraph problem" which, given a graph with weights on the edges, consists in finding a subgraph of treewidth at most k and of maximum weight. To solve it, they propose a greedy algorithm to find a subgraph of treewidth k and large weight. In addition to the quality of their algorithm, they show its interest in terms of its practical use, and this was its main objective. To do this, their algorithm exactly solves the inference problem in the subgraph using dynamic programming. Then, using a proof of lower bound, the authors argue that the solution for the subgraph is a good approximation to the optimum for the original energy.

This type of work has also been developed in the management of Bayesian networks. In [26], Nie et al. propose an algorithm to find k-trees with maximum informative scores, which is a measure of quality for the k-tree in yielding good Bayesian networks. The algorithm achieves close to optimal performance compared to exact solutions in small domains and can discover better networks than existing approximate methods can in large domains. It also provides an optimal elimination order of variables that guarantees small complexity for later runs of exact inference.

In this paper, we are interested in the question specifically expressed in terms of hypergraphs. Indeed, if the literature has a very large number of graph processing algorithms, it seems necessary to enrich this corpus with algorithms that directly process hypergraphs. In this work, we seek to calculate a partial hypergraph whose width, in the sense of tree-decomposition, is bounded by a constant k. It turns out that this problem is NP-hard if one is interested in the maximum number of hyperedges to be selected. But it also turns out to be NP-hard if we limit ourselves to a maximal set in the sense of inclusion. So, we propose a polynomial-time algorithm called k-PH (for Partial Hypergraph of width $k-1$) which looks neither for optimality in terms of the number of hyperedges, nor for maximality in set terms (i.e. for inclusion). However, its interest is already justified on a theoretical level. Indeed, we show that for the case where the input hypergraph is α-acyclic, and if the value of the parameter k is equal to the size of the largest hyperedge, then the algorithm calculates the optimal result. But the interest of this algorithm is also shown experimentally.

To this end, we conducted experiments on a large benchmark containing 11,302 instances from different communities and whose treewidth is known, and we consider the proportion of hyperedges included in the calculated partial hypergraph. For example, these experiments show that for about 90% of the instances, more than 70% of the hyperedges are retained in the calculated partial hypergraph, and more than 80% of the hyperedges are retained in 75% of the instances. These experiments also make it possible to verify that calculation times remain limited because they are on average well below one second on a basic computer. Finally, this algorithm can easily be adapted according to the considered objective. Indeed, one can thus add to it different heuristics which make it possible, for example, to take into account weights for hyperedge to be selected in the partial hypergraph. This should make it easier to process different types of graphical models.

In the following section, we introduce the notations and we formally specify the different problems associated with this

type of question. Section 3 presents the scheme of the k-PH algorithm and after having provided proof of its validity, we show its optimality for the case of α-acyclic hypergraphs. We give an evaluation of the time complexity of this algorithm but its analysis needs to presents the details of the implementation. So, these details are given in Appendix A. The penultimate section provides an experimental study to assess the quality of the results as well as the runtimes. Finally, we conclude and give some perspectives in Section 5.

2 Preliminaries

A hypergraph $H = (V, E)$ is defined by a set V of vertices and a set $E \subseteq 2^V$ of hyperedges. The size of V is denoted n while the size of E is e. For $v \in V$, $d(v)$ is the degree of v, that is the number of hyperedges containing v while r is the rank of the hypergraph (i.e. the maximum number of vertices per hyperedge).

Definition 1 (subhypergraph) *Given a hypergraph $H = (V, E)$ and $V' \subseteq V$, the* subhypergraph *of H induced by V' is the hypergraph $H[V'] = (V', E')$, where $E' = \{E_i \in E \mid E_i \subseteq V'\}$.*

Note that all hyperedges appearing in $H[V']$ must be included in V'. It is not the case for what we call *expanded subhypergraph*:

Definition 2 (expanded subhypergraph) *Given a hypergraph $H = (V, E)$ and $V' \subseteq V$, the* expanded subhypergraph *of H induced by V' is the hypergraph $H[[V']] = (V', E')$, where $E' = \{E_i \subseteq V' \mid \exists E_j \in E, E_j \cap V' \neq \emptyset, E_i = E_j \cap V'$ and E_i is maximal$\}$.*

In this definition, *maximal* means that there is no $E_{i'}$ satisfying the same conditions as E_i such that $E_i \subsetneq E_{i'}$. So, by verifying this condition, in an expanded subhypergraph, if a hyperedge does not appear in $H[V']$ but has a non-empty intersection with V', $H[[V']]$ contains as hyperedge this maximal intersection.

Definition 3 (partial hypergraph) *Given a hypergraph $H = (V, E)$ and $E' \subseteq E$, the* partial hypergraph *of H induced by E' is the hypergraph $H[E'] = (V, E')$.*

To define the width of a hypergraph, we recall the definition of the 2-section [27] of a hypergraph (note that this graph is sometimes called the *primal graph* [3]):

Definition 4 (2-section of hypergraph) *Given a hypergraph $H = (V, E)$, the* 2-section *of H is the graph $2_{SEC}(H) = (V, E')$ where an edge $\{x, y\} \in E'$ if and only if there is a hyperedge $E_i \in E$ such that $\{x, y\} \subseteq E_i$.*

Given a hypergraph $H = (V, E)$, a path of length ℓ in H between two vertices x and y is a sequence of vertices $(x = v_0, v_1, \ldots, v_\ell = y)$ such that $\forall i, 1 \leq i \leq \ell, \exists E_i \in E$ such that $\{v_{i-1}, v_i\} \subseteq E_i$. We can see that there is a path in H if and only if there is a path in $2_{SEC}(H)$. In the sequel, we call *connected component* of a hypergraph $H = (V, E)$ a subset of vertices[1] of V which is a connected component in $2_{SEC}(H)$. In other words, two vertices of H appear in the same connected component if there is a path between them in H, as for graphs. So, if $V' \subseteq V$ is a connected component of the hypergraph $H = (V, E)$, for all hyperedge $E_i \in E$, either $E_i \subseteq V'$, or $E_i \cap V' = \emptyset$.

To define the tree-decomposition and the treewidth of a hypergraph, we need to use the corresponding definitions for graphs [7]:

Definition 5 (tree-decomposition and treewidth of hypergraph) *A* tree-decomposition *of a graph $G = (V, E)$ is a pair (B, T) where $T = (I, F)$ is a tree (I is a set of nodes and F a set of edges) and $B = \{B_i : i \in I\}$ a family of subsets of V such that every $B_i \in B$ (called* bag*) corresponds to a node i of T and satisfies:*

(i) $\cup_{i \in I} B_i = V$,

(ii) $\forall \{x, y\} \in E$, $\exists i \in I$ *such that* $\{x, y\} \subseteq B_i$, *and*

(iii) $\forall i, j, k \in I$, *if k is on a path between i and j in T, then* $B_i \cap B_j \subseteq B_k$

The width *of a tree-decomposition is equal to* $max_{i \in I}|B_i| - 1$. *The* treewidth *of G denoted w is equal to the minimum width among all the tree-decompositions of G. So, given a hypergraph $H = (V, E)$ and its 2-section $2_{SEC}(H)$, a* tree-decomposition *of H is a tree-decomposition of $2_{SEC}(H)$, and the* treewidth *of H is the treewidth of $2_{SEC}(H)$.*

[1] Usually, a connected component in a graph is defined as the subgraph induced by the vertices that compose it. Here, for the case of hypergraphs, we assimilate a connected component to the set of vertices that compose it because this simplifies the notations without reducing their precision.

In this paper, we study the issue of calculating a partial hypergraph that includes as many hyperedges as possible while guaranteeing a bounded width. Also, we are interested in the question of a maximum (size) set of hyperedges or, at least, maximal (for inclusion). This problem is based on the following definitions.

Definition 6 (k partial hypergraph) *Given a hypergraph $H = (V, E)$ and an integer k, a k* partial hypergraph of H *is a hypergraph $H' = (V, E')$ with $E' \subseteq E$, such that $2_{SEC}(H')$ has a treewidth at most $k-1$. A k partial hypergraph H' of H is* maximal *if there is no E'' such that $E' \subsetneq E'' \subseteq E$ and such that the treewidth of $H'' = (V, E'')$ is at most equal to $k-1$. It is* maximum *if it is a partial hypergraph of treewidth at most $k-1$ with as many hyperedges as possible.*

Note that for a hypergraph H and an integer k, it is possible that no partial hypergraph H' exists which has a width exactly equal to $k-1$. Indeed, if a hypergraph contains two edges, one of size $k-2$ and the other of size $k+1$, it does not have any partial hypergraph whose treewidth is equal to $k-1$. Before considering the associated optimization problems, we define a first decision problem:

Maximum Partial Hypergraph of Bounded Width
Instance: A hypergraph $H = (V, E)$, an integer $M \leq |E|$ and an integer $k \leq |V|$.
Question: Does H have a partial hypergraph with M hyperedges whose treewidth is at most $k-1$?

In fact, this problem has already been studied in the case of Markov random fields under consideration of weighted hyperedges and as an optimization problem called Maximal Hypertree [22, 23]:

Maximal Hypertree
Instance: A weighted hypergraph $H = (V, E, w)$ with $w : E \to \mathbb{R}$ and an integer $k \leq |V|$.
Question: Find a partial hypergraph of H for which the sum of the weights of its hyperedges is maximum and whose treewidth is at most $k-1$.

In [22], it is shown that this optimization problem is max-SNP-hard and that the associated decision problem restricted to graphs is NP-complete. The authors specify that even with unit weights, this problem is NP-hard. The details of the proofs are to be found in [23]. This being, in fact, we can deal with two questions here, depending on whether we are interested in the maximum in terms of weights (or number of hyperedges for unit weights), or in terms of the maximality of the set of the selected hyperedges. We present these two variants:

Maximal Partial Hypergraph of Bounded Width
Instance: A hypergraph $H = (V, E)$ and an integer $k \leq |V|$.
Question: Find a maximal partial hypergraph of H whose treewidth is at most $k-1$.

It can be observed that the associated decision problem is trivial because, except for having only hyperedges of which size is strictly greater than k, the answer to the question of the existence of a partial hypergraph of treewidth at most $k-1$ is always yes. It is then necessary to reformulate the question by removing "maximal", and by replacing "of treewidth at most $k-1$" by "of treewidth $k-1$":

Partial Hypergraph of Given Width
Instance: A hypergraph $H = (V, E)$ and an integer $k \leq |V|$.
Question: Does H have a partial hypergraph of treewidth $k-1$?

Unfortunately, this problem is as difficult as the problems associated with the tree-decomposition and treewidth in graphs. And we know that given a graph G and an integer k, it is NP-complete to determine whether the treewidth of G is at most k [17]. So, to determine whether a graph G have a partial graph of treewidth $k-1$ is also NP-complete and thus, Partial Hypergraph of Given Width is NP-complete. Another immediate consequence is that Maximal Partial Hypergraph of Given Width is NP-hard. Finally, it is useful to recall, as mentioned above, that for a given hypergraph and an integer k, there may exist a partial hypergraph of treewidth $k-2$ while there exists no partial hypergraph of treewidth $k-1$. Therefore, in the following (see Section 3), we will be interested in finding partial hypergraphs whose treewidth is bounded by $k-1$ rather than of treewidth $k-1$ exactly.

In the following section, we propose a polynomial-time algorithm, called k-PH, which, given a hypergraph and an integer k, calculates a partial hypergraph whose width is at most $k-1$, that is a k partial hypergraph. We show that this algorithm is of theoretical interest since it guarantees optimality of the result for the case of α-acyclic hypergraphs [6] (see Theorem 2). Before that, we recall here some properties related to the notion of acyclic hypergraphs.

Definition 7 (chordal graph [28]) *A graph is* chordal *(or* triangulated*) if it contains no chordless induced cycle of length 4 or more, a chord being an edge joining two non consecutive vertices along a path.*

It has been shown by Dirac [29] that a graph is chordal if and only if all its minimal separators are cliques. Chordal graphs can also be characterized by means of perfect elimination orderings:

Definition 8 (perfect elimination ordering) *Given a graph $G = (V, E)$, a* perfect elimination ordering *on V is a function $\sigma : V \rightarrow [1, n]$ such that $N^+_\sigma(x)$ is a clique for every $x \in V$, where $N^+_\sigma(x) = \{y \in V \mid \{x, y\} \in E$ and $\sigma(x) < \sigma(y)\}$.*

In fact, this ordering σ corresponds to a numbering of the vertices from 1 to n. So, in the following, we will speak about numbering to evoke this ordering. It has been shown by Fulkerson and Gross [30] that a graph is chordal if and only if it admits a perfect elimination ordering. We can now define the notion of acyclicity in hypergraphs. It should be noted that different types of acyclicity have been defined such as Berge-acyclicity [31], γ-acyclicity [32], β-acyclicity [32] and α-acyclicity [6], but the most usable and used in practice is clearly the last one:

Definition 9 (α-acyclicity) *A hypergraph $H = (V, E)$ is α-acyclic if $2_{SEC}(H)$ is chordal and H is conformal, that is if for any clique $X \subseteq V$ of $2_{SEC}(H)$, there is at least one hyperedge $E_i \in E$ such that $X \subseteq E_i$.*

It should be noted that the relationship between hypergraph α-acyclicity and tree-decomposition are very close. Indeed, for any graph $G = (V, E)$, for any tree-decomposition of G, it is well known that the set B of bags allows defining a hypergraph $H = (V, B)$ which is α-acyclic.

Finally, in the next section, we use the notion of neighborhood of a vertex and neighborhood of sets of vertices in a hypergraph. We give below the associated definitions.

Definition 10 (neighborhood in a hypergraph) *Let $H = (V, E)$ be a hypergraph. For $x \in V$, the* neighborhood *$N_H(x)$ of x in H is defined as $\{y \in V \mid \exists E_i \in E, x, y \in E_i\}$. For $X \subseteq V$, the* neighborhood *$N_H(X)$ of X in H is defined as $\cup_{x \in X} N_H(x)$. The subset of vertices of a set Y which are neighbors of a set of vertices X is denoted $N_H(X, Y) = \{y \in Y \mid \exists x \in X : \exists E_j \in E, \{x, y\} \subseteq E_j\}$.*

It is obvious that $N_H(X, \emptyset) = \emptyset$.

