Computing Max-SAT Refutations using SAT Oracles

Matthieu Py, Mohamed Sami Cherif, Djamal Habet

▶ **To cite this version:**

Matthieu Py, Mohamed Sami Cherif, Djamal Habet. Computing Max-SAT Refutations using SAT Oracles. International Conference on Tools with Artificial Intelligence (ICTAI), Nov 2021, Visioconférence, France. 10.1109/ICTAI52525.2021.00066 . hal-03737738

HAL Id: hal-03737738

https://amu.hal.science/hal-03737738v1

Submitted on 25 Jul 2022

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

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

Computing Max-SAT Refutations using SAT Oracles

Matthieu Py, Mohamed Sami Cherif and Djamal Habet

Aix-Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
{matthieu.py, mohamed-sami.cherif, djamal.habet}@univ-amu.fr

***Abstract*—Adapting a resolution refutation for SAT into a Max-SAT resolution refutation without increasing considerably the size of the refutation is an open question. This paper contributes to this topic by introducing an algorithm, called *substitute generation*, able to adapt any resolution refutation to get a Max-SAT refutation using SAT oracles. This algorithm is able to efficiently adapt k-stacked diamond patterns, whose transformation is exponential in the literature.**

***Index Terms*—Max-SAT, Resolution Refutation, Max-SAT Resolution**

I. Introduction

Given a Boolean formula in Conjunctive Normal Form (CNF), the Max-SAT problem consists in determining the maximum (resp. minimum) number of clauses that it is possible to satisfy (resp. falsify) by an assignment of the variables, while the SAT problem consists in verifying the existence of an assignment which satisfies all the clauses in the formula. A well-known proof system for Max-SAT is Max-SAT resolution [21] which extends the resolution rule [27] used in the context of SAT. Max-SAT resolution plays a prominent role in Max-SAT as it is the most studied inference rule, both in theory and practice [1], [8], [19], [21], [23].

In the context of SAT, an unsatisfiable formula can be refuted with a sequence of resolution steps which leads to the empty clause. However, while resolution adds the conclusion to the formula, the Max-SAT resolution rule replaces the premises by the conclusions. As such, switching from a read-once resolution refutation, where each clause is used once, to get a valid Max-SAT transformation deducing the empty clause (referred to as Max-SAT refutation or simply max-refutation as in [25]) is possible and well-known [12]. To this aim, it is sufficient to replace each resolution step by a Max-SAT resolution step. However, the adaptation of any resolution refutation to get a valid max-refutation is an established problem. Bonet et al. state that "*it seems difficult to adapt a classical resolution proof to get a Max-SAT resolution proof, and it is an open question if this is possible without increasing substantially the size of the proof*" [21]. Recently, the split rule was used to adapt any resolution refutation into a Max-SAT refutation, but the proposed adaptation is exponential in the worst case [25].

This paper attempts to contribute to this question by proposing an algorithm, called *substitute generation*, which is able to use any resolution refutation to compute a Max-SAT refutation, i.e. a Max-SAT equivalence-preserving transformation from an initial unsatisfiable formula to an equivalent one containing an empty clause. The idea of this algorithm is to follow the resolution steps of the resolution refutation, to replace each of these steps by a Max-SAT resolution step, and to apply it on the current formula. In the case of a non-read-once resolution refutation, substitutes for missing clauses consumed by Max-SAT resolution are generated by calling a SAT oracle. This algorithm shows a remarkable efficiency on diamond patterns which were shown exponential for the adaptation in [25] but are linear using the substitute generation algorithm.

This paper is organized as follows. Section II includes some necessary definitions and notations. In Section III, we introduce the substitute generation algorithm to adapt any resolution refutation into a max-refutation. Section IV includes a detailed example of algorithm's execution. In Section V, we study a particular pattern of resolution refutations whose adaptation is exponential in [25] and we show that can be linearly adapted using the substitute generation algorithm. Finally, we conclude and discuss future work in Section VI.

II. Preliminaries

A. Definitions and Notations

Let X be the set of propositional variables. A literal l is a variable $x \in X$ or its negation $\overline{x}$. A clause c is a disjunction (or a set) of literals $(l_1 \vee l_2 \vee ... \vee l_k)$. A formula in Conjunctive Normal Form (CNF) ϕ is a conjunction (or a multiset) of clauses $\phi = c_1 \wedge c_2 \wedge ... \wedge c_m$. An assignment $I : X \rightarrow \{true, false\}$ maps each variable to a boolean value and can be represented as a set of literals. A literal l is satisfied (resp. falsified) by an assignment I if $l \in I$ (resp. $\overline{l} \in I$). A clause c is satisfied by an assignment I if at least one of its literals is satisfied by I, otherwise it is falsified by I. The empty clause $\square$ contains zero literals and is always falsified. A clause c opposes a clause c' if c contains a literal whose negation is in c', i.e. $\exists l \in c,\ \overline{l} \in c'$. A clause c subsumes a clause c' if each literal of c is a literal of c', i.e. $\forall l \in c,\ l \in c'$. We denote $var(c)$ the variables appearing in the clause c. A CNF formula ϕ is satisfied by an assignment I, that we call model of ϕ, if each clause $c \in \phi$ is satisfied by I, otherwise it is falsified by I. Solving the Satisfiability (SAT) problem consists in determining whether there exists an assignment I that satisfies a given CNF formula ϕ. In the case

where such an assignment exists, we say that ϕ is satisfiable, otherwise we say that ϕ is unsatisfiable or inconsistent. The cost of an assignment I, denoted $cost_I(\phi)$, is the number of clauses falsified by I. The Maximum Satisfiability (Max-SAT) problem is an optimization extension of SAT which, for a given CNF formula ϕ, consists in determining the maximum number of clauses that can be satisfied by an assignment of the variables. Equivalently, it consists in determining the minimum number of clauses that each assignment must falsify, i.e. $\min_I \ cost_I(\phi)$.

