Journal of Artificial Intelligence Research 75 (2022) 1373-1400 Submitted 03/2022; published 12/2022

Proofs and Certificates for Max-SAT

Matthieu Py MATTHIEU.PY@UNIV-AMU.FR
Université Clermont Auvergne, Mines Saint-Etienne,
CNRS, LIMOS, F-63000 Clermont-Ferrand, France

Mohamed Sami Cherif MOHAMED-SAMI.CHERIF@UNIV-AMU.FR
Djamal Habet DJAMAL.HABET@UNIV-AMU.FR
Aix-Marseille Université, Université de Toulon,
CNRS, LIS, Marseille, France

Abstract

Current Max-SAT solvers are able to efficiently compute the optimal value of an input instance but they do not provide any certificate of its validity. In this paper, we present a tool, called MS-Builder, which generates certificates for the Max-SAT problem in the particular form of a sequence of equivalence-preserving transformations. To generate a certificate, MS-Builder iteratively calls a SAT oracle to get a SAT resolution refutation which is handled and adapted into a sound refutation for Max-SAT. In particular, we prove that the size of the computed Max-SAT refutation is linear with respect to the size of the initial refutation if it is semi-read-once, tree-like regular, tree-like or semi-tree-like. Additionally, we propose an extendable tool, called MS-Checker, able to verify the validity of any Max-SAT certificate using Max-SAT inference rules. Both tools are evaluated on the unweighted and weighted benchmark instances of the 2020 Max-SAT Evaluation.

1. Introduction

Given a Boolean formula in Conjunctive Normal Form (CNF), the Maximum Satisfiability (Max-SAT) problem consists in determining the maximum number of clauses that it is possible to satisfy by an assignment of the variables, while the Satisfiability (SAT) problem simply ascertains whether there exists an assignment which satisfies all the clauses. Max-SAT is an optimization extension of the satisfiability problem and is a natural formalism enabling to model many real-world and crafted problems (Muise, Beck, & McIlraith, 2016; Zhang & Bacchus, 2012; Demirovic & Musliu, 2017; Manyà, Negrete, Roig, & Soler, 2020; Achá & Nieuwenhuis, 2014; Bofill, Garcia, Suy, & Villaret, 2015; Xu, Rutenbar, & Sakallah, 2003; Guerra & Lynce, 2012; D'Almeida & Grégoire, 2012), making it of major interest in theory as well as in practice. Different complete solving paradigms for Max-SAT have seen the day in recent years including Branch and Bound algorithms (Li, Manyà, & Planes, 2007; Küegel, 2012; Abramé & Habet, 2014; Li, Xu, Coll, Manyà, Habet, & He, 2022) and SAT-based algorithms (Fu & Malik, 2006; Manquinho, Silva, & Planes, 2009; Ansótegui, Bonet, & Levy, 2009; Davies & Bacchus, 2011; Ansótegui, Bonet, & Levy, 2013; Martins, Manquinho, & Lynce, 2014; Ignatiev, Morgado, & Marques-Silva, 2019) among others. For a detailed overview of SAT and Max-SAT theory and solving, the reader can refer to the *Handbook of Satisfiability* (Biere, Heule, van Maaren, & Walsh, 2021).

Inference plays an important role in the context of Max-SAT solving (Li et al., 2007; Narodytska & Bacchus, 2014; Abramé & Habet, 2014) and this has led to an increasing interest in studying proof systems for Max-SAT in the literature (Larrosa & Heras, 2005; Bonet, Levy, & Manyà, 2006, 2007; Larrosa & Rollon, 2020a, 2020b; Bonet & Levy, 2020; Filmus, Mahajan, Sood, & Vinyals, 2020; Cherif, Habet, & Py, 2022). In particular, Max-SAT resolution (Larrosa & Heras, 2005; Bonet et al., 2006, 2007) is one of the first known complete systems for Max-SAT and is a natural extension of the resolution rule (Robinson, 1965) used in the context of SAT. Max-SAT resolution proofs are more constrained than their SAT counterparts as the premise clauses are replaced by the conclusions when applying Max-SAT resolution. Consequently, switching from a resolution proof to a Max-SAT resolution proof is possible and well-known for the particular case of read-once resolution (Bonet et al., 2007; Heras & Marques-Silva, 2011), where clauses can be used at most once in the proof. The adaptation of any resolution proof to a Max-SAT resolution proof 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"* (Bonet et al., 2006). This open problem coupled with the variety of paradigms and techniques used in Max-SAT solving hinders certificate generation for Max-SAT, which remains an unexplored topic in practice.