3 Finding a Partial Hypergraph of Bounded Width

In this section, we present the algorithm k-PH (for Partial Hypergraph parametrized by an integer k). Given a hypergraph $H = (V, E)$ and an integer k, this algorithm computes a partial hypergraph $H' = (V, E')$ of H and a tree-decomposition of its 2-section whose width is at most $k - 1$. This tree-decomposition will be represented by the set B of its bags, that is $B = \{B_1, B_2, \dots\}$. In practice, B is constructed bag by bag and, at the end, E' is the set of hyperedges of H which are covered by a bag B_i. Roughly speaking, the bag B_i is built by selecting some hyperedges of H' (B_i is then defined as the union of these hyperedges) in such a way that the size of B_i does not exceed k. This computation only depends on the previously built bags and the way the hyperedges are selected. So, the hyperedge selection is an important step in the construction of H' and its associated decomposition. Since our algorithm does not aim to guarantee to obtain an optimal solution, this step is achieved thanks to a heuristic $\mathcal{H}$ that constitutes the third input argument of k-PH. Note that $\mathcal{H}$ only relies on the previously built bags and the current value of H'. Once B_i is built, all the hyperedges of H' that call into question the validity of the built tree-decomposition are removed from E'.

Now, we give a detailed description of this algorithm and all the arguments to ensure its correctness. The time complexity is given in this section. However, our analysis requires a precise presentation of the implementation of the algorithm (e.g. the data structures used in the algorithm to ensure its efficiency). So, these elements of complexity analysis are presented in great detail in Appendix A. Then, to facilitate the understanding of k-PH, we describe it by considering the $\mathcal{H}_\alpha$ heuristic which will be also used in the proof of Theorem 3. Finally, without loss of generality, we consider that the rank of the hypergraph is at most k, that is no hyperedge of H has a size strictly greater than k. If some hyperedges do not satisfy this condition, we only consider the partial hypergraph of H induced by the removal of these hyperedges. Moreover, we assume that any vertex of this hypergraph belongs to at least one hyperedge.

Initially, H' is equal to H (line 1) and it is computed by gradually removing some of its hyperedges. The obtained tree-decomposition of $2_{SEC}(H')$ is represented by a set of bags called B (initially, $B = \emptyset$, line 2) which is also computed by gradually adding at each step a new bag denoted B_i. As it is possible that H is not connected, we first compute its connected components $X_1, X_2, \dots X_{n_0}$ using the *Comp-CC* algorithm[2]. This algorithm takes as inputs the current partial

[2] A detailed description of this algorithm is given in Appendix A. Although the calculation of the connected components in a hypergraph is very simple, we prefer to describe this algorithm because *Comp-CC* also maintains data structures allowing to ensure the efficiency of k-PH.

Algorithm 1: *k-PH*

Input: A hypergraph $H = (V, E)$, an integer k and a heuristic $\mathcal{H}$.
Output: A partial hypergraph $H' = (V, E')$ of H and a tree-decomposition of $2_{SEC}(H')$ represented by B (a set of bags) whose width is at most $k - 1$.

1 $H' = (V, E') \leftarrow (V, E)$
2 $B \leftarrow \emptyset$
3 $i \leftarrow 1$
4 $Q \leftarrow \emptyset$
5 $Enqueue(Q, Comp\text{-}CC(i, H', V, \emptyset))$
6 **while** $Q \neq \emptyset$ **do**
7 $\quad X_i \leftarrow Dequeue(Q)$
8 $\quad B_i \leftarrow \emptyset$
9 $\quad N_i \leftarrow N_{H'}(X_i, \cup_{1 \leq j \leq i-1} B_j)$
10 $\quad$ **if** $\mathcal{H} = \mathcal{H}_\alpha$ **then**
11 $\quad\quad$ **if** $N_i = \emptyset$ **then**
12 $\quad\quad\quad$ Sort the hyperedges $E_j \in E'$ such that $E_j \subseteq X_i$ in descending order of their size
13 $\quad\quad\quad$ Visit this ordered list and insert the current hyperedge E_j into B_i if $|B_i \cup E_j| \leq k$
14 $\quad\quad$ **else**
15 $\quad\quad\quad$ $N_{H'}(N_i, X_i) \leftarrow \{y \in X_i \mid \exists x \in N_i : \exists E_j \in E', \{x, y\} \subseteq E_j\}$
16 $\quad\quad\quad$ $S_i \leftarrow \{E_j \in E' \mid E_j \subseteq N_i \cup N_{H'}(N_i, X_i)\}$
17 $\quad\quad\quad$ Select $B_\ell \in B$ such that $|B_\ell \cap N_i|$ is maximum
18 $\quad\quad\quad$ Sort the hyperedges $E_j \in S_i$ such that $E_j \not\subseteq N_i$ and $E_j \not\subseteq N_{H'}(N_i, X_i)$ in descending order of the size of $|B_\ell \cap E_j|$
19 $\quad\quad\quad$ Visit this ordered list and insert the current hyperedge E_j into B_i if $|B_i \cup E_j| \leq k$
20 $\quad\quad\quad$ Insert in B_i the hyperedges $E_j \in S_i$ such that $E_j \subseteq B_i$ which were not selected in the previous step
21 $\quad\quad\quad$ Remove from H' the hyperedges $E_j \in S_i$ such that $E_j \not\subseteq N_i$ and $E_j \not\subseteq N_{H'}(N_i, X_i)$ and $E_j \not\subseteq B_i$
22 $\quad B \leftarrow B \cup \{B_i\}$
23 $\quad Enqueue(Q, Comp\text{-}CC(i, H', X_i, B_i))$
24 $\quad i \leftarrow i + 1$
25 For each vertex v that does not appear in any bag from B, add the bag $\{v\}$

hypergraph H', a set of vertices X and another set of vertices Y. *Comp-CC* returns the set of connected components of the expanded subhypergraph of H' induced by $X \backslash Y$, i.e. $H[[X \backslash Y]]$. These components are then inserted into the queue Q (line 5).

Once these preliminary processes have been realized, at each new step of the loop in line 6, a connected component X_i is removed from the queue Q to build a new bag B_i (line 7). So, before, B_i is initialized to the empty set (line 8). This new bag will be added to the tree-decomposition already computed (current set of bags B), and H' will be updated by removing the hyperedges whose conservation would distort the tree-decomposition. Considering the connected component X_i removed from the queue Q, the set of nodes N_i is initialized (line 9). N_i is the set of vertices that belong to at least one of the bags in $B = \{B_1, B_2, \ldots, B_{i-1}\}$ and that are adjacent to at least one vertex of X_i in H'. Formally, $N_i = N_{H'}(X_i, \cup_{1 \leq j \leq i-1} B_j)$ (see Figure 1 and recall Definition 10). There are then two cases to consider for the connections in the hypergraph H':

(1) (Basic case) Either X_i has no neighbor in N_i (this is the case, for instance, in the first step since the set N_i is empty because $B = \emptyset$);

(2) (General case) Or X_i has at least one neighbor in N_i.

In case (1), we proceed by searching within X_i a new bag B_i. This bag is obtained by computing a set of hyperedges included in X_i whose union is equal to the bag B_i and such that $|B_i| \leq k$. Such a bag is found using a given heuristic $\mathcal{H}$. In the description of *k-PH* presented in Algorithm 1, this step is illustrated with the $\mathcal{H}_\alpha$ heuristic (lines 12 and 13). For this case, no hyperedge has to be deleted from H'.

Case (2) is the general case and it is more intricate. The new bag B_i is calculated in the neighborhood of an already computed bag B_ℓ ($1 \leq \ell < i$) such that $B_i \cap B_\ell = N_i \cap B_\ell \neq \emptyset$. This guarantees that we obtain a tree-decomposition because any new bag B_i is linked to at least one bag $B_\ell \in \{B_1, B_2, \ldots, B_{i-1}\}$ and such that $(\cup_{1 \leq j \leq i-1} B_j) \cap B_i \subseteq (B_\ell \cap B_i)$. This condition allows us to verify that the computed set of bags can be structured as a tree. Moreover, it allows satisfying the third condition (condition (*iii*)) of tree-decompositions which deals with the links between bags. About the bag B_ℓ, note that there are two options. Either this bag B_ℓ can be pre-determined and then B_i is defined with respect to B_ℓ, or, on the contrary, B_ℓ is determined with respect to the new bag B_i. This choice depends on the used heuristic $\mathcal{H}$. Several heuristics are conceivable but we consider here as a reference and as for case 1, the $\mathcal{H}_\alpha$ heuristic, which is a very simple and natural heuristic that chooses first as bag B_ℓ the one with the largest intersection with N_i and then, chooses as the first hyperedge to be in B_i the one that shares the maximum number of vertices with B_ℓ (the next hyperedges are taken with the same principle, in descending order of the size of their intersection with B_ℓ).

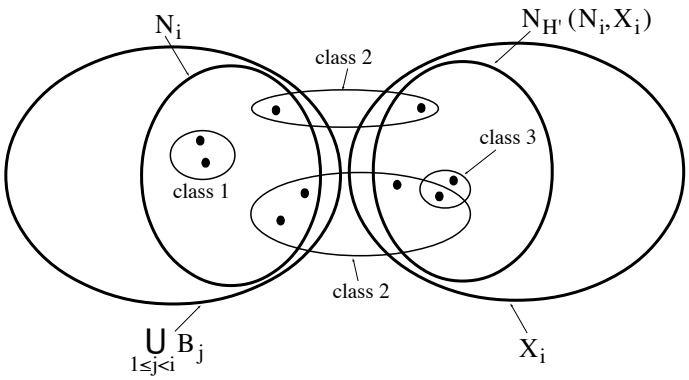


Figure 1: Beginning of a new bag building stage with the representation of the sets $\cup_{1\leq j\leq i-1}B_j, N_i, X_i$ and $N_{H'}(N_i, X_i)$. 4 hyperedges illustrate the 3 possible classes of hyperedges inside $N_i \cap N_{H'}(N_i, X_i)$.

The set of hyperedges that are candidates for the construction of B_i is denoted S_i and is defined by $S_i = \{E_j \in E' \mid E_j \subseteq N_i \cup N_{H'}(N_i, X_i)\}$. To compute B_i, some hyperedges from S_i are selected and they constitute a set $S'_i \subseteq S_i$. Thus, B_i is obtained by the union of the hyperedges from S'_i, i.e. $B_i = \cup_{E_j \in S'_i} E_j$. So, we analyze the hyperedges E_j of S_i. We have three classes of hyperedges:

(1) $E_j \subseteq N_i$ and therefore $E_j \cap N_{H'}(N_i, X_i) = \emptyset$. In this case, the hyperedge E_j appears in N_i, thus in at least one bag already computed.

(2) $E_j \not\subseteq N_i$ and $E_j \not\subseteq N_{H'}(N_i, X_i)$. Thus $E_j \cap N_i \neq \emptyset$ and $E_j \cap N_{H'}(N_i, X_i) \neq \emptyset$. In this case, the hyperedge E_j overlaps N_i and $N_{H'}(N_i, X_i)$.

(3) $E_j \subseteq N_{H'}(N_i, X_i) \subseteq X_i$ and thus $E_j \cap N_i = \emptyset$.

Figure 1 shows the sets $\cup_{1\leq j\leq i-1}B_j, N_i, X_i$ and $N_{H'}(N_i, X_i)$, and indicates the location of the hyperedges in relation to the class to which they belong.

The hyperedges E_j of class (1) have already been treated during the calculation of the previous bags. They can therefore be ignored and they will appear in the hypergraph H' at the end of processing. They will therefore not be taken into account in the calculation of B_i even if they may eventually be included inside. Likewise, calculating B_i, we will not consider the hyperedges of class (3). However, some of these could also be included in B_i. Thus, only the hyperedges of class (2) will be explicitly considered for the calculation of B_i. We analyze this case.

B_i will be defined as $\cup_{E_j \in S'_i} E_j$. So, when selecting a subset S'_i of hyperedges of S_i, a first condition to satisfy is related to the bound of the width and therefore we must calculate a subset S'_i of S_i such that $|B_i| \leq k$. In addition, we must impose that $B_i \cap N_{H'}(N_i, X_i) \neq \emptyset$, that is B_i must include at least one vertex which does not appear in the bags already computed. This second condition is necessary because one must ensure that the size of the tree-decomposition grows strictly during the calculation (i.e. at each step of the loop) so as to guarantee the termination of the algorithm. So, by increasing the size of the tree-decomposition, we mean the number of bags of the tree-decomposition, and thus, the number of new vertices appearing in at least one bag of B. Note that this second condition necessarily holds since only class (2) hyperedges are taken into account and each of them contains at least one vertex that does not appear in an already calculated bag. A third condition must be verified to make sure that there exists an already built bag B_ℓ, with $1 \leq \ell < i$, such that $(\cup_{1\leq j\leq i-1}B_j) \cap B_i \subseteq (B_\ell \cap B_i)$. By fulfilling these three conditions, we just have to choose, with the heuristic, hyperedges of S_i which will define the set S'_i. Recall that we suppose that this heuristic takes as a bag B_ℓ the one that has the largest intersection with N_i. Figure 2 shows different possibilities for the selection of hyperedges in a new bag.

Once S'_i is calculated, H' has to be updated (line 21). Only hyperedges $E_j \in S_i$ belonging to the class (2) such that $E_j \not\subset B_i$ may be removed. Indeed, all hyperedges from class (1) are preserved because they belong to bags already computed. Hyperedges from classes (3) are preserved too: if they are included in B_i, they must be retained; and it is also the case if they are not included in B_i because they will be considered later. It is the same for hyperedges E_j from class (2) such that $E_j \subseteq B_i$. The hyperedges $E_j \in S_i$ that must be deleted are exactly such that $E_j \cap N_i \not\subset B_i$ and $E_j \cap B_i \not\subset N_i$. In other words, these are the hyperedges that have disjoint intersections with at least two bags already constructed, because including them later in a bag will prevent us from maintaining the tree structure of B.

After updating H' (for case 2) and computing B_i, we calculate the connected components of $H'[[X_i \backslash B_i]]$ (line 23), that is $X_{i_1}, X_{i_2}, \ldots X_{i_{n_i}}$ which are then be inserted into Q. This task is repeated until the queue Q is empty.