B. Resolution Refutations in SAT

To certify that a CNF formula is satisfiable, it is sufficient to simply exhibit a model of the formula. On the other hand, to prove that a CNF formula is unsatisfiable, we need to refute the existence of a model. To this end, we can exhibit a SAT refutation which consists of a sequence of equivalence-preserving transformations (in the sense of SAT as defined below) starting from the formula and ultimately deducing an empty clause.

Definition 1 (SAT Equivalence)**.** *Let ϕ and ϕ' be two CNF formulas. We say that ϕ is equivalent (in the sense of SAT) to ϕ' if for any assignment $I : var(\phi) \cup var(\phi') \rightarrow \{true, false\}$, I is a model of ϕ if and only if I is a model of ϕ'.*

A well-known SAT refutation system is based on an inference rule for SAT called resolution [27]. Refutations in this system are referred to as resolution refutations. The resolution rule, defined below, deduces a clause called resolvent from two opposed clauses which can be added to the formula while preserving SAT equivalence. Resolution plays an important role in the context of Conflict Driven Clause Learning (CDCL) [22]. Furthermore, it was shown that CDCL can polynomially simulate general resolution [24]. As showcased in Example 1, a resolution refutation can be represented as a Directed Acyclic Graph (DAG) whose nodes are clauses in the refutation either having two or zero incoming arcs (resp. if they are resolvents or clauses of the initial formula).

Definition 2 (Resolution [27])**.** *Given two clauses $c_1 = (x \vee A)$ and $c_2 = (\overline{x} \vee B)$, the resolution rule is defined as follows:*

$$\frac{c_1 = (x \vee A) \quad c_2 = (\overline{x} \vee B)}{c_3 = (A \vee B)}$$

Example 1. *We consider the CNF formula $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$. A resolution refutation of ϕ is represented as a DAG in Figure 1.*

Many restricted classes of resolution refutations have been studied in the literature namely linear resolution [20], unit resolution [13], input resolution [13], regular resolution [28], read-once resolution [14] and tree (or tree-like) resolution [2] refutations among others. In particular, a resolution refutation is tree-like if every intermediate clause, i.e. resolvent, is used at most once in the refutation. It is known that the DPLL algorithm [9] on unsatisfiable instances corresponds to tree resolution refutations [11]. Similarly, a resolution refutation is

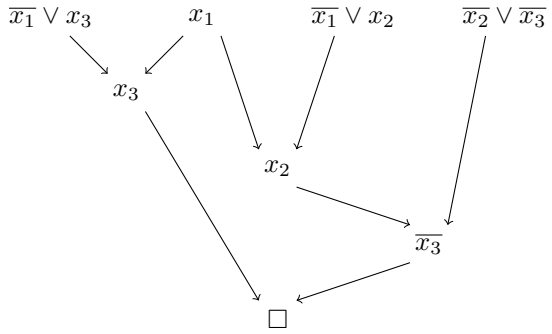

Fig. 1. Resolution refutation

read-once if each clause is used at most once in the refutation. Clearly, read-once resolution refutations are also tree-like since they form a restricted class of tree resolution refutations. It was shown in [14] that there exists unsatisfiable CNF formulas which cannot be refuted using read-once resolution. A resolution is regular if every variable is resolved on at most once in each branch of the DAG, i.e. path from a clause of the initial formula to the empty clause. It was shown that CDCL without restarts can polynomially simulate regular resolution [7]. Finally, a resolution refutation is semi-tree-like if, for any branch of the DAG, at most one clause is not read-once (i.e. used several times as a premise of a resolution step).

Example 2. *We consider the refutation of ϕ in Example 1. The refutation is tree-like and semi-tree-like but it is not read-once since clause (x_1) is used two times as a premise of a resolution step. The refutation is also regular as every variable is resolved on at most once in every branch of the DAG in Figure 1.*

C. Max-SAT Proofs

Several complete proof systems for Max-SAT were introduced in the literature, namely the Max-SAT resolution Calculus in [21] and the Clause Tableau Calculus in [18]. In particular, Max-SAT resolution, one of the first known complete systems for Max-SAT, was inspired from resolution [27]. The aim of complete Max-SAT systems is not to refute the formula per se but to prove the Max-SAT optimum of a given CNF formula, i.e. the minimum number of falsified clauses. The formula is thus refuted as many times as its optimum through equivalence-preserving transformations in the sense of Max-SAT as defined below.

Definition 3 (Max-SAT Equivalence)**.** *Let ϕ and ϕ' be two CNF formulas. We say that ϕ is equivalent (in the sense of Max-SAT) to ϕ' if for any assignment $I : var(\phi) \cup var(\phi') \rightarrow \{true, false\}$, we have $cost_I(\phi) = cost_I(\phi')$.*

The Max-SAT resolution proof system relies on an inference rule that extends resolution for Max-SAT. Other than the resolvent clause, this rule, called Max-SAT resolution and defined below, introduces new clauses referred to as compensation clauses and essential to preserve Max-SAT equivalence.