In this paper, we propose two main contributions for Max-SAT. Firstly, we contribute to the open problem of adapting resolution refutations, i.e., proofs deducing an inconsistency in the form of an empty clause, for Max-SAT. To this end, we augment Max-SAT resolution with the split rule which allows to generate two clauses subsumed by the original clause. We prove that it is always possible to adapt a resolution refutation into a max-refutation, i.e., a refutation using Max-SAT inference rules, whose size is linear with respect to the initial refutation for the following cases: semi-read-once resolution, tree-like regular resolution, tree-like resolution and semi-tree-like resolution. Furthermore, we propose a complete adaptation for any (unrestricted) resolution refutation into a max-refutation, although with a worst-case exponential blow-up in the size of the proofs. Secondly, we propose an independent tool, called MS-Builder, able to build certificates for the Max-SAT problem. To build such certificates, MS-Builder iteratively calls a SAT oracle to get a resolution refutation, adapts it for Max-SAT and applies it on the current formula. Moreover, we implemented an associated tool, called MS-Checker to check the validity of the certificates. Both tools are experimentally evaluated on the unweighted and weighted benchmarks of the 2020 Max-SAT Evaluation (Bacchus, Järvisalo, & Martins, 2020). This paper extends previous work published in (Py, Cherif, & Habet, 2020) and (Py, Cherif, & Habet, 2021a). The theoretical background on the adaptation of any resolution refutation for Max-SAT is based on our work in (Py et al., 2020) with one additional case, called *semi-read-once*, which is formally described and proved. The implementation of the tools has been improved since the work in (Py et al., 2021a) and MS-Builder is now able to generate certificates for weighted (partial) formulas.

This paper is organized as follows. Section 2 includes some necessary definitions and notations. Section 3 describes our proposed adaptations from resolution refutations to valid max-refutations. Section 4 describes our tools MS-Builder and MS-Checker and includes an experimental evaluation of both tools on the unweighted and weighted benchmark of the 2020 Max-SAT Evaluation (Bacchus et al., 2020). Finally, we conclude and discuss future work in Section 5.

2. Preliminaries

2.1 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 set) of literals, i.e., $c = (l_1 \lor l_2 \lor ... \lor l_k)$. A unit clause is composed of only one literal. A formula in Conjunctive Normal Form (CNF) ϕ is a conjunction (or multiset) of clauses, i.e., $\phi = c_1 \land c_2 \land ... \land c_m$. An assignment $I : X \longrightarrow \{0, 1\}$ 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. $\bar{l} \in I$). 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., $\exists l \in c$, $\bar{l} \in c'$. 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 problem (SAT) 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. Solving the (plain) Maximum Satisfiability problem (Max-SAT) consists in determining the maximum number of clauses that can be satisfied by an assignment of a CNF formula ϕ. Equivalently, it consists in determining the minimum number of clauses that each assignment must falsify, i.e., $opt(\phi) = \min_I cost_I(\phi)$. In the weighted partial Max-SAT problem, a finite or infinite weight is associated to each clause, representing the penalty of falsifying it. Clauses with infinite weight are called hard clauses and must be satisfied while clauses with finite weight $w_c \in \mathbb{N}^*$ are called soft clauses.

Example 1. *We consider the CNF formula $\phi = (\overline{x_1} \lor x_3) \land (x_1) \land (\overline{x_1} \lor x_2) \land (\overline{x_2} \lor \overline{x_3})$. ϕ is not satisfiable and it is possible to falsify only one clause, for instance with the assignment $I = \{x_1, x_2, x_3\}$, and therefore the $opt(\phi) = 1$.*

2.2 SAT Resolution

To certify that a CNF formula is satisfiable, it is sufficient to 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. A well-known SAT refutation system is based on an inference rule for SAT called resolution (Robinson, 1965). The resolution rule, defined below, deduces a clause called resolvent which can be added to the formula. Note that this rule is sound for SAT as it maintains SAT equivalence and it is extensively used in the context of SAT solving and particularly the Conflict Driven Clause Learning (CDCL) framework (Silva & Sakallah, 1996).