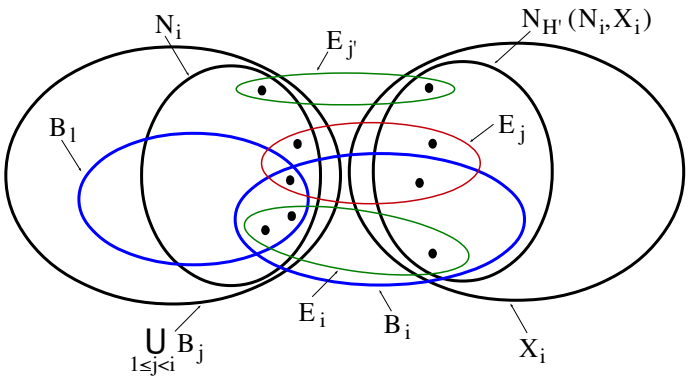


Figure 2: Bag building stage. Bags B_ℓ and B_i are in blue. Among the 3 hyperedges of class 2, E_j must be deleted while E_i will be preserved in the final partial hypergraph and $E_{j'}$ is temporary at least preserved.

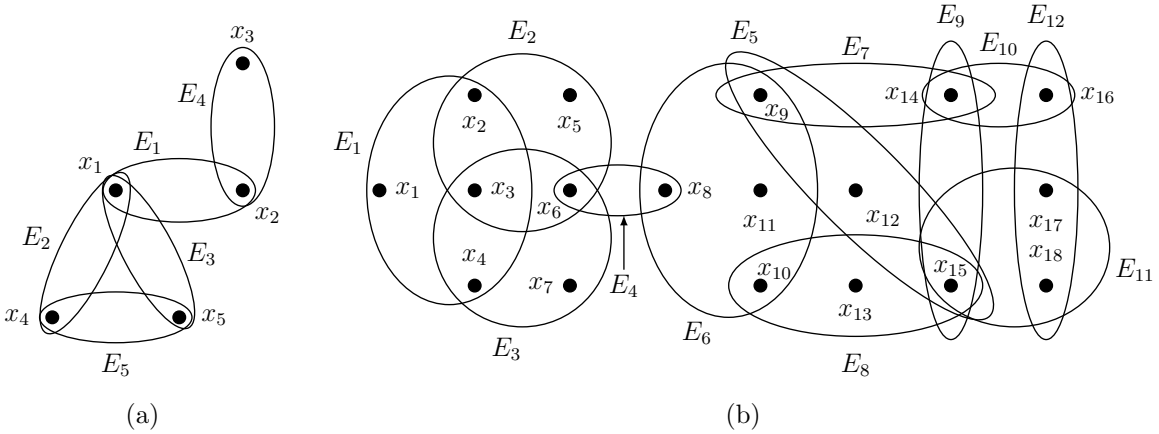


Figure 3: Illustration of how k-PH works on two possible instances: a graph with $k = 2$ (a) and a hypergraph with $k = 5$ (b).

Note that at the end of the **while** loop, some vertices may not have been processed. Indeed, this case is possible if some vertices only appear in the hyperedges deleted during the construction of H'. So, none of these vertices can be included in a bag when constructing B during the loop. This is why, at the end of processing (line 25), specific bags are defined for each of these vertices, so as to ensure that B will correspond to the bags of a tree-decomposition of $2_{SEC}(H')$ since every vertex must appear in at least one bag. Note that if at the end of the process, H' is not connected, we finally obtain a collection of trees, and thus a forest of tree-decompositions.

We illustrate how k-PH with the heuristic $\mathcal{H}_\alpha$ works on the two instances of Figure 3. First, we apply k-PH with $k = 2$ on the graph $(V = \{x_1, \ldots, x_5\}, E = \{E_1, \ldots, E_5\})$ depicted in Figure 3(a). Initially, X_1 corresponds to V since the graph is connected, $N_1 = \emptyset$ and any edge can be selected by the heuristic. In line 13 of k-PH, suppose that the first chosen edge is $E_1 = \{x_1, x_2\}$. It ensues that $B_1 = \{x_1, x_2\}$. The call of *Comp-CC* then builds two connected components, $\{x_3\}$ and $\{x_4, x_5\}$, which will be inserted in Q. In the next step, suppose that $\{x_3\}$ is removed from Q and so that $X_2 = \{x_3\}$. In this case, line 9 calculates $N_2 = \{x_2\}$ and line 14 is executed. So we have $N_{H'}(N_2, X_2) = \{x_3\}$ (line 15) and $S_2 = \{E_4\}$ (line 16). It ensues that we have necessarily $B_l = B_1$. Only the edge E_4 (of class 2) is then considered and we have $B_2 = E_4$. No edge is added to B_2 in line 20, nor deleted from H' in line 21. At this point, no new connected component is found by *Comp-CC* and therefore, nothing will be added in Q. In the next step, $X_3 = \{x_4, x_5\}$ is removed from Q. Line 9 computes $N_3 = \{x_1\}$, and then line 14 is executed. We obtain $N_{H'}(N_3, X_3) = \{x_4, x_5\}$ (line 15), and then $S_3 = \{E_2, E_3, E_5\}$ (line 16) where E_2 and E_3 are of class 2 and E_5 of class 3. So, only E_2 or E_3 can be chosen in line 19. Assume that E_2 is chosen and therefore $B_3 = \{x_1, x_4\}$. No edge is added to B_2 in line 20, nor deleted from H' in line 21. The call of *Comp-CC* on line 23 calculates a new connected component $\{x_5\}$ which is then inserted in Q. The while loop therefore continues with $X_4 = \{x_5\}$, then line 9 calculates $N_4 = \{x_1, x_4\}$. Again line 14 is executed. Line 15 calculates $N_{H'}(N_4, X_4) = \{E_3, E_5\}$, then line 16 obtains $S_4 = \{E_3, E_5\}$. There are then two possibilities for B_ℓ, either $B_\ell = B_1$, or $B_\ell = B_3$ (line 17). If $B_\ell = B_1$, E_3 is necessarily used to build B_4, and no edge will be added to it (line 20). Then, E_5 is removed from H' since we have $E_5 \not\subseteq N_4$, $E_5 \not\subseteq N_{H'}(N_4, X_4)$ and $E_5 \not\subseteq B_4$. Running *Comp-CC* (line 23) will not compute a new connected component, and since Q is empty, the algorithm will stop. We notice that if $B_\ell = B_3$, the algorithm will calculate $B_4 = E_5$ because E_5 will be kept but E_3 deleted from H'.

Now, let us consider the hypergraph ($V = \{x_1, \ldots, x_{18}\}, E = \{E_1, \ldots, E_{12}\}$) of Figure 3(b). We illustrate the execution of k-PH over it when k is set to 5 thanks to Table 1. This table specifies the state of the various objects manipulated at each stage. Let us note that, unlike the first example, and because of the deletion of the hyperedge E_3 from H', a vertex will be isolated, namely the vertex x_7 which belongs to E_3 but which does not appear in any hyperedge kept in H'. Also, the bag B_7 which contains only x_7 is calculated on Line 25 of k-PH.

i	X_i	N_i	$N_{H'}(N_i, X_i)$	S_i	B_ℓ	B_i	H'	Q
-	-	-	-	-	-	-	(V, E)	$[V]$
1	V	$\emptyset$	-	-	-	$E_6 \cup E_4 = \{x_6, x_8, \ldots, x_{11}\}$	(V, E)	$[\{x_{12}, \ldots, x_{18}\}, \{x_1, \ldots, x_5, x_7\}]$
2	$\{x_{12}, \ldots, x_{18}\}$	$\{x_9, x_{10}\}$	$\{x_{12}, \ldots, x_{15}\}$	$\{E_5, E_7, E_8, E_9\}$	B_1	$E_5 \cup E_8 = \{x_9, x_{10}, x_{12}, x_{13}, x_{15}\}$	(V, E)	$[\{x_1, \ldots, x_5, x_7\}, \{x_{14}, x_{16}, x_{17}, x_{18}\}]$
3	$\{x_1, \ldots, x_5, x_7\}$	$\{x_6\}$	$\{x_2, \ldots, x_5, x_7\}$	$\{E_2, E_3\}$	B_1	$E_2 = \{x_2, x_3, x_5, x_6\}$	(V, E)	$[\{x_{14}, x_{16}, x_{17}, x_{18}\}, \{x_1, x_4, x_7\}]$
4	$\{x_{14}, x_{16}, x_{17}, x_{18}\}$	$\{x_9, x_{15}\}$	$\{x_{14}, x_{17}, x_{18}\}$	$\{E_7, E_9, E_{11}\}$	B_2	$E_7 \cup E_9 \cup E_{11} = \{x_9, x_{14}, x_{15}, x_{17}, x_{18}\}$	(V, E)	$[\{x_1, x_4, x_7\}, \{x_{16}\}]$
5	$\{x_1, x_4, x_7\}$	$\{x_2, x_3, x_6\}$	$\{x_1, x_4, x_7\}$	$\{E_1, E_3\}$	B_3	$E_1 = \{x_1, \ldots, x_4\}$	$(V, E \backslash \{E_3\})$	$[\{x_{16}\}]$
6	$\{x_{16}\}$	$\{x_{14}, x_{17}, x_{18}\}$	$\{x_{16}\}$	$\{E_{10}, E_{12}\}$	B_4	$E_{10} \cup E_{12} = \{x_{14}, x_{16}, x_{17}, x_{18}\}$	$(V, E \backslash \{E_3\})$	$\emptyset$
7	-	-	-	-	-	$B_7 = \{x_7\}$	$(V, E \backslash \{E_3\})$	-

Table 1: Execution of k-PH on the hypergraph of Figure 3(b) with $k = 5$.

As k-PH exploits a heuristic $\mathcal{H}$, its correctness also depends on that of $\mathcal{H}$. So, before proving the correctness of k-PH, we specify the necessary conditions that $\mathcal{H}$ must verify. To be correct, the heuristic $\mathcal{H}$ must therefore :

- calculate a new bag B_i of size at most k,
- select at least one hyperedge of S_i whose class is 2,
- make sure that the new bag B_i is such that there exists one already computed bag B_ℓ such that $(\cup_{1 \leq j \leq i-1} B_j) \cap B_i \subseteq (B_\ell \cap B_i)$.

For example, it can easily be shown that the heuristic $\mathcal{H}_\alpha$ satisfies these conditions.

Theorem 1 *k-PH is correct under the assumption that the heuristic $\mathcal{H}$ is correct.*

Proof: To prove the correction of k-PH, we must show that:

(1) k-PH ends;

(2) $H' = (V, E')$ is a partial hypergraph of H;

(3) B is a the set of bags of a tree-decomposition of $2_{SEC}(H')$ whose width is at most $k - 1$.

We prove each of these assertions.

(1) At each step of the loop, we add to B a new bag that contains at least one vertex that does not appear in any previous bag. Also, after a finite number of iterations through the loop, no new connected components will be inserted into the queue, and this queue will become empty and thus the loop will stop.

(2) By construction, $H' = (V, E')$ is a partial hypergraph of H.

(3) We must show that the 3 conditions required for a tree-decomposition are verified:

(i) ($\cup_{i \in I} B_i = V$) By construction, we know that each vertex of H' belongs to at least one bag. It should be remembered here that for each one of the vertices that do not appear in any hyperedge selected in H', the last step of k-PH builds a specific bag. So, this property is satisfied.

(ii) (For every edge $\{x, y\}$ of the graph, $\exists i \in I$ such that $\{x, y\} \subseteq B_i$) This property is verified since the bags in B are defined by the hyperedges of H' and thus, every hyperedge of H' is included in at least one bag. Since we consider here $2_{SEC}(H')$, that is the 2-section of H', all the edges of this graph are included in at least one bag.

(iii) ($\forall i, j, k \in I$, if k is on a path between i and j in T, then $B_i \cap B_j \subseteq B_k$) We know that by construction, for any new bag B_i, there is at least one previous bag B_ℓ such that $(\cup_{1 \leq j \leq i-1} B_j) \cap B_i \subseteq (B_\ell \cap B_i)$. So, if we assume that in the tree-decomposition, there is an edge that connects B_ℓ to B_i, it is easy to see that for any previous bag B_j (that is $1 \leq j \leq l$), necessarily we have $B_i \cap B_j \subseteq B_\ell$. Then by induction, it is easy to see that this property holds for every $B_{\ell'}$ which is on a path between B_i and B_j in the tree-decomposition, that is $B_i \cap B_j \subseteq B_{\ell'}$. So, this third property of tree-decompositions is satisfied.

Finally, by construction, we know that the size of bags belonging to B is at most k. So, the width of the tree-decomposition of $2_{SEC}(H')$ induced by B is at most $k-1$.

To conclude, it should be remembered that the validity of the algorithm is of course conditioned by the validity of the used heuristics. In fact, it is enough to consider heuristics that will be compatible with the conditions imposed. Indeed, trivially, a heuristic allowing to build bags whose size is strictly greater than k will not allow ensuring the correctness of k-PH algorithm if it uses it. □

The time complexity of k-PH cannot be evaluated without knowing in detail the cost of the different internal treatments such as *Comp-CC*, the computing of a new bag B_i, the cost of the heuristic $\mathcal{H}$, and also, the cost of updating H'. In order to make these different treatments efficient, several data structures must be implemented and these must be detailed. These descriptions and a detailed version of the different algorithms and their time complexity analysis are given in Appendix A. We present here the results of this evaluation. So, before that, we recall some notations:

- n is the number of vertices in V;
- e is the number of hyperedges in E;
- d is the maximum degree among all the vertices in H (the degree $d(v)$ of a vertex v is the number of hyperedges in E containing v);
- r is the rank of the hypergraph, that is the maximum size of the hyperedges in E;
- N is the number of bags in the computed tree-decomposition.

Moreover, to assess the complexity of k-PH, we assume that the cost of the considered heuristic $\mathcal{H}$ is h.

Theorem 2 *The time complexity of k-PH is in $O(N(h + e \cdot r + n \cdot d(N + r)))$.*

To give an illustration of this complexity, we consider the complexity of the heuristic $\mathcal{H}_\alpha$ which is $O(n \cdot k + e \cdot r)$. So, using this heuristic, the time complexity of k-PH is $O(N(h + e \cdot r + n \cdot d(N + r))) = O(N(n \cdot k + e \cdot r + n \cdot d(N + r)))$. This expression can be rephrased if we consider certain upper bounds. Indeed, we know that $N \leq n$, we thus obtain the complexity $O(n(n \cdot k + e \cdot r + n \cdot d(n + r))) = O(n \cdot n \cdot k + n \cdot e \cdot r + n \cdot n \cdot d \cdot n + n \cdot n \cdot d \cdot r) = O(n^2 \cdot k + n \cdot e \cdot r + n^3 \cdot d + n^2 \cdot d \cdot r) = O(n^2 \cdot (k + n \cdot d + r \cdot d) + n \cdot e \cdot r)$. On the other hand, if we also consider that the width $k-1$ is a constant, and therefore that the treatments will only consider hyperedges whose size is less than or equal to k, this complexity can be reduced to $O(n^3 \cdot d + n \cdot e)$.