As a sound and complete rule for Max-SAT [21], Max-SAT resolution plays an important role in the context of Max-SAT theory and solving. In particular, it is extensively used and studied particularly in the context of Branch and Bound algorithms for Max-SAT [1], [8], [15], [19] and more marginally in the context of SAT-based ones [12], [23]. For a given CNF formula, it is possible to generate a Max-SAT resolution proof of its optimum by applying the saturation algorithm [21] to deduce empty clauses. As showcased in Example 3, a Max-SAT resolution proof can also be represented as a DAG whose nodes are multisets of clauses either having two or zero incoming arcs (resp. if they are clauses produced by a Max-SAT resolution step or clauses of the initial formula). A Max-SAT transformation consisting of a sequence of application of Max-SAT rules and deducing the empty clause from a given unsatisfiable formula is referred to as Max-SAT refutation or simply max-refutation [25].

Definition 4 (Max-SAT resolution [5], [16], [21])**.** *Given two clauses $c_1 = x \vee A$ and $c_2 = \overline{x} \vee B$ with $A = a_1 \vee ... \vee a_s$ and $B = b_1 \vee ... \vee b_t$. The Max-SAT resolution rule is defined as follows:*

$$\frac{c_1 = x \vee A \qquad c_2 = \overline{x} \vee B}{\begin{array}{c} c_r = A \vee B \\ cc_1 = x \vee A \vee \overline{b_1} \\ cc_2 = x \vee A \vee b_1 \vee \overline{b_2} \\ \vdots \\ cc_t = x \vee A \vee b_1 \vee ... \vee b_{t-1} \vee \overline{b_t} \\ cc_{t+1} = \overline{x} \vee B \vee \overline{a_1} \\ cc_{t+2} = \overline{x} \vee B \vee a_1 \vee \overline{a_2} \\ \vdots \\ cc_{t+s} = \overline{x} \vee B \vee a_1 \vee ... \vee a_{s-1} \vee \overline{a_s} \end{array}}$$

where c_r is the resolvent clause and $cc_1, ..., cc_{t+s}$ are compensation clauses.

Remark 1. *Unlike resolution, the Max-SAT resolution rule replaces the premises by the conclusions.*

Example 3. *We consider the CNF formula from Example 1. A hand-made Max-SAT resolution refutation of ϕ was proposed in [21] and is represented in Figure 2.*

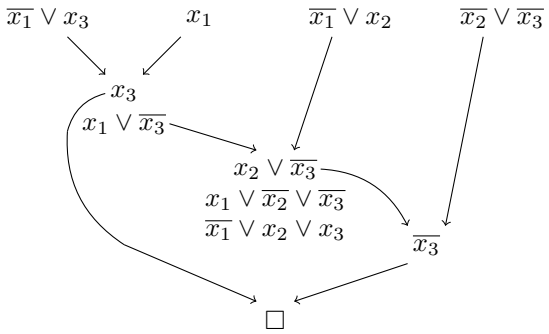

Fig. 2. A hand-made Max-SAT resolution proof

Because of the specificity of Max-SAT resolution, which replaces the premises by the conclusions, it is not easy to adapt a resolution refutation to obtain a max-refutation. Indeed, in resolution refutations, several resolution steps can share the same premise, because the premises are not consumed after the application of a resolution. On the other hand, the premises of a Max-SAT resolution step are consumed after its application. Consequently, the immediate adaptation of a resolution refutation for Max-SAT is only possible if it is read-once [12], which is the particular case where each clause is used at most once as a premise of a resolution step [14]. Adapting any resolution refutation into a max-refutation without substantially increasing its size was only recently shown possible in [25] and [10] for particular cases of resolution. The proposed adaptation relies on the split rule, defined below, which intuitively allows to duplicate a clause by adding one literal. The split rule has been widely studied in the context of Max-SAT [6], [10], [17], [25], [26] and can particularly be used to adapt tree-like regular resolution refutations by iteratively fixing the non-read-once clauses, as showcased in Example 4. The adaptation can also be extended to tree-like, semi-tree-like and unrestricted resolution refutations, but with the disadvantage that the size of the obtained max-refutation in the case of unrestricted resolution can be exponential (refer to Table I).

Definition 5 (Split rule)**.** *Given a clause $c_1 = (A)$ where A is a disjunction of literals and x a variable, the Max-SAT split rule is defined as follows:*

$$\frac{c_1 = (A)}{c_2 = (x \vee A) \quad c_3 = (\overline{x} \vee A)}$$

Remark 2. *Like the Max-SAT resolution rule, the split rule replaces the premise by the conclusions.*

Resolution Refutation	Size of the max-refutation
Read-once	Linear [21], [12]
Tree-like regular	Linear [25], [10]
Tree-like	Linear [25], [10]
Semi-tree-like	Linear [25]
Unrestricted	Exponential [25]

TABLE I
ADAPTATION OF RESOLUTION REFUTATIONS FOR MAX-SAT

Example 4. *We consider the regular tree resolution refutation in Example 1 represented by the DAG in Figure 1. The clause (x_1) is used two times as a premise of a resolution step. The junction point of the left and right branches leaving from (x_1) eliminates variable x_3. We apply the split rule on clause (x_1) to get $(x_1 \vee x_3)$ and $(x_1 \vee \overline{x_3})$ and we replace (x_1) by $(x_1 \vee x_3)$ and $(x_1 \vee \overline{x_3})$ respectively on the left and right branches. Finally, we replace all resolutions by Max-SAT resolutions to obtain the complete max-refutation in Figure 3.*