Definition 1 (Resolution (Robinson, 1965))**.** *Given two clauses $c_1 = (x \lor A)$ and $c_2 = (\overline{x} \lor B)$, the resolution rule deduces a third additional clause as follows:*

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

Definition 2 (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 $I : var(\phi) \cup var(\phi') \rightarrow \{1, 0\}$, I is a model of ϕ if and only if I is a model of ϕ'.*

It is possible to prove that a formula is unsatisfiable using a resolution refutation, which is a sequence of resolutions leading to an empty clause. A resolution refutation is represented as a Directed Acyclic Graph (DAG) whose nodes are either clauses having zero or two incoming arcs, respectively if they are clauses of the initial formula or clauses deduced by a resolution step. We refer to the clauses from the original formula, having zero incoming arcs in the DAG representation, as leaf clauses while clauses deduced by resolution and having two incoming arcs are referred to as intermediate clauses. Many restricted classes of resolution refutations have been studied in the literature namely linear resolution (Loveland, 1970), unit resolution (Hertel & Urquhart, 2009), input resolution (Hertel & Urquhart, 2009), regular resolution (Urquhart, 2011), read-once resolution (Iwama & Miyano, 1995) and tree-like resolution refutations (Ben-Sasson, Impagliazzo, & Wigderson, 2004) among others. In particular, a resolution refutation is tree-like if every intermediate clause is used at most once in the proof. Similarly, a resolution refutation is read-once if each clause is used at most once in the proof. Clearly, read-once refutations are also tree-like since they form a restricted class of tree-like resolution refutations. Finally, a resolution refutation is regular if each branch, i.e., path from a leaf to the empty clause, contains at most one resolution per variable. An irregularity is a sequence of clauses (each clause must be deduced using the previous one as premise) such that the first clause and the last one contain a literal l but at least one of the intermediate clauses does not contain this literal l. In other words, an irregularity is a certificate that a resolution refutation is not regular.

Example 2. *We consider the CNF formula $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$. The resolution refutation of ϕ, represented in Figure 1, is tree-like (and) regular, but not read-once because of clause (x_1).*

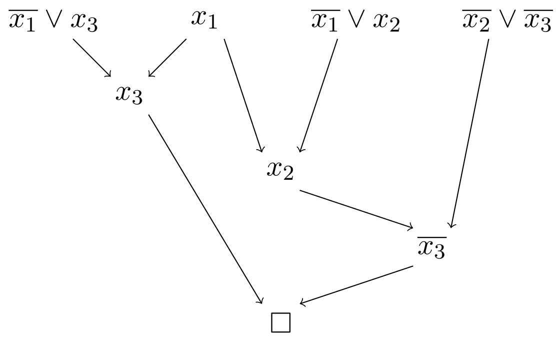

$\overline{x_1} \vee x_3$
x_1
$\overline{x_1} \vee x_2$
$\overline{x_2} \vee \overline{x_3}$
x_3
x_2
$\overline{x_3}$
$\square$

Figure 1: A resolution refutation

2.3 Max-SAT Resolution

In the last fifteen years, the study of inference rules for Max-SAT has led to major results in Max-SAT theory and solving. In particular, one of the first and most studied proof systems for Max-SAT is based on an inference rule called Max-SAT resolution (Larrosa &

Heras, 2005; Bonet et al., 2006, 2007), which is an extension of the resolution rule used in the context of SAT. The aim of complete Max-SAT systems is not to refute the formula per se but to compute the Max-SAT optimum of a given CNF formula, i.e., the maximum 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. Other than the resolvent clause, Max-SAT resolution introduces new clauses referred to as compensation clauses and which are essential to preserve Max-SAT equivalence.