As mentioned previously, this algorithm does not guarantee that the obtained hyperedge set is maximal for inclusion. However, it has certain properties related to optimality, as soon as hypergraphs belonging to particular classes are considered as input. Among these classes, one can find α-acyclic hypergraphs [6]. In this case, it is sufficient to adapt the parameter k to show that their processing is optimal using k-PH. It should be noted that the relationship between hypergraph α-acyclicity and tree-decomposition are very close. Indeed, for any graph $G = (V, E)$, for any tree-decomposition of G, it is well known that the set B of bags allows defining a hypergraph $H = (V, B)$ which is α-acyclic. Moreover, the associated tree-decomposition is then a join tree, and in this case, assuming that k is the rank of the hypergraph H, the maximal partial hypergraph H' that must be computed is exactly H. So, in this case, the Maximal Partial Hypergraph of Bounded Width problem can be solved efficiently. Finally, given an α-acyclic hypergraph, it is easy (possible in linear time [19]) to recognize it and then compute an optimal tree decomposition of its 2-section. Nevertheless, we show here that k-PH is optimal in terms of result without knowing beforehand that the hypergraph is α-acyclic.

To show the optimality of the result of k-PH for α-acyclic hypergraphs, we first need to give a particular value for the parameter k which is related to the size of the largest hyperedge in H. Then we have to consider the heuristic $\mathcal{H}_\alpha$ recalling that $\mathcal{H}_\alpha$ constructs the first bag B_1 by choosing as many maximum size hyperedges as possible while making sure that the condition $|B_1| \leq k$ holds. By so doing, when k is equal to m, B_1 will contain a hyperedge of size k, and possibly smaller included hyperedges. Then, for building a bag B_i, it first chooses as bag B_ℓ the existing bag having the largest intersection with N_i. Once B_ℓ selected, it selects as the first hyperedge to be in B_i the one that shares the maximum

number of vertices with N_i. Finally, this latter step is repeated until no hyperedge can be added without violating the condition $|B_i| \leq k$.

Theorem 3 *If a connected hypergraph $H = (V, E)$ is α-acyclic and if k is equal to the maximum size of hyperedges in H, then the result of k-PH with $\mathcal{H}_\alpha$ is optimal, that is k-PH finds a partial hypergraph $H' = (V, E')$ such that $E' = E$.*

Proof: We have to show that during the calculation of H', no hyperedge is deleted.

In k-PH, the removed hyperedges belong to class (2). These are the hyperedges that overlap N_i and $N_{H'}(N_i, X_i)$, namely, hyperedges $E_j \in E'$ such that $E_j \subseteq N_i \cup N_{H'}(N_i, X_i)$ and $N_{H'}(N_i, X_i) \cap E_j \neq \emptyset$ and $N_i \cap E_j \neq \emptyset$. The hyperedges $E_j \in S_i$ that are deleted are exactly such that $E_j \cap N_i \not\subset B_i$ and $E_j \cap B_i \not\subset N_i$.

The hyperedges E_j that are deleted are such that $E_j \cap B_i \neq \emptyset$ and of course also $E_j \not\subset B_i$, but among these, the ones that satisfy $E_j \cap N_i \subseteq B_\ell \cap B_i$ must not be deleted. We show that if H is α-acyclic, none of these hyperedges can appear during the calculation of H', and thus no hyperedge is deleted. To do this, we show that at each step of the algorithm, the partial sub-hypergraph taken into account for the calculation of the bags is acyclic and that this ensures that there is at least one hyperedge that avoids any deletion of hyperedge. The proof is organized first by treating the basic case for the calculation of the first bag, then for each of the subsequent steps.

First, we take for B_1 a hyperedge of maximum size, that is k. It is the only particular adaptation of the algorithm to find the optimal result. B_1 contains a hyperedge of size k, and possibly smaller included hyperedges. Then, connected components induced by the deletion of B_1 are calculated and inserted into Q. Of course, no hyperedge is removed for the calculation of H' since it is the particular case of the first step. Moreover, once B_1 has been calculated, if $X_1, X_2, \ldots X_{n_1}$ are the n_1 sets stored in Q, one can observe that for each X_i $(1 \leq i \leq n_1)$, the hypergraph $H'[B_1 \cup X_i]$ is α-acyclic. We prove this below.

We know that a hypergraph is α-acyclic if and only if it is conformal and chordal. As any subgraph of a chordal graph is chordal, then the graph $2_{SEC}(H'[B_1 \cup X_i])$ is chordal because it is a subgraph of $2_{SEC}(H)$. We now show that $H'[B_1 \cup X_i]$ is conformal. No hyperedge included in $B_1 \cup X_i$ is deleted, and no new edge has appeared in $2_{SEC}(H'[B_1 \cup X_i])$. Indeed, all edges of $2_{SEC}(H'[B_1 \cup X_i])$ induced by hyperedges of H intersecting B_1 and which do not appear in $H'[B_1 \cup X_i]$ are already in $2_{SEC}(H'[B_1 \cup X_i])$ due to the existence of the hyperedge B_1 in $H'[B_1 \cup X_i]$. It follows that no new clique which would not be covered by a hyperedge of $H'[B_1 \cup X_i]$ does not appear in $2_{SEC}(H'[B_1 \cup X_i])$ and thus that the hypergraph $H'[B_1 \cup X_i]$ is conformal.

We need another property to show the maximality of the calculation performed by k-PH, i.e. no hyperedge is removed when calculating H'. It deals with the interaction between the already constructed bags and the connected components stored in Q. We first study this property concerning the computation of B_1. We know that B_1 is a hyperedge of $H'[B_1 \cup X_i]$ and thus, there is at least one hyperedge E_i of $H'[B_1 \cup X_i]$ such that $E_i \cap X_i \neq \emptyset$ and $E_i \cap B_1 = N_i$. We show the existence of such a hyperedge. Since the graph $2_{SEC}(H'[B_1 \cup X_i])$ is chordal, it admits a perfect elimination ordering. In this perfect elimination ordering, we consider the numbering of the vertices among those of $N_i \cup N_{H'}(N_i, X_i)$. More precisely, rather than considering $N_{H'}(N_i, X_i)$ we consider $Sep \subseteq N_{H'}(N_i, X_i)$ which is a minimal separator in $2_{SEC}(H'[B_1 \cup X_i])$ between N_i and the vertices of X_i that do not belong to $N_{H'}(N_i, X_i)$. Since $2_{SEC}(H'[B_1 \cup X_i])$ is a chordal graph and any minimal separator in such a graph is a clique, we know that Sep is a clique of $2_{SEC}(H'[B_1 \cup X_i])$. Since Sep is a separator in $2_{SEC}(H'[B_1 \cup X_i])$, we know that all the edges that connect N_i to X_i have a vertex in Sep. We will consider two cases. Either (case 1) a vertex of N_i is numbered before the first vertex of Sep is numbered, or (case 2) some vertices of Sep are numbered before vertices of N_i are numbered:

(1) Let x be the first vertex of N_i which is numbered before the vertex of Sep that is numbered first. As we consider a perfect elimination ordering, necessarily, all neighbors of x in $N_i \cup N_{H'}(N_i, X_i)$ constitute a clique. The neighbors of x in $N_i \cup N_{H'}(N_i, X_i)$ are on the one hand all the other vertices of N_i and on the other hand at least one vertex of Sep because there is at least one edge from x linking an unnumbered vertex of Sep in $2_{SEC}(H'[B_1 \cup X_i])$. On the other hand, as $H'[B_1 \cup X_i]$ is α-acyclic, this hypergraph is thus conformal, and it has at least one hyperedge E_i which contains all the vertices of this clique formed of N_i and at least the neighboring vertex of x in $N_{H'}(N_i, X_i)$.

(2) Consider now that the first numbered vertices appear in Sep. We distinguish two cases:

   (i) All the vertices of Sep are numbered before the ones of N_i. In this case, each time a vertex of Sep is numbered, its neighbors in N_i must be linked to all vertices of Sep not yet numbered because Sep is a clique. And so, the last vertex of Sep which is numbered has for neighbors all the vertices of N_i because they all have at least one neighbor vertex in Sep. Thus, the last vertex of Sep which is numbered is a neighbor of all the vertices of N_i and thus all these vertices constitute a clique.

   (ii) A vertex x of N_i is numbered before that all the vertices of Sep are numbered. We then consider the first vertex of N_i which is numbered. If this vertex x has a neighbor y in Sep not yet numbered, then this vertex of Sep is linked to all vertices of N_i because x is the first vertex of N_i numbered, and the vertices of N_i with y thus constitute a clique. Otherwise, suppose all the neighbors of x in Sep are already numbered. Consider $y \in Sep$,

a neighbor of x that has already been numbered. When y has been numbered, necessarily, it had to form a clique containing both x, but also, all the vertices of *Sep* not yet numbered, including the last vertex of *Sep* that has been numbered. This leads to a contradiction because x should then have at least one neighbor in *Sep* not numbered, or x was not numbered before all the vertices of *Sep* were numbered.

In cases (i) and (ii), we have thus shown that there is a clique consisting of all the vertices of N_i and at least one vertex of *Sep* in $2_{SEC}(H'[B_1 \cup X_i])$. Thus, because of the conformity of $H'[B_1 \cup X_i]$, there is at least one of its hyperedges E_i which contains all these vertices.

Assume now that what is valid for this first stage holds for any stage, namely that for any X_i memorized in Q, X_i was built then inserted in Q starting from a bag B_ℓ such that the hypergraph $H'[B_\ell \cup X_i]$ to which one adds a hyperedge formed by the vertices of B_ℓ is α-acyclic (this hypergraph is denoted $H'[B_\ell \cup X_i] + B_\ell$). Moreover, there is at least one hyperedge E_i of $H'[B_\ell \cup X_i] + B_\ell$ such that $E_i \cap X_i \neq \emptyset$ and $B_\ell \cap E_i = N_i$.

Now, for the i^{th} step $(1 < i)$, B contains the bags $B_1, \dots B_{i-1}$. Let X_i be the connected component removed from Q, and let B_ℓ be the existing bag used to build X_i. We recall that the heuristic $\mathcal{H}_\alpha$ first chooses as bag B_ℓ the existing bag having the largest intersection with N_i and then, chooses as the first new hyperedge to be in B_i the one that shares the maximum number of vertices with N_i. First, the previous bag selected by $\mathcal{H}_\alpha$ is necessarily the same bag B_ℓ that was used to construct X_i and insert it into Q (or a bag with the same intersection with N_i). By induction hypothesis, we know that the hypergraph $H'[B_\ell \cup X_i] + B_\ell$ is α-acyclic and that there is at least one hyperedge E_i of $H'[B_\ell \cup X_i] + B_\ell$ such that $E_i \cap X_i \neq \emptyset$ and $B_\ell \cap E_i = N_i$.

So, for the calculation of B_i, necessarily such a hyperedge E_i is chosen by the heuristic $\mathcal{H}_\alpha$ and belongs to the new bag B_i ($E_i \subseteq B_i$). Other hyperedges from S_i can be added to B_i. By induction hypothesis, since $B_\ell \cap E_i = N_i$, we have $N_i \subset E_i$. Thus, all the hyperedges E_j of class 2 verify $E_j \cap N_i \subseteq E_i$ and as by construction $E_i \subseteq B_i$, necessarily $E_j \cap N_i \subseteq B_i$. As the hyperedges $E_j \in S_i$ that are deleted are such that $E_j \cap N_i \not\subset B_i$ (and $E_j \cap B_i \not\subset N_i$), no hyperedge of H' is deleted.

So, after the calculation of the new connected components induced by B_i, we must ensure that the induction property holds. First, for the connected components of $H'[[X_i \backslash B_i]]$, that is $X_{i_1}, X_{i_2}, \dots X_{i_{n_i}}$ which will then be inserted into Q, we must verify that each hypergraph $H'[B_i \cup X_{i_j}] + B_i$ (with $1 \leq j \leq n_i$) is α-acyclic and that there exists at least one hyperedge E_{i_j} belonging to $H'[B_i \cup X_{i_j}] + B_i$ such that $E_{i_j} \cap X_{i_j} \neq \emptyset$ and $B_i \cap E_{i_j} = N_{i_j}$.

We prove first that $\forall j, 1 \leq j \leq n_i$, the hypergraph $H'[B_i \cup X_{i_j}] + B_i$ is α-acyclic. The proof is close to the one given above for B_1 but slightly more complicated because of the existence in $H'[B_i \cup X_{i_j}] + B_i$ of the hyperedge B_i. We know that a hypergraph is α-cyclic if and only if it is conformal and chordal.

- (Chordal). As $H'[B_\ell \cup X_i] + B_\ell$ is α-acyclic, so the graph $2_{SEC}(H'[B_\ell \cup X_i] + B_\ell)$ is chordal. Contrary to the basic case where B_1 is both a bag and a hyperedge of H, here B_i is not necessarily a hyperedge of H and therefore $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$ is not necessarily a subgraph of $2_{SEC}(H'[B_\ell \cup X_i] + B_\ell)$ because edges that are not in $2_{SEC}(H'[B_\ell \cup X_i] + B_\ell)$ may belong to $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$ due to the existence of the hyperedge B_i and its completion in the 2-section. We show that no chordless cycle can have been created by the addition of such edges. For such a cycle to be created, it must contain at least 4 vertices of which two vertices x and y belong to B_i, and two other vertices u and v belong to X_{i_j}, with $\{x, u\}$ et $\{y, v\}$, edges of $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$ considering that x and y are not neighbors in $2_{SEC}(H'[B_\ell \cup X_i] + B_\ell)$ but are neighbors in $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$ (by completion of the hyperedge B_i). We know that there is necessarily a path from u to v internal to X_{i_j} in $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$ (and thus already in $2_{SEC}(H'[B_\ell \cup X_i] + B_\ell)$) because X_{i_j} is a connected component.