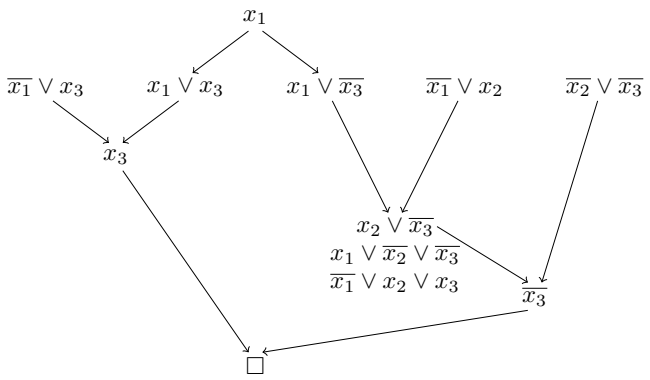

Fig. 3. Adapting a tree-like (regular) resolution refutation for Max-SAT [25]

As the unrestricted case is exponential, we propose in this paper an operational algorithm, called *substitute generation*, to compute a max-refutation of any unsatisfiable formula using an associated resolution refutation. We will see in Section V that there exists some exponential cases for the adaptation in [25] which are linear using the substitute generation algorithm.

III. The substitute generation algorithm

Given an unsatisfiable formula ϕ and an associated resolution refutation, we introduce in this section an algorithm, called *substitute generation*, able to compute a max-refutation of ϕ to transform it into a Max-SAT equivalent one containing an empty clause. To this aim, we follow the resolution steps of the resolution refutation and, for each resolution step, we check if the premises are in the current formula. If they are, we simply apply the Max-SAT resolution rule on these premises like in the read-once case [12]. If one premise is missing, we generate a substitute for it using a particular resolution refutation computed through a SAT oracle [3], [4]. Indeed, if we need to generate a missing clause c, it is possible to propagate $\overline{c}$ in the current formula and any resolution refutation of the obtained formula can be easily adapted to generate a clause subsuming c and consequently to generate c using the split rule. Hereafter, for a given unsatisfiable CNF formula ϕ and a resolution refutation $P = (r_1, r_2, ..., r_s)$ of ϕ, we denote $MS(P)$ the projection of P in Max-SAT, i.e. $MS(P) = (mr_1, mr_2, ..., mr_s)$ where mr_i is the application of the Max-SAT resolution rule on the premise clauses of r_i for $i \in \{1, .., s\}$. Clearly, $MS(P)$ is a valid max-refutation of ϕ only if P is read-once. However, we will prove in the following theorem that we are able to compute a max-refutation of ϕ containing every inference steps of $MS(P)$ in the same order.

Theorem 1. *Let ϕ be an unsatisfiable formula and P a resolution refutation of ϕ. There exists a max-refutation of ϕ containing every Max-SAT resolution step of $MS(P)$ in the same order.*

Proof. We set $P = (r_1, r_2, ..., r_s)$ and $MS(P) = (mr_1, mr_2, ..., mr_s)$. For each resolution step r_i ($i \in \{1, .., s\}$) applied on premises c_1 or c_2, we apply the Max-SAT resolution step mr_i if both clauses are in the current formula. When a clause c is not in the current formula, we generate a substitute for this clause using the following method. Let $\phi_{|\overline{c}}$ be the formula obtained from ϕ after the propagation of each literal in $\{\overline{l} \mid l \in c\}$. $\phi_{|\overline{c}}$ is unsatisfiable because ϕ is. As $\phi_{|\overline{c}}$ is unsatisfiable, there exists a sequence of resolution steps R (a resolution refutation) from $\phi_{|\overline{c}}$ to $\square$. If we replace each clause of this refutation by its version before propagation in ϕ, we obtain a sequence of resolution steps from ϕ_2 to a clause subsuming c. We prove by induction on the number of variables used as a pivot of a least one resolution step in R that there exists a sequence of Max-SAT equivalence-preserving transformations from ϕ to an equivalent containing c:

- If the number of variables used in the resolution refutation is 0, then R contains 0 resolution steps so there exists an empty clause $\square \in \phi_{|\overline{c}}$ and consequently there exists a clause $c_s \in \phi$ subsuming c. By the equivalence $c_s \equiv (c_s \vee \overline{l_1}) \wedge (c_s \vee l_1 \vee \overline{l_2}) \wedge ... \wedge (c_s \vee l_1 \vee ... \vee l_{k-1} \vee \overline{l_k}) \wedge c$, we obtain a Max-SAT equivalent of ϕ containing c with a finite sequence of split steps. If there is only one variable x in the resolution refutation, then R contains one resolution step on (x) and $(\overline{x})$ deducing the empty clause $\square$ and it is possible to generate c by applying one Max-SAT resolution step on the same clauses and then applying a finite sequence of split steps starting from $\square$ to get an equivalent containing c.
- Let $k > 1$, suppose that the property is true for each $k' < k$ and let R be a resolution refutation on k variables.
  - If R is read-once, we replace each resolution step by a Max-SAT resolution step and we obtain a finite transformation generating a clause c_s subsuming c. Then, we apply a finite sequence of split steps starting from c_s to get an equivalent containing c.
  - If R is not read-once, we apply the same method as in the read-once case but with one difference. As R is not read-once, it is possible that we have to apply a Max-SAT resolution step on a missing premise. By induction, we generate a substitute for this premise after propagating its opposite literals. Each recursion uses a new resolution refutation with at least one less variable and therefore each substitute generation is finite (the number of variables is strictly decreasing). We have at most $2|R|$ substitutes to generate so the complete transformation to get a clause $c_s \in \phi$ subsuming c is finite. Then, we apply a finite sequence of split steps to get c and we obtain a complete transformation generating c which is finite.