Definition 3 (Max-SAT resolution (Larrosa & Heras, 2005; Bonet et al., 2006, 2007))**.** *Given two clauses $c_1 = x \vee A$ and $c_2 = \overline{x} \vee B$ where $A = a_1 \vee ... \vee a_s$ and $B = b_1 \vee ... \vee b_t$, the Max-SAT Resolution rule is defined as follows:*

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

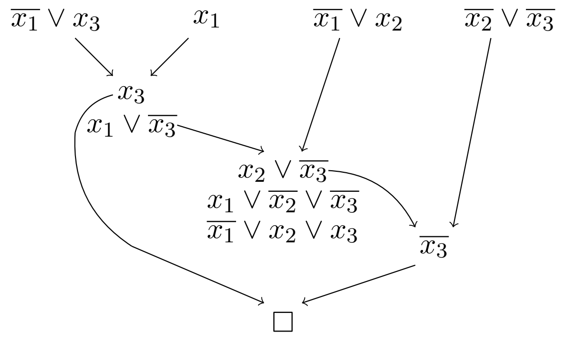
where c_r is called the resolvent clause and $cc_1, ..., cc_{t+s}$ the compensation clauses.

Definition 4 (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 $I : var(\phi) \cup var(\phi') \to \{1, 0\}$, we have $cost_I(\phi) = cost_I(\phi')$.*

Remark 1. *Unlike resolution, Max-SAT resolution replaces the premises by the conclusions. This is essential to maintain Max-SAT equivalence.*

As a sound and complete rule for Max-SAT (Bonet et al., 2006), Max-SAT resolution plays an important role in the context of Max-SAT theory and solving. In particular, it is extensively used in the context of Branch and Bound algorithms for Max-SAT to transform inconsistent subsets (Li et al., 2007; Küegel, 2012; Abramé & Habet, 2014; Cherif, Habet, & Abramé, 2020) and more marginally in the context of SAT-based algorithms to transform cores returned by SAT oracles (Heras & Marques-Silva, 2011; Narodytska & Bacchus, 2014). For a given CNF formula, it is always possible to generate a Max-SAT resolution proof of its optimum by applying the saturation algorithm (Bonet et al., 2006) to deduce empty clauses. A Max-SAT refutation, or simply max-refutation, is a sequence of Max-SAT inference steps (specifically Max-SAT resolution augmented with other rules defined hereafter) deducing the empty clause. A max-refutation can be represented as a bipartite DAG whose nodes are either clauses or inference steps. For more simplicity, we will omit inference step nodes and represent a max-refutation as a plain DAG (as showcased in Figure 2). We consider that the size of a SAT or Max-SAT refutation is the number of its inference steps.

Example 3. *We consider the CNF formula from Example 2. A hand-made max-refutation of ϕ was proposed in (Bonet et al., 2006) and is represented in Figure 2.*



$\overline{x_1} \vee x_3$
x_1
$\overline{x_1} \vee x_2$
$\overline{x_2} \vee \overline{x_3}$
x_3
$x_1 \vee \overline{x_3}$
$x_2 \vee \overline{x_3}$
$x_1 \vee \overline{x_2} \vee \overline{x_3}$
$\overline{x_1} \vee x_2 \vee x_3$
$\overline{x_3}$
$\square$

Figure 2: A max-refutation

In recent work, Max-SAT resolution was augmented with other rules such as the split rule (Larrosa & Rollon, 2020b; Bonet & Levy, 2020; Py, Cherif, & Habet, 2021b) defined below or the extension rule (Larrosa & Rollon, 2020a). It was shown that the addition of such rules to Max-SAT resolution can improve its efficiency in generating shorter proofs (Larrosa & Rollon, 2020b, 2020a; Py, Cherif, & Habet, 2021c) or in simulating other proof systems (Filmus et al., 2020; Bonet & Levy, 2020). To be exhaustive, we must also mention that other Max-SAT proof systems were introduced and studied in the literature (Li, Manyà, & Soler, 2016; Atserias & Lauria, 2019; Larrosa & Rollon, 2020a; Filmus et al., 2020).

Definition 5 (Split rule)**.** *Given a clause $c_1 = (A)$ where A is a disjunction of literals and x a variable, the split rule replaces the premise c_1 by two new clauses as follows:*

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

If these proof systems have been extensively studied in theory, generating proofs remains an unexplored topic in practice. Hence, this work aims to contribute to this topic by proposing tools to build and check certificates for the Max-SAT problem. To this aim, we first propose adaptations from resolution refutations to max-refutations. These adaptations are used in a tool enabling to build certificates for the Max-SAT problem. For the sake of simplicity, we will exhibit examples with unweighted unpartial (i.e., all clauses are soft) formulas to introduce MS-Builder. However, MS-Builder is also able to generate certificates for weighted (partial) Max-SAT formulas. To this end, two inference rules can be used to deal with weights, namely the fold and the unfold rules defined below. Using these rules, Max-SAT resolution and split can be easily lifted to the weighted case (Bonet et al., 2007; Larrosa, Heras, & de Givry, 2008; Larrosa & Rollon, 2020b).