  Adding the edge $\{x, y\}$ would then create a chordless cycle in $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$. By construction of X_{i_j}, x and y necessarily appear in a separator of X_{i_j}, and more precisely, in a minimal separator of X_{i_j} because of the existence of the edges $\{x, u\}$ and $\{y, v\}$. But since $2_{SEC}(H'[B_\ell \cup X_i] + B_\ell)$ is chordal, and any minimal separator in a chordal graph is a clique, necessarily, x and y are neighbors in that graph, and the edge $\{x, y\}$ has not been added by completion of B_i, and therefore cannot have created a new cycle without a chord in $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$. Therefore, this graph is chordal.

- (Conformal). It is necessary to show that $H'[B_i \cup X_{i_j}] + B_i$ is conformal, and thus that any clique of $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$ is included in a hyperedge of $H'[B_i \cup X_{i_j}] + B_i$. We know that all the cliques included in B_i are covered by the hyperedge B_i. In the same way, all the cliques included in X_{i_j} are covered by a hyperedge appearing in this part of $H'[B_i \cup X_{i_j}] + B_i$ since it was the case before the computation of B_i as $H'[B_\ell \cup X_i] + B_\ell$ is by hypothesis α-acyclic and thus conformal, and that no edge has been added in $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$ between vertices of X_{i_j}. The only cliques that may appear when creating the B_i hyperedge are therefore overlapping B_i and X_{i_j}, and must therefore concern at least 3 vertices, two vertices x and y of B_i initially non-neighboring, and a vertex z of X_{i_j} such that $\{x, z\}$ and $\{y, z\}$ appear in $2_{SEC}(H'[B_\ell \cup X_i] + B_\ell)$. It is thus necessary to prove that x, y and z are covered by

a hyperedge of $H'[B_i \cup X_{i_j}] + B_i$. By reasoning like for the proof of the chordality of $2_{SEC}(H'[B_i \cup X_{i_j}] + B_i)$, we show that the edge $\{x, y\}$ was not added by completion of B_i (addition of all possible edges), because here again, due to the presence of the edges $\{x, z\}$ and $\{y, z\}$, x and y are necessarily in a minimal separator of X_{i_j} and thus are already connected. Thus, as x, y and z already constitute a clique appearing in $2_{SEC}(H'[B_\ell \cup X_i] + B_\ell)$ and that the hypergraph $H'[B_\ell \cup X_i] + B_\ell$ was conformal before completion of B_i, there is a hyperedge of $H'[B_i \cup X_{i_j}] + B_i$ which covers this clique.

It remains now to be proved that there exists at least one hyperedge E_{i_j} belonging to $H'[B_i \cup X_{i_j}] + B_i$ such that $E_{i_j} \cap X_{i_j} \neq \emptyset$ and $B_i \cap E_{i_j} = N_{i_j}$. In fact, it is sufficient to use the same scheme of proof as in the basic case. Indeed, and unlike the case of the α-acyclicity of the hypergraph $H'[B_i \cup X_{i_j}] + B_i$, here the conditions of the basic case are preserved and the proof is identical. □

Surprisingly, unless the basic heuristic proposed here is modified, the class of chordal graphs is unfortunately not processed optimally by this algorithm. One can very easily find counter-examples based on a few vertices (e.g. a k-tree with $k = 2$).

The following section presents an evaluation of the performance of this algorithm both in terms of computation times and the quality of the computation results, and thus the proportion of hyperedges belonging to the obtained partial hypergraph.

4 Experimental Evaluation

In this section, we study the behavior of k-PH on a large benchmark of hypergraphs from various communities. We first describe our experimental protocol. Then, we assess its efficiency by considering its capacity to get closer to the optimum and its runtime. Finally, we consider the behavior of k-PH depending on the value of k.

4.1 Experimental Protocol

We implement k-PH in our own C++ hypergraph library. The experiments are performed on Dell PowerEdge R440 servers with Intel Xeon Silver 4112 processors (clocked at 2.6 GHz) under Ubuntu 18.04. For each instance and a given integer k, k-PH with a given heuristic $\mathcal{H}$ is allocated a slot of 30 minutes and at most 16 GB of memory per instance.

We describe below the heuristics $\mathcal{H}$ we consider in k-PH and the benchmark exploited in our experimentations.

4.1.1 The Considered Heuristics

We considered and implemented several heuristics. In this section, we only report the results obtained by the two more interesting ones. Moreover, in the following definitions, we assume that the hyperedges whose size exceeds the parameter k of k-PH have been deleted first.

The first heuristic we consider is the heuristic $\mathcal{H}_\alpha$ defined for the need of Theorem 3 and which is used in the description of the k-PH algorithm. We give some more details about this heuristic. $\mathcal{H}_\alpha$ constructs the first bag B_1 by choosing as many maximum size hyperedges as possible while making sure that the condition $|B_1| \leq k$ holds. Then, for building a bag B_i, it first chooses as bag B_ℓ the existing bag having the largest intersection with N_i. Once B_ℓ is selected, it selects as the first hyperedge to be in B_i the one that shares the maximum number of vertices with N_i. Finally, this latter step is repeated until no hyperedge can be added without violating the condition $|B_i| \leq k$. As mentioned above, $\mathcal{H}_\alpha$ is designed in order to ensure the optimality of k-PH on α-acyclic hypergraphs when $k = m$.

The second one, denoted $\mathcal{H}_\cap$, starts the construction of B_1 like $\mathcal{H}_\alpha$ by choosing a hyperedge of maximum size. Then it adds as many hyperedges as possible in B_1 (that is under the condition $|B_1| \leq k$) by selecting first the hyperedges having the largest intersection with B_1. Regarding the build of B_i, it first chooses as bag B_ℓ the existing bag which intersects the largest number of candidate hyperedges from S_i. By so doing, we want to preserve as many hyperedges as possible among the hyperedges of S_i. Finally, $\mathcal{H}_\cap$ builds B_i like B_1 but by only considering the hyperedges of S_i which intersect B_ℓ. In some manner, $\mathcal{H}_\cap$ aims to built bags whose hyperedges overlap each others. Such a property may be notably of interest when we consider the problems related to these hypergraphs (e.g. solving graphical models). Whereas solving the related problems is out of the scope of this paper, studying such a heuristic has sense here.

Regarding the time complexity, $\mathcal{H}_\alpha$ and $\mathcal{H}_\cap$ run respectively in $O(n \cdot k + e \cdot r)$ and $O(n^2 \cdot e + k^2 \cdot e^2)$. For $\mathcal{H}_\cap$, using the *Bag_list* data structure, the selection of B_ℓ can be achieved in $O(N \cdot |S_i|)$. Indeed we have to count the number of hyperedges of S_i which intersect each existing bag and the number of existing bags is bounded by N. Building a bag B_i can be achieved in $O(k^2 \cdot |S_i|^2)$. Indeed, computing the intersection of each candidate hyperedge with B_i is feasible in $O(k^2)$ because the hyperedges and the bag have a size bounded by k and this is done at most $|S_i|$ times. So it results that $\mathcal{H}_\cap$ has a time complexity in $O(N \cdot |S_i| + k^2 \cdot |S_i|^2) = O(n \cdot e + k^2 \cdot e^2)$ knowing that $N \leq n$ and $|S_i| \leq e$.

4.2 Benchmarks

In order to make our experimentations as representative as possible, we consider hypergraphs from various communities. The considered hypergraphs are produced from instances that are usually exploited for benchmarking by these communities. Notably, in each community, these instances make it possible to compare solving methods for different problems which are generally at least NP-complete. They model both academic and real-world problems. We divide the benchmark into five categories depending on their origin. In each category, we only consider hypergraphs whose treewidth can be computed by the method of Tamaki[3] [33]. We now describe each category:

- $\mathcal{B}_{CSP}$: We consider 5,996 instances from the CSP3 repository[4]. From each CSP instance I, we build a hypergraph (called the *constraint hypergraph*) whose vertices correspond to the variables of I and the hyperedges correspond to the scopes of its constraints.
- $\mathcal{B}_{WCSP}$: We consider 1,535 WCSP instances from the EvalGM repository[5] and the ERGO repository[6]. The Weighted Constraint Satisfaction Problem (WCSP) allows to express optimization problems [34]. Like for CSP instances, from each WCSP instance I, we build a hypergraph whose vertices correspond to the variables of I and the hyperedges correspond to the scopes of its weighted constraints (or cost functions). Note that these hypergraphs are computed after applying a preprocessing step on the original WCSP instances. This preprocessing step is performed thanks to Toulbar2[7]. It consists in enforcing VAC (for Virtual Arc-Consistency [35]) and applying the MSD (for Min Sum Diffusion) algorithm with 1,000 iterations. It is usually exploited in this community and may lead to reduce the number of variables and cost functions before solving the instances.
- $\mathcal{B}_{H}$: We consider 2,871 hypergraphs from the HyperBench repository[8] [36, 37, 38, 39, 40]. These hypergraphs have been produced from Conjunctive Queries and CSP instances with the aim in view to study their hypertree width, generalized hypertree width and fractional hypertree width [41].
- $\mathcal{B}_{MC}$: We consider 511 instances of model counting from the Cachet repository[9] [42]. Each instance models a Bayesian Inference problem as a weighted model counting instance. From each instance I, we build a hypergraph whose vertices correspond to the Boolean variables of I and the hyperedges correspond to its clauses.
- $\mathcal{B}_{MN}$: We consider 389 Markov networks from the Probabilistic Inference Challenge 2011[10]. From each instance I, we build a hypergraph whose vertices correspond to the variables of I and the hyperedges correspond to the scopes of its functions.

Table 2 provides some information about the hypergraphs of each category, namely the minimum, maximum, average and standard deviation for the number of vertices, the number of hyperedges, the maximum size of hyperedges and the treewidth for each category. Finally, we denote $\mathcal{B}_{all}$ the union of the five categories. So $\mathcal{B}_{all}$ involves 11,302 hypergraphs.

4.3 Optimality and Runtime of k-PH

In this part, we aim to assess how close k-PH can be to optimality. For this purpose, we consider the percentage of hyperedges that are retained by k-PH when the parameter k is set to the treewidth of the considered hypergraph plus one. The higher the percentage is, the closer k-PH will be to optimality.

Table 3 presents the number of hypergraphs depending on the percentage of hyperedges retained by k-PH for $\mathcal{H}_{\alpha}$ and $\mathcal{H}_{\cap}$. First, we can observe that about 12.5% of hypergraphs (respectively 13%) in $\mathcal{B}_{all}$ are optimally processed by k-PH with $\mathcal{H}_{\alpha}$ (resp. $\mathcal{H}_{\cap}$). These hypergraphs are not necessarily α-acyclic. Indeed, there exist 20 α-acyclic hypergraphs in $\mathcal{B}_{CSP}$ (resp. 484 in $\mathcal{B}_{H}$ and 5 in $\mathcal{B}_{MN}$) while there is none in $\mathcal{B}_{WCSP}$ and $\mathcal{B}_{MC}$. Note that all the α-acyclic hypergraphs except 20 are also β-acyclic. Now, we consider the algorithm MCS [19] in order to determine whether the 2-section of a hypergraph is chordal. If not, we exploit the number of edges that will be added if we triangulate the 2-section according to the elimination order produced by MCS. This number allows us to estimate how the 2-section is close to being chordal. We then observe that 1,117 hypergraphs have a chordal 2-section among the hypergraphs which are processed optimally. This number includes of course the 509 α-acyclic hypergraphs. Moreover, 131 hypergraphs are close to being chordal with at most 10 edges added by the triangulation based on MCS. In the same spirit, we remark that k-PH with $\mathcal{H}_{\alpha}$ and $\mathcal{H}_{\cap}$ respectively retains more than 75% of hyperedges for about 93% and 96% of hypergraphs having a chordal 2-section (1,632

[3] github.com/TCS-Meiji/PACE2017-TrackA
[4] http://www.CSP.org/series
[5] http://genoweb.toulouse.inra.fr/~degivry/evalgm
[6] http://carlit.toulouse.inra.fr/SoftCSP/Files/cflibtars/ergo.tgz
[7] https://miat.inrae.fr/toulbar2/
[8] hyperbench.dbai.tuwien.ac.at
[9] https://www.cs.rochester.edu/u/kautz/Cachet/Model_Counting_Benchmarks/index.htm
[10] https://www.cs.huji.ac.il/project/PASCAL/

		n	e	m	w
$\mathcal{B}_{CSP}$	min	6	5	2	1
	max	9,226	77,327	181	200
	avg.	97.47	506.25	4.48	20.71
	std dev.	192.45	3,101.11	7.05	17.12
$\mathcal{B}_{WCSP}$	min	4	11	2	3
	max	1,689	120,792	68	485
	avg.	95.48	1,470.71	2.36	19.77
	std dev.	108.39	6,088.28	2.45	33.11
$\mathcal{B}_{H}$	min	2	2	2	1
	max	1,454	893	145	144
	avg.	55.50	53.80	6.44	14.48
	std dev.	72.50	47.08	6.43	12.99
$\mathcal{B}_{MC}$	min	100	250	4	10
	max	672	828	5	30
	avg.	172.48	332.53	4.88	22.25
	std dev.	149.02	106.68	0.32	4.96
$\mathcal{B}_{MN}$	min	14	13	2	2
	max	1,094	100,400	3	200
	avg.	371.92	12,078.29	2.13	31.70
	std dev.	367.37	23,445.22	0.33	33.59

Table 2: Minimum, maximum, average and standard deviation for the number of vertices, the number of hyperedges, the maximum size of hyperedges and the treewidth for each category.

hypergraphs). At the same time, the corresponding percentage for non-chordal hypergraphs is about 82%. So, having a chordal 2-section turns to be an interesting property to guarantee a result close to the optimality.

Now, if we compare the results obtained by k-PH with $\mathcal{H}_\alpha$ and $\mathcal{H}_\cap$, we can note that no heuristic outperforms the other. $\mathcal{H}_\alpha$ seems to perform better on the category $\mathcal{B}_{WCSP}$ while $\mathcal{H}_\alpha$ obtains better results on $\mathcal{B}_H$ and $\mathcal{B}_{MN}$. The results for $\mathcal{B}_{CSP}$ depend on the desired percentage of retained hyperedges. Indeed, $\mathcal{H}_\cap$ turns to be more relevant when this percentage exceeds 90%.