Consequently, given a premise c of a resolution step mr_i, we are able to generate a substitute for this clause without consuming the other premise. We are hence sure to iteratively apply every step of $MS(P)$ and the complete computed transformation is a finite max-refutation of ϕ containing each step of $MS(P)$ in the same order. ■

The procedure given in the proof of Theorem 1 can be

described under the form of an algorithm, which we refer to as *substitute generation* and which is described below (see Algorithm 1). It uses a sub-procedure, described in Algorithm 2, to generate a substitute for any missing clause. We use the following notations:

- substitute_generation_algorithm(ϕ, P) denotes the application of the substitute generation algorithm (Algorithm 1) on the formula ϕ and on the sequence of resolution steps P.
- generate_substitute(c, ϕ) launches the sub-procedure (Algorithm 2) which generates a substitute for clause c from the current formula ϕ.
- $T.T'$ denotes the concatenation of the max-refutations T and T'.
- apply_Max-SAT_resolution(c_1, c_2, ϕ) applies the Max-SAT resolution rule on clauses c_1 and c_2 which are in the formula ϕ and returns the transformed formula.
- deduce_by_split(c, c', ϕ) uses several split steps on clause c' in the current formula ϕ to deduce clause c and returns the transformed formula.
- propagate($\overline{c}$, ϕ) propagates the literals in $\{l \mid l \in c\}$ and returns the simplified formula.
- compute_resolution_refutation(ϕ) computes and returns a resolution refutation on the unsatisfiable formula ϕ using a SAT oracle
- cancel_propagation($\overline{c}$, ϕ, P) cancels the propagation of literals $\{l \mid l \in c\}$ in the current formula ϕ and modifies the resolution refutation (which becomes a sequence of resolution steps not necessarily deducing the empty clause).
- find_subsuming_clause(c, ϕ) returns a clause $c' \in \phi$ which subsumes c.

Algorithm 1 Substitute Generation Algorithm

Require: unsatisfiable CNF formula ϕ, resolution refutation P of ϕ
Ensure: (ϕ', T) where T is a max-refutation of ϕ and ϕ' the result of application of T on ϕ

$T \leftarrow \emptyset$
for all resolution step of P on clauses c_1 and c_2 **do**
  if $c_1 \notin \phi$ **then**
    $(\phi, T') \leftarrow$ generate_substitute(c_1, ϕ)
    $T \leftarrow T.T'$
  end if
  if $c_2 \notin \phi$ **then**
    $(\phi, T') \leftarrow$ generate_substitute(c_2, ϕ)
    $T \leftarrow T.T'$
  end if
  $(\phi, T') \leftarrow$ apply_Max-SAT_resolution(c_1, c_2, ϕ)
  $T \leftarrow T.T'$
end for
return (ϕ, T)

Algorithm 2 Sub-procedure: generate_substitute

Require: clause c, unsatisfiable CNF formula ϕ
Ensure: (ϕ', T) where T is a Max-SAT equivalence-preserving transformation from ϕ to ϕ' containing c

if $\exists c' \in \phi$, c' subsumes c **then**
  ϕ, $T \leftarrow$ deduce_by_split(c, c', ϕ)
  return (ϕ, T)
end if
$\phi \leftarrow$ propagate($\overline{c}$, ϕ)
$P \leftarrow$ compute_resolution_refutation(ϕ)
$(\phi, P) \leftarrow$ cancel_propagation($\overline{c}$, ϕ, P)
$(\phi, T) \leftarrow$ substitute_generation_algorithm(ϕ, P)
if $c \notin \phi$ **then**
  $c' \leftarrow$ find_subsuming_clause(c, ϕ)
  $\phi, T' \leftarrow$ deduce_by_split(c, c', ϕ)
  $T \leftarrow T.T'$
end if
return (ϕ, T)

In the next section, we apply the substitute generation algorithm on a non-read-once resolution refutation.

IV. Illustration Example

In this section, we use the substitute generation algorithm to transform the formula $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$ using the associated resolution refutation in Figure 1.

Notice that this resolution refutation is ambiguous. Indeed, resolution steps are partially ordered while we need a full order. To apply the substitute generation algorithm, we arbitrarily choose a full order presented in Table II. The application of the substitute generation algorithm is summarized in Table II and in Figure 6 and we provide further details hereafter.

A. First resolution step

The first resolution step is on clauses $(\overline{x_1} \vee x_3)$ and (x_1) which are in the current formula. We can therefore apply the Max-SAT resolution rule on these clauses and replace them by clauses (x_3) and $(x_1 \vee \overline{x_3})$. The current formula is now $\phi = (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (x_3) \wedge (x_1 \vee \overline{x_3})$.

B. Second resolution step

The second resolution step is on clauses (x_1) and $(\overline{x_1} \vee x_2)$.

1) Generation of substitute for (x_1): We must generate a substitute for (x_1) because it is not in the current formula. To this aim, we propagate $\overline{x_1}$, we compute a resolution refutation on the obtained formula and we cancel the propagation to get a sequence of resolution steps generating clause (x_1) (Figure 4). Then, we replace each resolution step by a Max-SAT resolution and we apply it to the current formula.