Definition 6 (Fold & Unfold)**.** *Given a weighted clause c and two positive weights w_1 and w_2, the fold and unfold rules are respectively defined as follows:*

$$\frac{(c, w_1) \quad (c, w_2)}{(c, w_1 + w_2)} \qquad\qquad \frac{(c, w_1 + w_2)}{(c, w_1) \quad (c, w_2)}$$

3. From Resolution Refutations to Max-Refutations

In this section, we show how to adapt any resolution refutation to get a max-refutation, i.e., a refutation using Max-SAT-equivalence-preserving inference rules. In particular, we prove that the size of the computed max-refutation is linear with respect to the size of the initial resolution refutation in the case of semi-read-once, tree-like regular, tree-like or semi-tree-like resolution. We also prove that the adaptation is always possible in the unrestricted case, but with a worst-case exponential blow-up in the size of the proofs. The theoretical results are resumed in Table 1.

Resolution Refutation	**Size of the Max-SAT Refutation**
Read-Once	Linear (Bonet et al., 2007; Heras & Marques-Silva, 2011)
Semi-Read-once	Linear
Tree-like regular	Linear
Tree-like	Linear
Semi-tree-like	Linear
Unrestricted	Exponential

Table 1: Adaptation of resolution refutations for Max-SAT

3.1 From Semi-Read-Once Resolution Refutations to Max-Refutations

SAT algorithms are based on unit propagation, which means that when a unit clause is deduced, the value of its only literal is propagated in the whole formula, because satisfying this literal is necessary to satisfy the formula. Applying unit propagation can be seen as the use of a particular unit clause in several resolution steps. As such, transforming resolution refutations to fix non-read-once unit clauses can therefore be a useful preprocessing technique to our proof builder which relies on iterative calls to a SAT oracle, as will be shown in Section 4. To fix a non-read-once unit clause, we remove the resolution steps in which it is involved and we add a new resolution step at the end of the refutation. Such a strategy works when the refutation is *based on unit propagation*, i.e., every time a resolution step is applied on a unit clause, the variable contained in the unit clause no longer appears in the rest of the refutation. As SAT algorithms make a strong application of the unit propagation technique, we made the hypothesis, confirmed by experiments, that the computed resolution refutation will be often based on unit propagation. Accordingly, we propose in Lemma 1 to adapt any resolution refutation based on unit propagation to Max-SAT by fixing every non-read-once unit clause.

Lemma 1. *Let ϕ be an inconsistent formula and P a resolution refutation of ϕ based on unit propagation. There exists a resolution refutation of ϕ such that every unit clause is read once containing $O(|P|)$ inference steps.*

Proof. Let ϕ be an inconsistent formula and P a resolution refutation of ϕ based on unit propagation. We show that if P has $k > 0$ non-read-once unit clauses, then there exists a resolution refutation with $k-1$ non-read-once unit clauses. To this aim, we consider the last non-read-once unit clause c of P and assume w.l.o.g. that $c = (x)$. Since P is based on unit propagation, then we can delete resolution steps on clause c to generate $c' = (\overline{x})$ instead of

$\Box$. We can now add a resolution step on clauses $c = (x)$ and $c' = (\overline{x})$ to get a resolution refutation of ϕ with $k - 1$ non-read-once unit clauses. Repeating this transformation, we clearly obtain a resolution refutation of ϕ such that every unit clause is read-once containing $O(|P|)$ inference steps.

■

The proposed transformation can be seen as a preprocessing technique for any non-read-once resolution refutation. In particular, some non-read-once resolution refutations can be non-read-once only because of unit clauses and we say that such refutations are *semi-read-once*. These semi-read-once resolution refutations can be adapted for Max-SAT as described in Theorem 1.