Finally, we consider the runtime of k-PH. Table 4 provides the minimum and maximum runtime, the average runtime and the standard deviation for the runtime of k-PH for $\mathcal{H}_\alpha$ and $\mathcal{H}_\cap$. Clearly, we can observe that $\mathcal{H}_\alpha$ and $\mathcal{H}_\cap$ lead to the same behavior with respect the runtime. Indeed, on each category, the runtimes are very close. Moreover, whatever the heuristic, applying k-PH requires very little time. In our experiments, the runtime does not exceed 3 minutes. Note that this runtime is reached for a hypergraph from $\mathcal{B}_{WCSP}$ having 500 vertices and 120,792 hyperedges and explains why the standard deviation for $\mathcal{B}_{WCSP}$ is larger than that of the other categories. Finally, it can be noted that, for 99.6% of the considered hypergraphs, the runtime does not exceed 1 second. So our approach is very efficient.

Percentage of retained hyperedges	# instances											
	$\mathcal{B}_{CSP}$		$\mathcal{B}_{WCSP}$		$\mathcal{B}_{H}$		$\mathcal{B}_{MC}$		$\mathcal{B}_{MN}$		$\mathcal{B}_{all}$	
	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$
$= 100\%$	373	373	**125**	118	841	**902**	0	0	71	71	1,410	**1,464**
$\geq 95\%$	1,512	**1,524**	**329**	308	890	**975**	0	0	215	**225**	2,946	**3,032**
$\geq 90\%$	2,345	**2,440**	**861**	736	1,092	**1,215**	0	0	256	**280**	4,554	**4,671**
$\geq 85\%$	**3,513**	3,455	**1,269**	1,122	1,453	**1,534**	**2**	0	313	**342**	**6,550**	6,453
$\geq 80\%$	**4,715**	4,527	**1,382**	1,330	1,969	**2,009**	**23**	6	375	**381**	**8,464**	8,253
$\geq 75\%$	**5,249**	5,243	**1,412**	1,389	2,297	**2,331**	**175**	142	385	**388**	**9,518**	9,493
$\geq 70\%$	**5,450**	5,449	**1,454**	1,413	2,469	**2,471**	489	**499**	388	**389**	**10,250**	10,221
$\geq 65\%$	**5,606**	5,591	**1,484**	1,426	2,545	**2,558**	511	511	388	**389**	**10,534**	10,475
$\geq 60\%$	**5,820**	5,762	**1,506**	1,431	**2,685**	2,681	511	511	388	**389**	**10,910**	10,774
$\geq 55\%$	**5,889**	5,847	**1,526**	1,451	**2,762**	2,757	511	511	389	389	**11,077**	10,955
$\geq 50\%$	**5,962**	5,928	**1,535**	1,486	**2,819**	2,796	511	511	389	389	**11,216**	11,110

Table 3: Number of hypergraphs depending on the percentage of hyperedges retained by k-PH for $\mathcal{H}_\alpha$ and $\mathcal{H}_\cap$ when k is equal to the treewidth of the considered hypergraph plus one.

	$\mathcal{B}_{CSP}$		$\mathcal{B}_{WCSP}$		$\mathcal{B}_{H}$		$\mathcal{B}_{MC}$		$\mathcal{B}_{MN}$	
	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$	$\mathcal{H}_\alpha$	$\mathcal{H}_\cap$
min	2.3E-5	2.2E-5	3.7E-5	3.6E-5	1.0E-5	9.0E-6	4.8E-04	5.3E-04	1.3E-4	1.3E-4
max	40.110	40.721	177.307	179.737	0.009	0.008	0.007	0.007	2.449	2.394
avg.	0.017	0.016	0.361	0.363	4.5E-04	4.6E-04	0.002	0.002	0.138	0.137
std dev.	0.598	0.599	6.654	6.640	5.1E-04	4.8E-04	0.001	0.001	0.423	0.421

Table 4: Minimum and maximum runtime, average runtime and standard deviation for the runtime of k-PH for $\mathcal{H}_\alpha$ and $\mathcal{H}_\cap$ when k is equal to the treewidth of the considered hypergraph plus one.

4.4 Behavior of k-PH Depending on the Value of k

In this part, we evaluate the behavior of k-PH when we do not make an assumption on the value of the treewidth of the considered instance. This makes sense from a practical viewpoint because it is not always possible to compute it for a matter of time, depending on the size of the instances. At this aim, we assess the behavior of k-PH on the whole benchmark $\mathcal{B}_{all}$ by varying the value of k as a percentage of the number n of vertices. For each considered category, Figures 4-8 present, for each heuristic, the percentage of instances for which k-PH retains a given percentage of hyperedges when k varies from 5% to 100% of the number of vertices in steps of 5%. Figure 9 does the same for the whole benchmark.

Whatever the considered category, we can observe that k-PH behaves similarly with $\mathcal{H}_\alpha$ and $\mathcal{H}_\cap$. However, we can note that for a given k, k-PH with $\mathcal{H}_\cap$ often turns out to be able to retain a few more hyperedges than k-PH with $\mathcal{H}_\alpha$. This trend is clearly visible for the category $\mathcal{B}_{MC}$ and less marked for the other categories.

Regarding the runtime of k-PH, again the heuristics $\mathcal{H}_\alpha$ and $\mathcal{H}_\cap$ lead to similar results. Processing any hypergraph from $\mathcal{B}_H$ or $\mathcal{B}_{MC}$ requires less than one second whatever the heuristic we consider. Likewise, about 99% (resp. 97%) of hypergraphs of $\mathcal{B}_{CSP}$ (resp. $\mathcal{B}_{WCSP}$) are processed in less than one second. Processing a hypergraph among the remaining ones of $\mathcal{B}_{CSP}$ (resp. $\mathcal{B}_{WCSP}$) require at most 154 seconds (resp. 205 seconds). These runtimes are reached for the largest hypergraphs in terms of number of hyperedges. The trend is different for $\mathcal{B}_{MN}$. Indeed, applying k-PH requires less than one second for a percentage of hypergraphs varying between 68% and 95% depending on the value of k. For the remaining hypergraphs, the runtime is generally about 30 seconds and rarely exceeds 4 minutes. Again, this concerns the largest hypergraphs in terms of number of hyperedges.

5 Conclusion

In this paper, we have proposed an algorithm called k-PH which calculates for a given hypergraph and a given constant k, a partial hypergraph for which the width of its 2-section is at most $k-1$. We have also shown that k-PH allows constructing an optimal partial hypergraph in the sense that all the hyperedges are selected for the case of α-acyclic hypergraphs. To assess the efficiency of k-PH with respect to the optimality criterion related to the maximum number of selected hyperedges, we performed experiments on a large benchmark including 11,302 instances from several communities. These experiments show for example that for about 90% of the instances, more than 70% of the hyperedges are retained in the calculated partial hypergraph when the value of k is equal to the treewidth plus one of the instances while runtimes being limited in practice. Moreover, this algorithm can easily be adapted according to the considered objective of selection of hyperedges. Indeed, one can thus add to it different heuristics which make it possible, for example, to take into account weights for hyperedge to be selected in the partial hypergraph. This should make it easier to process different types of graphical models. To conclude, the interest of this algorithm must now be evaluated for the treatment of graphical models.

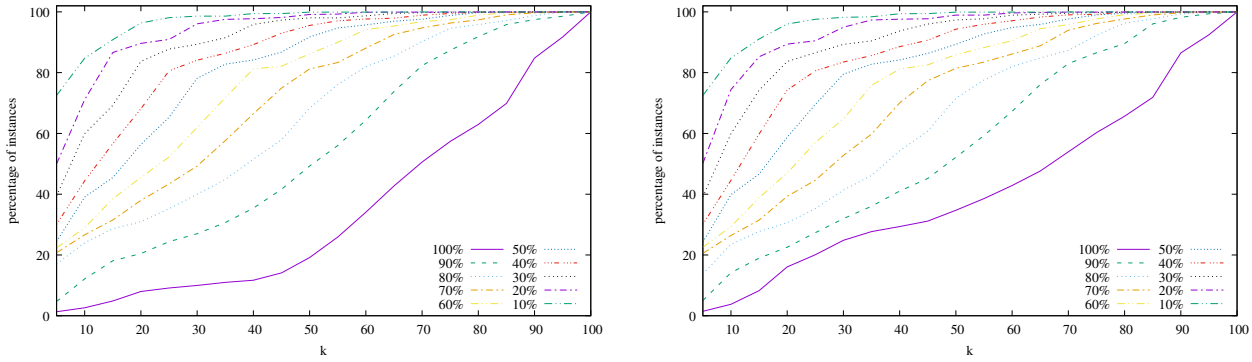
Acknowledgement

This work has been funded by the Agence Nationale de la Recherche project ANR-16-CE40-0028.

Appendix A - Complexity of k-PH

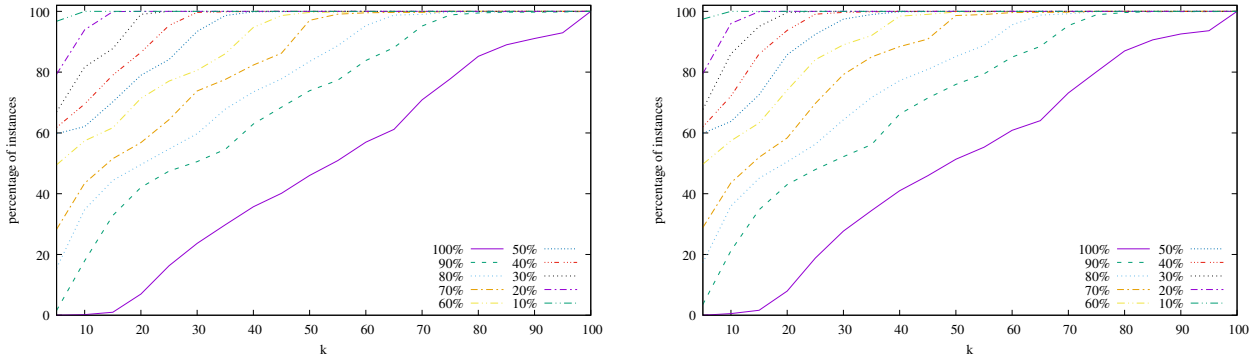
The time complexity of k-PH cannot be evaluated without knowing in details the cost of the different internal treatments such as *Comp-CC*, the computing of a new bag B_i, the computing of the sets N_i, $N_{H'}(N_i, X_i)$ and S_i, the cost of the heuristic $\mathcal{H}$, and also, the cost of updating H'. In order to make these different treatments efficient, several data structures must be implemented and these must be detailed. Thus, after their description and the time evaluation of these treatments, we evaluate the time complexity of k-PH. Before that, we recall some notations:

- n is the number of vertices in V;



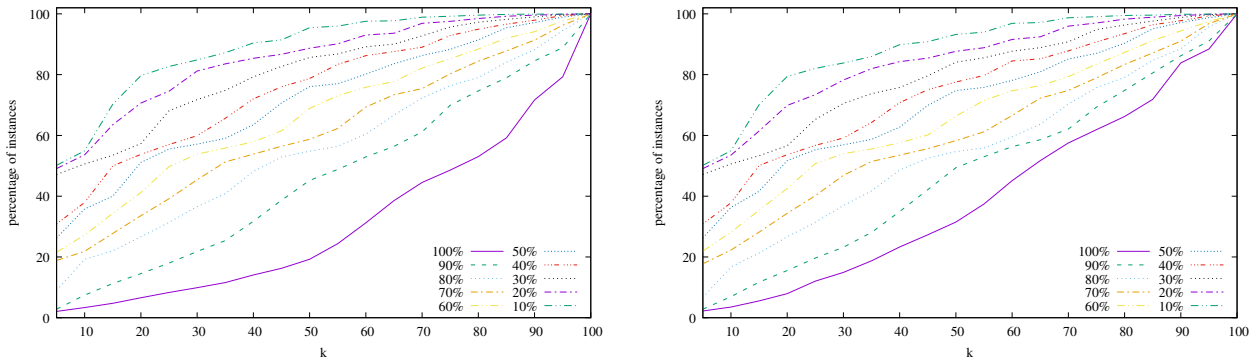
(a) (b)

Figure 4: Percentage of instances for a given percentage of retained hyperedges when k is defined as a percentage of n for $\mathcal{H}_\alpha$ (a) and $\mathcal{H}_\cap$ (b) (benchmark $\mathcal{B}_{CSP}$).



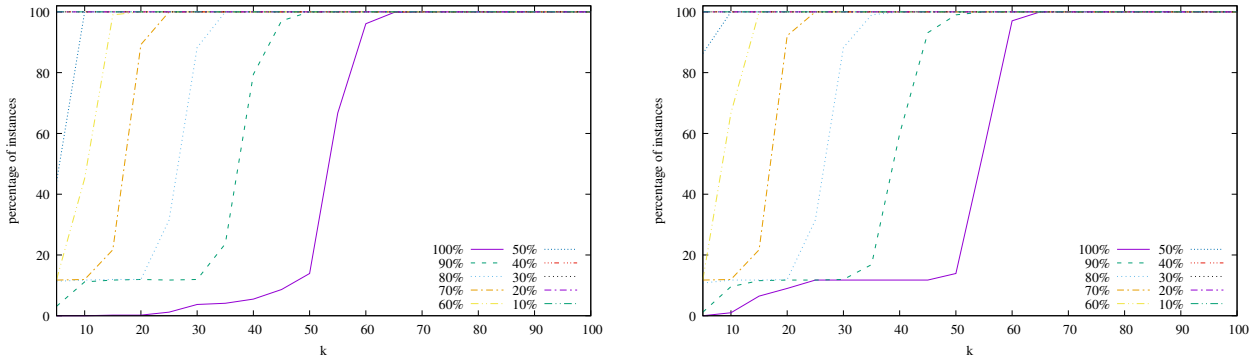
(a) (b)

Figure 5: Percentage of instances for a given percentage of retained hyperedges when k is defined as a percentage of n for $\mathcal{H}_\alpha$ (a) and $\mathcal{H}_\cap$ (b) (benchmark $\mathcal{B}_{WCSP}$).