Remark 3. *We could have obtained a non-read-once resolution refutation in which case we would have to generate other substitutes for the current refutation.*

Remark 4. *We could have obtained a transformation deducing a clause subsuming the substitute in which case we would have to apply the split rule after the transformation to obtain the substitute.*

Step	Initial Resolution	Substitute Generation Algorithm	Current Formula
1	$(\overline{x_1} \vee x_3) \wedge (x_1) \rightarrow (x_3)$	$(\overline{x_1} \vee x_3) \wedge (x_1) \rightarrow (x_3) \wedge (x_1 \vee \overline{x_3})$	$(\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (x_3) \wedge (x_1 \vee \overline{x_3})$
2	$(x_1) \wedge (\overline{x_1} \vee x_2) \rightarrow (x_2)$	Substitute generation for clause (x_1) (Figure 4) $(x_3) \wedge (x_1 \vee \overline{x_3}) \rightarrow (x_1) \wedge (\overline{x_1} \vee x_3)$ Substitute for (x_1) has been generated $(x_1) \wedge (\overline{x_1} \vee x_2) \rightarrow (x_2) \wedge (x_1 \vee \overline{x_2})$	$(\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3) \vee (x_2) \wedge (x_1 \vee \overline{x_2})$
3	$(x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \rightarrow (\overline{x_3})$	$(x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \rightarrow (\overline{x_3}) \wedge (x_2 \vee x_3)$	$(\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_3}) \wedge (x_2 \vee x_3)$
4	$(x_3) \wedge (\overline{x_3}) \rightarrow \square$	Substitute generation for clause (x_3) (Figure 5) $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3) \rightarrow (\overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3})$ $\wedge(\overline{x_1} \vee x_2 \vee x_3)$ $(\overline{x_2} \vee x_3) \wedge (x_2 \vee x_3) \rightarrow (x_3)$ Substitute for (x_3) has been generated $(x_3) \wedge (\overline{x_3}) \rightarrow \square$	$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge \square$

TABLE II
EXECUTION OF THE SUBSTITUTE GENERATION ALGORITHM

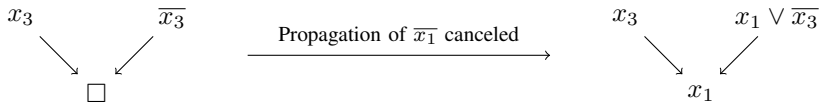

Fig. 4. Sequence of resolution steps to generate substitute for (x_1)

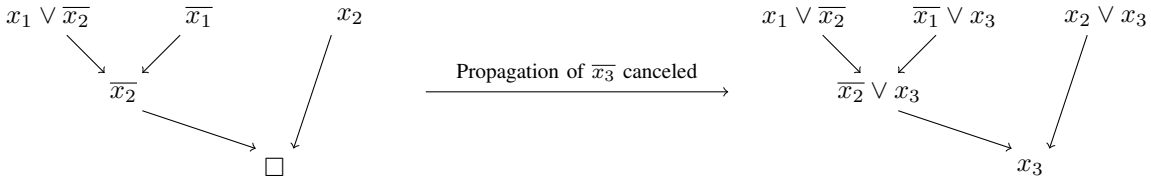

Fig. 5. Sequence of resolution steps to generate substitute for (x_3)

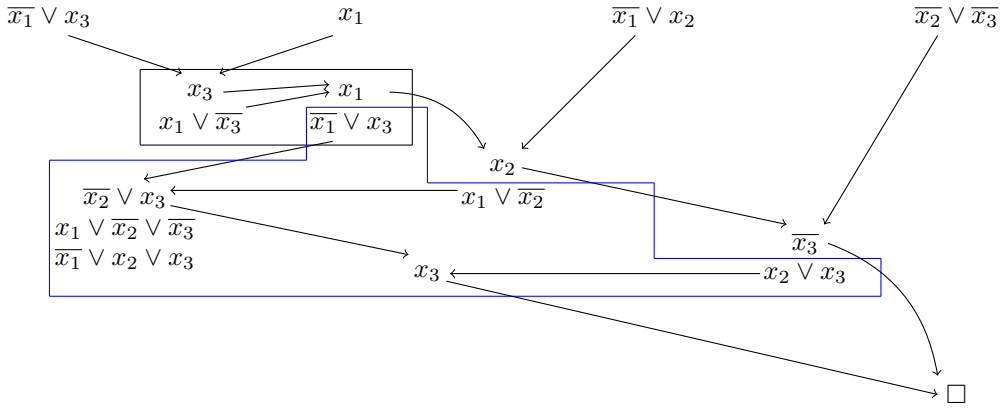

Fig. 6. Adaptation of a resolution refutation for Max-SAT using the substitute generation algorithm

2) First resolution step (second level): The first and unique resolution step to obtain the substitute is on clauses (x_3) and $(x_1 \vee \overline{x_3})$, which are in the current formula. We can therefore apply the Max-SAT resolution rule on these clauses and replace them by clauses (x_1) and $(\overline{x_1} \vee x_3)$. The substitute for clause (x_1) has now been generated and the current formula is now $\phi = (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (x_1) \wedge (\overline{x_1} \vee x_3)$.

3) Back-up at level 1 and end of the second resolution step: The two clauses (x_1) and $(\overline{x_1} \vee x_2)$ are now in the current formula. We can therefore apply the Max-SAT resolution on these clauses and replace them by clauses (x_2) and $(x_1 \vee \overline{x_2})$. The current formula is now $\phi = (\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3) \vee (x_2) \wedge (x_1 \vee \overline{x_2})$.

C. Third resolution step

The third resolution step is on clauses (x_2) and $(\overline{x_2} \vee \overline{x_3})$, which are in the current formula. We can therefore apply the Max-SAT resolution rule on these clauses and replace them by clauses $(\overline{x_3})$ and $(x_2 \vee x_3)$. The current formula is now $\phi = (\overline{x_1} \vee x_3) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_3}) \wedge (x_2 \vee x_3)$.

D. Fourth resolution step

The fourth resolution step is on clauses (x_3) and $(\overline{x_3})$.