Definition 7 (Semi-read-once)**.** *A resolution refutation is semi-read-once if it is based on unit propagation and if each non-read-once clause is also a unit clause.*

Theorem 1. *Given an unsatisfiable formula ϕ and a semi-read-once resolution refutation P of ϕ, there exists a max-refutation of ϕ containing $O(|P|)$ inference steps.*

Proof. We apply the transformation in the proof of Lemma 1 to get a read-once resolution refutation P' of size $O(|P|)$. P' can then be immediately adapted to a Max-SAT refutation of size $|P'|$ (Bonet et al., 2007) and therefore of size $O(|P|)$.

■

Example 4. *We consider the tree-like regular resolution refutation in Figure 1. This refutation is semi-read-once. To make it read-once, we remove the two resolution steps on clause (x_1) and we add a new resolution step at the end of the refutation. We obtain the read-once resolution refutation in Figure 3. We can now replace each resolution step by a Max-SAT resolution step to get a max-refutation, represented in Figure 4.*

Remark 2. *For simplicity of presentation of the examples in the following subsections, we intentionally avoid applying the unit-propagation-fixing preprocessing described in Lemma 1.*

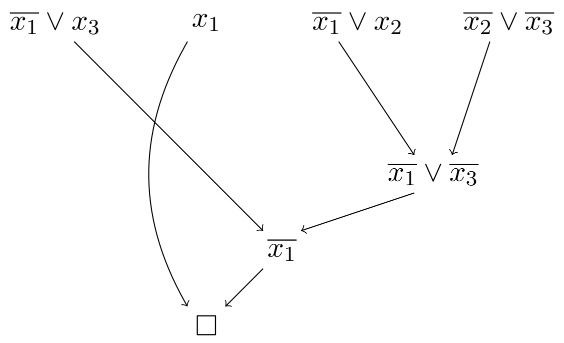

$\overline{x_1} \vee x_3$
x_1
$\overline{x_1} \vee x_2$
$\overline{x_2} \vee \overline{x_3}$
$\overline{x_1} \vee \overline{x_3}$
$\overline{x_1}$
$\Box$

Figure 3: Transformation of a *semi-read-once* resolution refutation into a *read-once* resolution refutation

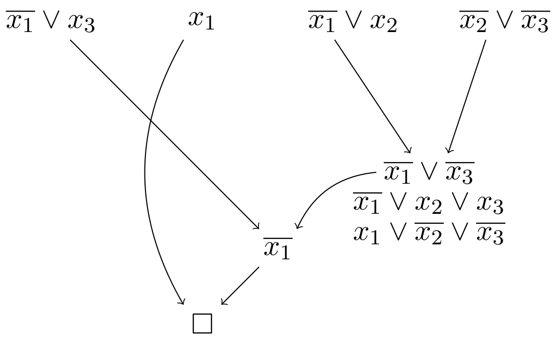

$\overline{x_1} \vee x_3$
x_1
$\overline{x_1} \vee x_2$
$\overline{x_2} \vee \overline{x_3}$
$\overline{x_1} \vee \overline{x_3}$
$\overline{x_1} \vee x_2 \vee x_3$
$x_1 \vee \overline{x_2} \vee \overline{x_3}$
$\overline{x_1}$
$\square$

Figure 4: Adaptation of a *semi-read-once* resolution refutation into a max-refutation

3.2 From Tree-Like Regular Resolution Refutations to Max-Refutations

To adapt a tree-like regular resolution refutation for Max-SAT, the idea is to use the split rule to fix non-read-once clauses. More precisely, if a clause c is used k times ($k > 1$) as a premise of a resolution step, we use the split rule on clause c with respect to a particular variable x which is carefully chosen as in Lemma 2, to duplicate c into two distinct clauses $c \vee x$ and $c \vee \overline{x}$. We then use $c \vee x$ and $c \vee \overline{x}$ to replace c as a premise of its resolution steps. If necessary, we repeat the same process on clauses $c \vee x$ and/or $c \vee \overline{x}$. Given a branch starting from a leaf clause c in a regular refutation, we say that this branch accepts the substitution of c by $c \vee l$ if the literal l appears in at least one resolvent in the branch. Clearly, such a substitution does not impact the validity of the refutation. The following lemma guarantees that, for a given non-read-once clause, there exists a variable x such that some branches starting from c accepts the substitution of c by $c \vee x$ while the rest accepts the substitution of c by $c \vee \overline{x}$.