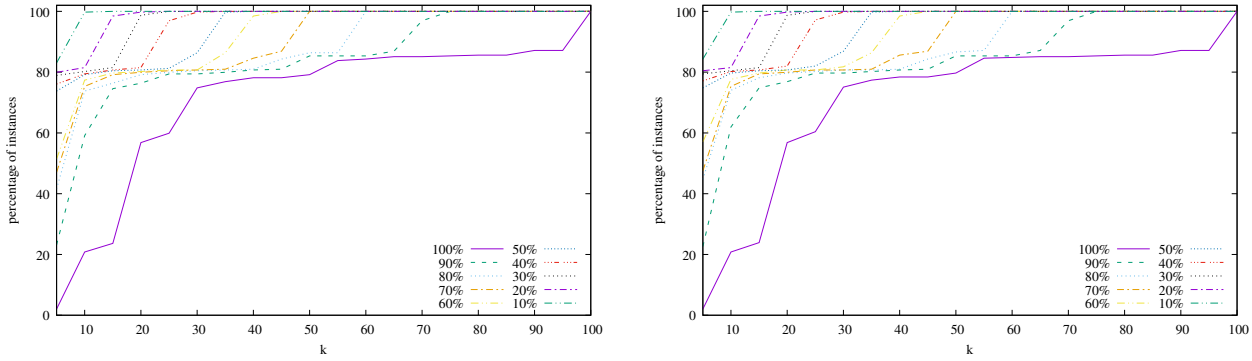
(a) (b)

Figure 6: Percentage of instances for a given percentage of retained hyperedges when k is defined as a percentage of n for $\mathcal{H}_\alpha$ (a) and $\mathcal{H}_\cap$ (b) (benchmark $\mathcal{B}_H$).



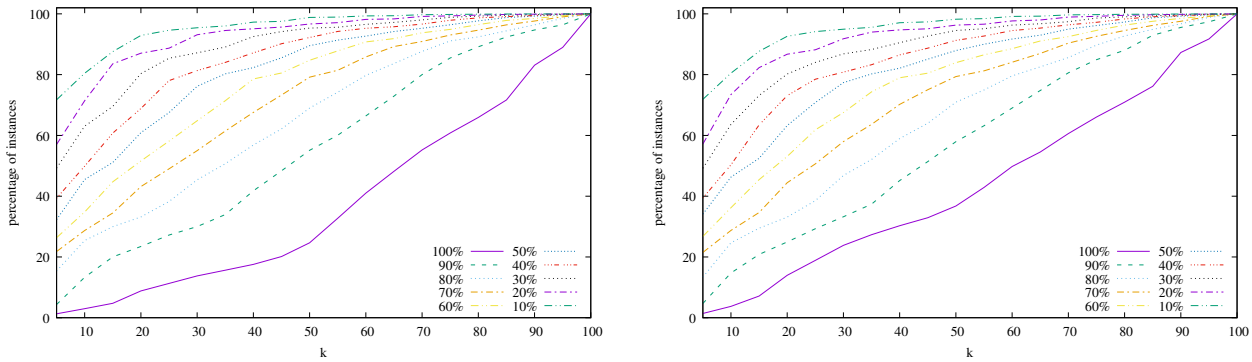
(a) (b)

Figure 7: Percentage of instances for a given percentage of retained hyperedges when k is defined as a percentage of n for $\mathcal{H}_\alpha$ (a) and $\mathcal{H}_\cap$ (b) (benchmark $\mathcal{B}_{MC}$).



(a) (b)

Figure 8: Percentage of instances for a given percentage of retained hyperedges when k is defined as a percentage of n for $\mathcal{H}_\alpha$ (a) and $\mathcal{H}_\cap$ (b) (benchmark $\mathcal{B}_{MN}$).



(a) (b)

Figure 9: Percentage of instances for a given percentage of retained hyperedges when k is defined as a percentage of n for $\mathcal{H}_\alpha$ (a) and $\mathcal{H}_\cap$ (b) (benchmark $\mathcal{B}_{all}$).

- e is the number of hyperedges in E;
- d is the maximum degree among all the vertices in H (the degree $d(v)$ of a vertex v is the number of hyperedges in E containing v);
- r is the rank of the hypergraph, that is the maximum size of the hyperedges in E;
- N is the number of bags in the computed tree-decomposition.

A.1 Data structures

We present the different data structures required to run the algorithms. These data structures which were not described in the previous section are used by k-PH in order to improve its complexity. They are updated in particular when calculating the connected components, also for reasons of complexity. Before, note that the size of an input is in $\Theta(n + e + \Sigma_{E_i \in E}|E_i|)$ which can be simplified by $\Theta(n + \Sigma_{E_i \in E}|E_i|)$.

(1) To represent and manage hypergraphs, we use arrays:

- One is indexed by the set of hyperedges E. For a given hyperedge E_i, it represents the list of the vertices included in E_i. Its size and the cost of its initialization are in $\Theta(e + \Sigma_{E_i \in E}|E_i|)$.
- Another one is indexed by the set of vertices V. For a given vertex v, it represents the list of the hyperedges containing v. Its size and the cost of its initialization are in $\Theta(n + \Sigma_{E_i \in E}|E_i|)$.
- To represent a partial hypergraph $H' = (V, E')$ of $H = (V, E)$, we only need an array of Booleans indexed on the set E of hyperedges which specifies the edges belonging to E'. Its size and the cost of its initialization are in $\Theta(e)$.

(2) An array B memorizes the set B of bags associated to the computed tree-decomposition. It is an array of lists indexed by its rank during the computation. Each list represents the set of vertices of the corresponding bag. As the number of bags is at most n (necessarily, we have $N \leq n$), the cost of its initialization is in $\Theta(n)$ and its size is in $\Theta(n + \Sigma_{B_i \in B}|B_i|)$ which is exactly the size of the result of k-PH.

(3) Throughout the execution, we manage a queue Q memorizing the connected components. So Q can express as a list of connected component X_{i_j}, that is a list of sets. The size of such a data structure is bounded by the number of vertices since it is a partition of a subset of the vertices of the hypergraph. The cost of its initialization is in $\Theta(1)$ while at some point of the computation, its size is in $O(n)$.

(4) Once B_i has been calculated, for the calculation of the associated connected components, we will use a marking table. This array, called $status[]$, is indexed by the set V of vertices. For a given vertex v, we have:

- $status[v] = -1$ if the vertex v belongs to a built bag,
- $status[v] = j$ if the vertex v has been processed when computing the connected components of H' for the j-th time,
- $status[v] = 0$ if the vertex v has never been considered.

The cost of its initialization and its size are in $\Theta(n)$.

(5) For the computation of bags, we use several data structures that allow to improve the efficiency of k-PH. These data structures will be assigned after the computation of a new bag B_i, during the computation of the new connected components X_{i_j}:

- A set of candidate hyperedges associated to each new connected component X_{i_j}. It is a list of hyperedges denoted S_{i_j}. So, the data structure S is a list of lists of hyperedges. The cost of the initialization for S, for one S_i (and thus for one S_{i_j}) is feasible in constant time, i.e. in $\Theta(1)$. Note that in the description of the algorithm k-PH, when a set X_i is removed from the queue Q to compute a new bag B_i, such a list S_{i_j} is denoted S_i.
- For a new connected components X_{i_j} which will be considered later to find a future bag, we memorize the possible potential parent bags (to connect it to the future new bag in the resulting tree-decomposition). These are bags B_ℓ already calculated (i.e. such that $1 \leq l \leq i$) and such that there exists a hyperedge intersecting simultaneously X_{i_j} and B_ℓ. Such bags will be represented in a list denoted P_{i_j}. So, we need a data structure P which is a list of lists of bags, and, the cost of the initialization for P, for one P_i (and thus for one P_{i_j}) is feasible in constant time, i.e. in $\Theta(1)$.

- An array of lists called *Bag_lists* represents the connections between hyperedges and bags: this array is indexed by the hyperedges E_i, and for a given E_i, the associated list contains all the bags B_j such that $B_j \cap E_i \neq \emptyset$. The cost of the initialization of *Bag_lists* is feasible in $\Theta(e)$ and its size is in $O(e \cdot N)$.

A.2 Implementation and Time Complexity Analysis

We can now analyze the complexity of *k-PH*. To do so, we must analyze each of its different steps, that is computation of a new bag B_i, update of H', and the computation of new connected components. In *k-PH*, the computation of a new bag is realized using a heuristic $\mathcal{H}$. Since different heuristics can be considered, we assume that its time complexity is in $O(h)$. Note however that, whatever the heuristic considered, if a vertex v belongs to a new bag B_i, $status[v]$ takes the value -1.

After the evaluation of the complexity of *k-PH*, we assess the time complexity of the heuristic considered in Theorem 3 as an illustration.

5.0.1 Update of H'

Once B_i is calculated, H' has to be updated by removing hyperedges $E_j \in S_i$ such that $E_j \cap N_i \not\subset B_i$ and $E_j \cap B_i \not\subset N_i$. To find such edges E_j, we simply browse S_i, and select the hyperedges which satisfy these two conditions:

- $(E_j \cap B_i \not\subset N_i)$ E_j must contain at least one vertex x of X_i that belongs both to B_i and E_j: such a vertex verifies $status[x] = -1$;
- $(E_j \cap N_i \not\subset B_i)$ E_j must contain at least one vertex y of N_i that does not belong to B_i: such a vertex verifies $status[x] > 0$.

The complexity is thus related to the size of S_i, and to the cost to visit each E_j. So, the cost is in $O(|S_i| \cdot r)$ because the size $|E_j|$ of the hyperedges is at most m. Finally, each hyperedge can be removed from H' in constant time using the array representing this partial hypergraph.

A.3 Calculation of Connected Components

In this part, we describe the algorithm *Comp-CC*. This algorithm computes, in a classical way, the connected components X_{i_j} deduced from B_i in X_i. However, at the same time, for efficiency reasons, it also computes the sets S_{i_j} of the candidate hyperedges for the construction of the future bags, and the sets P_{i_j} of the bags already found and that will be candidates to be bags connecting the future bags to those already obtained (they will then be called *parent bags*). That is why we give below a detailed description of this algorithm.

The *Comp-CC* algorithm actually considers as inputs the partial hypergraph $H' = (V, E')$, the set of vertices X_i, the newly created bag B_i, the array *status* and the array of bags *Bag_lists*. Note that the version we give here of the *Comp-CC* algorithm has more arguments than the one used in the *k-PH* algorithm. This now makes it possible to specify the implementation details needed for complexity analysis, details that had been deliberately omitted before in order to lighten the presentation of the *k-PH*.

As outputs (and inputs) of the algorithm, we have:

- the set CC of connected components of the expanded subhypergraph $H'[[X_i \setminus B_i]]$. These connected components are noted X_{i_j};
- the set of sets S, where each set $S_{i_j} \in S$ is associated with the connected component X_{i_j} and such that S_{i_j} represents the candidate hyperedges for this component;
- the set of sets P, where each set $P_{i_j} \in P$ is associated with the connected component X_{i_j} and such that P_{i_j} represents the candidate parent bags for this component.

This algorithm (see Algorithm 2) works in a similar way to the one that calculates the connected components in graphs, with a depth first search, except that it is adapted to the case of hypergraphs, and that it must update data structures used in our framework. Note that in the code of the algorithm, the index j will be noted x, x being the vertex from which a new descent will be made to find a new connected component.

The first step (lines 1-5) is devoted to initialize and update some data structures. Before, recall that the value of i denotes the number of times the algorithm *Comp-CC* has been called since the beginning of the computation of the partial hypergraph. In this step, we also update *status* (lines 4-5) in order to take into account the last computed bag (if any). Then a new descent in the hypergraph is performed (line 7) for each vertex x of X_i which has not been reached yet during the current call to *Comp-CC* (i.e. $status[x] < i$) and which does not belong yet to a built bag (i.e. $status[x] \geq 0$). It allows us to build a new connected component X_{i_x} for which x constitutes the first vertex (line 12). This vertex is

inserted in a stack called *Sta* and initializations are realized (lines 8-10). At the same time, x is marked as visited for the current call to *Comp-CC* (line 11). From there, the connected component X_{i_x} is computed starting from vertex x (lines 13-27). To do this, we first select an untreated vertex y in *Sta* and remove it from *Sta* (line 14). From this vertex, all hyperedges E_j containing y must be considered (note that the considered hyperedges E_j are those of which a part appears in the expanded subhypergraph $H'[[X_i \setminus B_i]]$). First, if all the vertices of E_j which belong to a computed bag appears in B_i, it means that E_j is necessarily connected to B_i (lines 16-17). Afterwards, these hyperedges are exploited by determining the already computed bags that intersect E_j because the latter are potential parents of the bag that will be constructed from X_{i_x} (lines 18-19). Then, the vertices z of each E_j hyperedge are processed depending on their status (lines 20-27). If the vertex z belongs to an already constructed bag ($1 \leq status[z] \leq i$), the hyperedges containing it must then be selected (lines 21-22) because they will be candidate hyperedges for the future construction of a bag from X_{i_x}. Otherwise, if z has never been reached during the current call (i.e. $0 \leq status[z] < i$), z is added both to the stack and the component X_{i_x} and is marked as visited for the current called (lines 24-27). In another case, i.e. if $status[z] = num$, the processing concerning z has already been carried out and so of course there is nothing else to do. Each descent into the hypergraph continues until the stack is empty. Once the stack is empty, X_{i_x} contains a new connected component that will be added to CC, and the process can continue until all the vertices of X_i have been reached. Finally the related data structures are adequately updated (lines 28-32). Note that, in some particular cases, P_{i_x} and S_{i_x} are empty at the end of loop of lines 6-27. This occurs when the new connected component X_{i_x} cannot be linked to an existing bag (e.g. for the first call to *Comp_CC*). When this phenomenon happens for another call than the first one, it means that H' has several connected components. In such a case, we consider for S_{i_x} all the hyperedges which are included in X_{i_x} (lines 28-29).

Remark that by marking a vertex as visited by setting its status to the value of i, we avoid resetting status at each called of *Comp_CC*.