1) Generation of substitute for (x_3): We must generate a substitute for (x_3) because it is not in the current formula. To this aim, we propagate $\overline{x_3}$, we compute a resolution refutation on the obtained formula and we cancel the propagation to get a sequence of resolution steps generating clause (x_3) (Figure 5). Then, we replace each resolution step by a Max-SAT resolution and we apply it to the current formula.

2) First resolution step (second level): The first resolution step to obtain the substitute is on clauses $(x_1 \vee \overline{x_2})$ and $(\overline{x_1} \vee x_3)$, which are in the current formula. We can therefore apply the Max-SAT resolution rule on these clauses and replace them by clauses $(\overline{x_2} \vee x_3)$, $(x_1 \vee \overline{x_2} \vee \overline{x_3})$ and $(\overline{x_1} \vee x_2 \vee x_3)$. The current formula is now $\phi = (\overline{x_3}) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$.

3) Second resolution step (second level): The second resolution step to obtain the substitute is on clauses $(\overline{x_2} \vee x_3)$ and $(x_2 \vee x_3)$, which are in the current formula. We can therefore apply the Max-SAT resolution rule on these clauses and replace them by the clause (x_3). The substitute for clause (x_3) has now been generated and the current formula is now $\phi = (\overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_3)$.

4) Back-up at level 1 and end of the fourth resolution step: The two clauses (x_3) and $(\overline{x_3})$ are now in the current formula. We can therefore apply Max-SAT resolution on these clauses and replace them by the empty clause $\square$. The current formula is now $\phi = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge \square$.

E. End of the execution and summary

The execution of the substitute generation algorithm is over. It allowed to use the resolution refutation in Figure 1 to transform the formula $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$ into a Max-SAT-equivalent formula $\phi' = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge \square$ containing an empty clause. The full transformation is summarized in Figure 6. The green area highlights the generation of a substitute for (x_1) while the blue one emphasises the generation of a substitute for (x_3).

V. Substitution and diamond patterns

In this section, we study a set of resolution refutations, called k-stacked diamond patterns, whose adaptation is exponential in [25] but linear using the substitute generation algorithm.

Definition 6 (Diamond pattern)**.** *Let A be a disjunction of literals and let $x \notin var(A)$ and $y \notin var(A)$ two distinct variables. We define the diamond pattern (x, y, A) as the sequence of resolutions represented in Figure 7.*

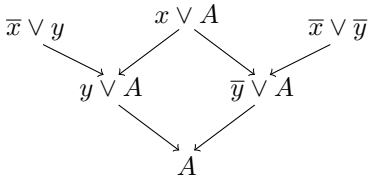

Fig. 7. Diamond pattern (x, y, A)

Fig. 8. Simplified representation of a diamond pattern

We can represent this pattern by a diamond as in Figure 8. Notice that in particular, the diamond pattern $(x, y, \square)$ is a resolution refutation. Now, imagine that the topmost clause of $(x, y, \square)$ is derived through another diamond pattern. We iterate the same reasoning to define a k-stacked diamonds pattern as follows:

Definition 7 (k-stacked diamond pattern)**.** *Let $k \geq 1$ be a natural number and let x_i and y_i where $1 \leq i \leq k$ be distinct variables. A k-stacked diamond pattern is formed by k diamond patterns (x_i, y_i, A_i) where $1 \leq i \leq k$ such that $A_1 = \square$ and $A_i = (x_1 \vee \cdots \vee x_{i-1})$ for $1 < i \leq k$. Each diamond (x_i, y_i, A_i) is stacked on top of $(x_{i-1}, y_{i-1}, A_{i-1})$ such that the last conclusion of the former is the topmost central premise of the latter.*

A k-stacked diamond pattern is represented as a stack of diamonds as shown in Figure 9 for $k = 3$. Clearly, k-stacked diamond are resolution refutations as they deduce

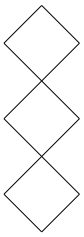

Fig. 9. Simplified representation of a 3-stacked diamond pattern

the empty clause $\square$. In particular, when $k > 2$, the size of a k-stacked diamond P is $|P| = 3k$ while the size of the computed max-refutation by the adaptation in [25] is at least 2^{k-1} which is exponential in the size of P. However, the substitute generation algorithm is able to compute a max-refutation of size $5k < 2|P|$, which is linear in the size of P. Indeed, it is able to adapt any diamond pattern for Max-SAT using at most 5 Max-SAT resolution steps, as we can see in Figure 10. Consequently, the substitute generation algorithm will iteratively adapt each diamond pattern using 5 Max-SAT resolution steps and the size of the full adaptation is $5k$ Max-SAT resolution steps and is therefore linear.

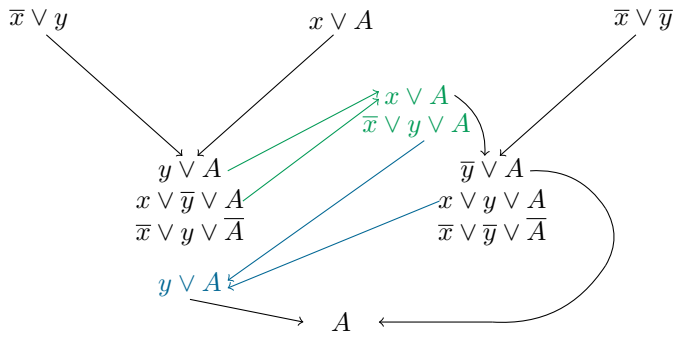

Fig. 10. Adaptation of the diamond pattern (x, y, A) using the substitute generation algorithm

VI. Conclusions and Future Work

In this paper, we presented an algorithm, called *substitute generation*, to compute a max-refutation of a given formula using an associated resolution refutation. This max-refutation allows to transform any inconsistent formula into a Max-SAT equivalent formula containing an empty clause. We also showed that k-stacked diamond patterns, whose adaptation is exponential in [25], can be linearly adapted using the substitute generation algorithm.

Future work include an empirical comparison between the substitute generation algorithm and the adaptation proposed in [25]. It could also be interesting to merge these approaches, by applying the existing adaptation on linear cases and the proposed algorithm in the exponential case. As the existing adaptation has been recently applied to generate certificates for the Max-SAT problem [26], the substitute generation algorithm can also be used for the same application. Finally, we can see both in the example of Section IV summarized in Figure 6 and in the diamond pattern in Figure 10 a kind of backtrack to generate a substitute, i.e. the algorithm may cancel the previous inference steps to generate a substitute. Consequently, it would be interesting to study a variation of the substitute generation algorithm which is not applied on the whole resolution refutation, but just on a branch in the refutation.

References

[1] Abramé, A., Habet, D.: Ahmaxsat: Description and Evaluation of a Branch and Bound Max-SAT Solver. Journal on Satisfiability, Boolean Modeling and Computation **9**, 89–128 (2015)

[2] Ben-sasson, E., Impagliazzo, R., Wigderson, A.: Near Optimal Separation Of Tree-Like And General Resolution. Combinatorica **24**, 585–603 (09 2004)

[3] Biere, A.: Booleforce, http://fmv.jku.at/booleforce/

[4] Biere, A.: TraceCheck, http://fmv.jku.at/tracecheck/

[5] Bonet, M., Levy, J., Manyà, F.: A complete calculus for max-SAT. In: SAT 2006. vol. 4121, pp. 240–251 (08 2006)

[6] Bonet, M.L., Levy, J.: Equivalence Between Systems Stronger Than Resolution. In: Theory and Applications of Satisfiability Testing – SAT 2020. Lecture Notes in Computer Science, vol. 12178 (2020)

[7] Buss, S., Hoffmann, J., Johannsen, J.: Resolution Trees with Lemmas: Resolution Refinements that Characterize DLL Algorithms with Clause Learning. Logical Methods in Computer Science **4** (11 2008)

[8] Cherif, M.S., Habet, D., Abramé, A.: Understanding the power of Max-SAT resolution through UP-resilience. Artificial Intelligence **289** (2020)

[9] Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5**(7), 394–397 (1962)

[10] Filmus, Y., Mahajan, M., Sood, G., Vinyals, M.: MaxSAT Resolution and Subcube Sums. In: Theory and Applications of Satisfiability Testing - SAT 2020 - Proceedings. vol. 12178, pp. 295–311. Springer (2020)

[11] Gelder, A.V.: Extracting (Easily) Checkable Proofs from a Satisfiability Solver that Employs both Preorder and Postorder Resolution. In: International Symposium on Artificial Intelligence and Mathematics (ISAIM) (2002)

[12] Heras, F., Marques-Silva, J.: Read-Once resolution for Unsatisfiability-Based Max-SAT Algorithms. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI) (2011)

[13] Hertel, A., Urquhart, A.: Algorithms and Complexity Results for Input and Unit Resolution. Journal of Satisfiability, Boolean Modeling and Computation **6** (05 2009)

[14] Iwama, K., Miyano, E.: Intractability of read-once resolution. In: Proceedings of the Tenth Annual Structure in Complexity Theory Conference (1995)

[15] Kügel, A.: Improved Exact Solver for the Weighted MAX-SAT Problem. In: POS-10. Pragmatics of SAT. EPiC Series in Computing, vol. 8, pp. 15–27 (2010)

[16] Larrosa, J., Heras, F.: Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 193–198 (01 2005)

[17] Larrosa, J., Rollon, E.: Towards a Better Understanding of (Partial Weighted) MaxSAT Proof Systems. In: Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Proceedings. vol. 12178, pp. 218–232 (2020)

[18] Li, C.M., Manyà, F., Soler, J.R.: A Clause Tableau Calculus for MaxSAT. In: International Joint Conference on Artificial Intelligence (IJCAI). p. 766–772 (2016)

[19] Li, C.M., Manyà, F., Planes, J.: New inference rules for Max-SAT. Journal of Artificial Intelligence Research (JAIR) **30**, 321–359 (09 2007)

[20] Loveland, D.: A linear format for resolution. Symposium on Automatic Demonstration pp. 147–162 (01 1970)

[21] Luisa Bonet, M., Levy, J., Manyà, F.: Resolution for Max-SAT. Artificial Intelligence Volume 171, Issues 8–9, June 2007, Pages 606-618 (2007)

[22] Marques Silva, J.P., Sakallah, K.A.: GRASP-A new search algorithm for satisfiability. In: Proceedings of International Conference on Computer Aided Design. pp. 220–227 (1996)

[23] Narodytska, N., Bacchus, F.: Maximum Satisfiability Using Core-Guided MaxSAT Resolution. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (2014)

[24] Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artificial Intelligence **175** (2011)

[25] Py, M., Cherif, M.S., Habet, D.: Towards Bridging the Gap Between SAT and Max-SAT Refutations. In: 32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI. pp. 137–144. IEEE (2020)

[26] Py, M., Cherif, M.S., Habet, D.: A Proof Builder for Max-SAT. In: Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing (SAT) (2021)

[27] Robinson, J.A.: A machine-oriented logic based on the resolution principle. In: Journal of the Association for Computing Machinery, vol. 12 (1965), pp. 23–41. (1965)

[28] Urquhart, A.: A Near-Optimal Separation of Regular and General Resolution. SIAM Journal of Computing **40**, 107–121 (01 2011)