Algorithm 2: Comp-CC

Input: Rank i of the current call, a hypergraph $H' = (V, E')$, a set of vertices X_i, a bag B_i, an array of bags *Bag_list*, an array *status*, a set of sets of hyperedges S, a set of sets of parents bags P

Output: A set of connected components CC, a set of sets of hyperedges S, a set of sets of parents bags P

```
1  CC ← ∅
2  P_i ← ∅
3  S_i ← ∅
4  for x ∈ B_i do
5      status[x] = −1
6  for x ∈ X_i do
7      if 0 ≤ status[x] < i then
8          Sta ← {x}
9          P_{i_x} ← ∅
10         S_{i_x} ← ∅
11         status[x] ← i
12         X_{i_x} ← {x}
13         while Sta ≠ ∅ do
14             Select y from Sta and remove it
15             for E_j ∈ E' | y ∈ E_j do
16                 if {z ∈ E_j | status[z] = −1} ⊆ B_i then
17                     Bag_list[E_j] ← B_i
18                 for bag ∈ Bag_list[E_j] do
19                     P_{i_x} ← P_{i_x} ∪ {bag}
20                 for z ∈ E_j \ {y} do
21                     if status[z] = −1 then
22                         S_{i_x} ← S_{i_x} ∪ {E_j}
23                     else
24                         if status[z] < i then
25                             Add z to Sta
26                             status[z] = i
27                             X_{i_x} ← X_{i_x} ∪ {z}
28         if P_{i_x} = ∅ then
29             S_{i_x} ← {E_j ∈ E' | E_j ⊆ X_{i_x}}
30         P_i ← P_i ∪ {P_{i_x}}
31         S_i ← S_i ∪ {S_{i_x}}
32         CC ← CC ∪ {X_{i_x}}
33 return CC
```

Proposition 1. The time complexity of *Comp-CC* for a bag B_i is in $O(|X_i| \cdot d \cdot n)$.

Proof: The initialization phase for the data structures (lines 1 to 3) can be done in constant time while the update of *status* is achieved in $O(|B_i|)$. There will be exactly $|X_i|$ passes in the **for** loop (lines 6-27) and thus, the test of line 7 will be realized $|X_i|$ times. However, the number of times the test is true will be exactly equal to the number of connected components, i.e. $|CC|$ times. The initializations of lines 8-12 can be carried out in constant time. Overall, they will be realized $|CC|$ times. As all the vertices of X_i will be processed, and at most once, there will be globally less than $|X_i|$ passes in the **while** loop (because of the vertices in B_i). In fact, there will be exactly as many passes globally in this loop as there will be vertices in the new computed connected components, namely precisely $|\bigcup X_{i_x}| < |X_i|$. For a given vertex y, the loop of line 15 is performed at most $d(y)$ times, i.e. the number of hyperedges to which it belongs in H. This number can be bounded by d. For one pass in this loop:

- Lines 16-17 can be achieved in $O(|E_j| \cdot |B_i|)$, and so in $O(r \cdot k)$ since we have $|E_j| \leq r$ and $|B_i| \leq k$.
- The processing of line 18 is possible in constant time (this is possible by using an array of Booleans for the temporary management of this step with a global cost in $\Theta(n)$ for *Comp-CC*), the cost of lines 18 and 19 is of the order of $|Bag\_list[E_j]|$, i.e. less than N.
- For one vertex y, there are exactly $|E_j|-1$ entries through the loop of line 20, which is bounded by r because for all $E_j \in E$, we have $|E_j| \leq r$. For a given vertex z, the processing performed in lines 21-27 can be done in constant time (for S_{i_x} as for P_{i_x}, this is possible by using an array of Booleans for the temporary management of this step with a global cost in $\Theta(n)$ for *Comp-CC*).

Summarizing, this means that for a given vertex y, the cost of lines 15-27 is at most by $O(d \cdot (N + r \cdot k))$ (N.B.: lines 18-19 can be replaced by a single line like $P_{i_x} \leftarrow P_{i_x} \cup Bag\_list[E_j]$). As globally, we will have less than $|X_i|$ vertices y to process, the global cost of lines 13-27 is thus bounded by $O(|X_i| \cdot d \cdot (N + r \cdot k))$. Finally, performing lines 30-32 is feasible in constant time. So, their overall cost is in $O(|CC|)$. Thus we have $O(|X_i| \cdot d \cdot (N + r \cdot k) + |CC| + |X_i|)$, and as necessarily we have $|CC| \leq |X_i|$, the complexity of *Comp-CC* for a B_i bag is thus $O(|X_i| \cdot d \cdot (N + r \cdot k))$. We have $N \leq n$ and $r \leq k$. So, as k is a constant, we obtain a complexity in $O(|X_i| \cdot d \cdot n)$. □

A.4 Complexity of k-PH

We can now assess the complexity of *k-PH*. For this, we assume that the cost of the considered heuristic $\mathcal{H}$ is h.

Theorem 2 (proof). The time complexity of *k-PH* is in $O(N(h + e \cdot r + n \cdot d(N + r)))$.

Proof: First of all, it is obvious that the cost of initializing the data structures is lower than the total cost of the algorithm. The **while** loop is performed at most N times, the number of bags in the resulting tree-decomposition. Each time there is a pass through this loop, a new bag is calculated, and the cost of the calculation of a new bag in exactly $\Theta(h)$. Note that the $Bag\_list[]$ update does not appear in *k-PH*. This can be done just after the computation of the bag B_i and the cost of this update is achievable in $O(k \cdot d \cdot N)$. We know that the cost of updating H' is $O(|S_i| \cdot r)$ with $|S_i| \leq e$ and that, for a new bag B_i, the cost of a *Comp-CC* call is $O(|X_i| \cdot d \cdot (N + r))$. Since the size of X_i is bounded by n, the cost for each *Comp-CC* is actually $O(n \cdot d(N + r))$. Such a size for X_i may occur if the 2-section of the input hypergraph is a (k+1)-tree, formed by a single path. Thus, at each pass, the new connected component will contain all the vertices not yet included in a bag, and there will be only one vertex less in the next connected component. Finally, the total cost of *k-PH* is therefore $O(N(h + e \cdot r + n \cdot d(N + r)))$. □

To give an illustration of this complexity, we now propose an evaluation of the complexity of the heuristic $\mathcal{H}_\alpha$ considered for Theorem 3. This one first chooses a bag B_ℓ with the largest intersection with N_i and then, chooses as the first new hyperedge to be in B_i a hyperedge that shares the maximum number of vertices with B_ℓ. The other hyperedges to be added to B_i are chosen in descending order of the size of their intersection with B_ℓ. This processing takes as inputs a connected component X_i, the set of candidate hyperedges S_i (a list stored in S), and the list of bags P_i (a list stored in P) already calculated and which can connect the new bag to one already obtained. So, we analyze first, the selection of B_ℓ, then the computation of B_i:

- Selection of B_ℓ.
  - The first step is to calculate the set N_i. This is possible by browsing S_i, and inserting in N_i, for each hyperedge E_j, the vertices that are not in X_i. Note that X_i is represented by a list, but for efficiency, we can also represent X_i by an array of Booleans indexed by the vertices. At the initialization step of *k-PH*, this array is initialized to zero and when processing a new set X_i, just assign its elements to 1. Once the calculation of B_i has been performed, all corresponding elements are reset to zero. This will result in a linear cost in the size of X_i. We

can do the same for N_i. Remember that the cost of visiting a hyperedge E_j is in $\Theta(|E_j|)$, and therefore in $O(r)$. Thus, the cost of the calculation of N_i is $O(|X_i| + |S_i| \cdot r)$.

- The calculation of B_ℓ is performed using N_i and P_i. To do this, we have to consider each element of P_i, i.e., all the bags already calculated that are candidates. For each bag, its elements present in N_i are counted. As the size of the bags is at most k, the complexity is $O(|P_i| \cdot k)$.

To summarize, the time complexity of the calculation of B_ℓ is therefore $O(|X_i| + |S_i| \cdot r + |P_i| \cdot k)$.

- Computation of B_i. All the hyperedges in S_i must be visited and for each one, the size of its intersection with B_ℓ is computed. To do this, we can proceed as above, using an array of Booleans initialized to 1 for all the elements of B_ℓ. Then for each hyperedge, we consider the size of its intersection with B_ℓ. Moreover, we can order the hyperedges of S_i according to the size of this intersection. To carry out this sorting, it is possible to have a data structure which would proceed by addressing in an array of size r and whose each entry would contain a list memorizing the hyperedges of the corresponding size. We just have to visit the set of ordered hyperedges and check which ones can be added to B_i. This treatment can be done in $O(|S_i| \cdot r)$ since all the hyperedges in S_i will be examined and the cost of treatment of a hyperedge is linear in its size, that is to say in $O(r)$. Note that at the end of this treatment, if a vertex v belongs to this new bag B_i, the value i is assigned to $status[v]$.

Thus, the cost of executing this heuristic is $O(|X_i| + |S_i| \cdot r + |P_i| \cdot k)$. Knowing that $|X_i| \leq n$, $|S_i| \leq e$ and $|P_i| \leq n$, one can thus bound the cost of this heuristic by $O(n + e \cdot r + n \cdot k) = O(n \cdot k + e \cdot r)$.

This gives an illustration of the time complexity of k-PH by specifying $O(N(h + e \cdot r + n \cdot d(N + r))) = O(N(n \cdot k + e \cdot r + n \cdot d(N + r)))$. This expression can be rephrased if we consider certain upper bounds. Indeed, we know that $N \leq n$, we thus obtain the complexity $O(n(n \cdot k + e \cdot r + n \cdot d(n + r))) = O(n \cdot n \cdot k + n \cdot e \cdot r + n \cdot n \cdot d \cdot n + n \cdot n \cdot d \cdot r) = O(n^2 \cdot k + n \cdot e \cdot r + n^3 \cdot d + n^2 \cdot d \cdot r) = O(n^2 \cdot (k + n \cdot d + r \cdot d) + n \cdot e \cdot r)$. On the other hand, if we also consider that the width $k - 1$ is a constant, and therefore that the treatments will only consider hyperedges whose size is less than or equal to k, this complexity can be reduced to $O(n^3 \cdot d + n \cdot e)$.

References

[1] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 631–637, 1995.

[2] U. Montanari. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Artificial Intelligence*, 7:95–132, 1974.

[3] R. Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.

[4] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[5] R. Dechter. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Morgan and Claypool Publishers, 2013.

[6] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30:479–513, 1983.

[7] N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of treewidth. *Algorithms*, 7:309–322, 1986.

[8] B.-M. Bui-Xuan, J.A. Telle, and M. Vatshelle. Rank-width and vertex-minors. *Theoretical Computer Science*, 412(39):5187–5204, 2011.

[9] J. Jeong, S.H. Sæther, and J.A. Telle. Maximum matching width: new characterizations and a fast algorithm for dominating set. In *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC)*, 2015.

[10] J. Gajarsky, M. Lampis, and S. Ordyniak. Parameterized Algorithms for Modular-Width. *CoRR abs/1308.2858*, 2013.

[11] P. Wollan. The structure of graphs not admitting a fixed immersion. *Journal of Combinatorial Theory Series B*, 110:47–66, 2015.

[12] J. Nesetril and P. Ossona de Mendez. Bounded height trees and tree-depth. In *Sparsity: Graphs, Structures, and Algorithms, Algorithms and Combinatorics 28*, pages 115–144. Springer, 2012.

[13] B. Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75, 1990.

[14] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124:243–282, 2000.

[15] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146:43–75, 2003.

[16] S. de Givry, T. Schiex, and G. Verfaillie. Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In *Proceedings of AAAI*, pages 22–27, 2006.

[17] S. Arnborg, D. Corneil, and A. Proskuroswki. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics*, 8:277–284, 1987.

[18] K. Hirata, M. Kuwabara, and M. Harao. On Finding Acyclic Subhypergraphs. In *Fundamentals of Computation Theory, 15th International Symposium, FCT 2005, Lübeck, Germany, August 17-20, 2005, Proceedings*, volume 3623 of *Lecture Notes in Computer Science*, pages 491–503, 2015.

[19] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13 (3):566–579, 1984.

[20] P. Jégou and C. Terrioux. A new filtering based on decomposition of constraint sub-networks. In *22nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010, Arras, France, 27-29 October 2010 - Volume 1*, pages 263–270, 2010.

[21] L.W. Beineke and R.E. Pippert. Properties and characterizations of k-trees. *Mathematika*, 18:141–151, 1971.

[22] D. R. Karger and N. Srebro. Learning Markov networks: maximum bounded tree-width graphs. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*, pages 392–401. ACM/SIAM, 2001.

[23] Nathan Srebro. Maximum likelihood Markov networks: An algorithmic approach. *Master's thesis, Massachusetts Institute of Technology*, 2000.

[24] D. Shahaf, A. Chechetka, and C. Guestrin. Learning Thin Junction Trees via Graph Cuts. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, volume 5 of *JMLR Proceedings*, pages 113–120. JMLR.org, 2009.

[25] A. Fix, J. Chen, E. Boros, and R. Zabih. Approximate MRF Inference Using Bounded Treewidth Subgraphs. In *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part I*, volume 7572 of *Lecture Notes in Computer Science*, pages 385–398. Springer, 2012.

[26] S. Nie, C. Polpo de Campos, and Q. Jih. Learning Bayesian Networks with Bounded Tree-width via Guided Search. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 3294–3300. AAAI Press, 2016.

[27] C. Berge. *Graphs and hypergraphs.* North-Holland, 1973.

[28] A. Hajnal and J. Suranyi. Uber die Auflosung von Graphen in vollstandige Teilgraphen. *Ann. Univ. Sci. Budapest Eotvos. Sect. Math. 1*, MR21 -1944:113–121, 1958.

[29] G.A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg 25*, MR24 -57:71–76, 1961.

[30] D.R. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15:835–855, 1965.

[31] C. Berge. *Graphes et Hypergraphes.* Dunod-France, 1970.

[32] R. Fagin. Degrees of Acyclicity for Hypergraphs and Relational Database Schemes. *J. ACM*, 30(3):514–550, 1983.

[33] H. Dell, C. Komusiewicz, N. Talmon, and M. Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In *IPEC*, pages 30:1–30:12, 2018.

[34] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming.* Elsevier, 2006.

[35] M. C. Cooper, S. de Givry, M. Sánchez, T. Schiex, and M. Zytnicki. Virtual arc consistency for weighted csp. In *AAAI*, pages 253–258, 2008.

[36] W. Fischl, G. Gottlob, D. M. Longo, and R. Pichler. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. In *PODS*, pages 464–480, 2019.

[37] A. Bonifati, W. Martens, and T. Timm. An Analytical Study of Large SPARQL Query Logs. *PVLDB*, 11(2):149–161, 2017.

[38] A. Bonifati, W. Martens, and T. Timm. Navigating the Maze of Wikidata Query Logs. In *WWW*, pages 127–138, 2019.

[39] R. Pottinger and A. Y. Halevy. MiniCon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2011.

[40] M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, D. Santoro P. Papotti, and E. Tsamoura. Benchmarking the chase. In *PODS*, pages 37–52, 2017.

[41] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 289–298. ACM Press, 2006.

[42] T. Sang, P. Beame, and H. A. Kautz. Performing Bayesian inference by weighted model counting. In *AAAI*, pages 475–482, 2005.