Lemma 2. *Given a regular tree-like resolution refutation P and a non-read-once clause c in P, there exists a variable $x \notin var(c)$ such that it is possible to partition the branches starting from c into two non-empty subsets of branches, the branches in the first subset accepting the substitution of c by $c \vee x$ and the branches in the second accepting the substitution of c by $c \vee \overline{x}$.*

Proof. Let P a non-read-once regular tree-like resolution refutation and c a non-read-once clause in P. There exists a node v of the DAG of P representing a resolution step on variable x such that v is the first junction point of all the paths starting from c. The existence is ensured since this junction point is eventually the empty clause. Furthermore, every path starting from the clause c leads to one (and only one) of the premises of the resolution step in the node v. Indeed, a path leading to both premises entails the existence of an intermediate non-read-once clause which is not possible since the refutation is tree-like. We partition the branches starting from c into two subsets containing respectively the paths leading to the first and second premise of the resolution step in the node v. Each partition is non empty since if there exists an empty subset v can't be the first junction point of the branches. Let x be the variable eliminated at this resolution step and suppose w.l.o.g that the first premise contains literal x while the second contains literal $\overline{x}$. As P is regular, x is not a variable of c and the subset of branches starting from c leading to the first premise accepts the substitution of c by $c \vee x$ while the subset of branches leading to the second premise accepts the substitution of c by $c \vee \overline{x}$.

■

The result established in Lemma 2 ensures the possibility to fix any non-read-once clause used $k > 1$ times by using the split rule. Indeed, we can apply this rule to replace a non-read-once clause used $k > 1$ times by two clauses used respectively $1 \leq k_1 < k$ and $1 \leq k_2 < k$ such that $k = k_1 + k_2$. By iterating this method, we can fix every non-read-once clause. Then, we only need to replace the resolution rule by the Max-SAT resolution rule to obtain an adaptation from any regular tree-like resolution refutation to a max-refutation in linear size as established in the following theorem.

Theorem 2. *Given an unsatisfiable formula ϕ and a regular tree-like resolution refutation P of ϕ, there exists a max-refutation of ϕ containing $O(|P|)$ inference steps.*

Proof. Let P be a regular tree-like resolution refutation of ϕ. We set $T_1 = \emptyset$ and $T_2 = MR(P)$, where $MR(P)$ is obtained from P after replacing each resolution by Max-SAT resolution. If P is read-once, T_2 is a max-refutation of ϕ containing $|P|$ inference steps (which is obviously in $O(|P|)$). Now, let c be a non-read-once clause of P. Using Lemma 2, there exists a variable $x \notin var(c)$ and a partition of the branches starting from c into two non-empty subsets, the first accepting $c \vee x$ and the second accepting $c \vee \overline{x}$. We apply the Max-SAT split rule on c to obtain $c \vee x$ and $c \vee \overline{x}$ and we replace c as premise by $c \vee x$ on the first subset of branches and c by $c \vee \overline{x}$ on the second. Doing this, we augment T_1 by adding one split and we change T_2 by replacing the premise clause c as described above. As T_2 is a tree-like regular resolution refutation of $(\phi \backslash c) \wedge (c \vee x) \wedge (c \vee \overline{x})$, it is possible to iteratively apply this operation on T_2 until we obtain a read-once regular tree-like resolution refutation. Therefore, after the last iteration, we have a couple (T_1, T_2) such that T_1 is a sequence of applications of the split rule transforming ϕ into a Max-SAT equivalent ϕ' and T_2 is a read-once regular max-refutation of ϕ'. Therefore, these transformations form a max-refutation of ϕ.

To prove that the size of the max-refutation is in $O(|P|)$, we first consider how to fix a leaf clause of P (i.e., how to replace it by read-once clauses). We prove by induction on k that it is possible to fix any leaf clause of P used k times using at most $k - 1$ splits:

- If $k = 1$, we clearly need 0 splits to fix the read-once clause c.
- Suppose the assertion is true for any $k' < k$ and let c be a clause used k times. Using Lemma 2, it is possible to use 1 split to replace c by two clauses c_1 and c_2 respectively used k_1 and k_2 times with $k_1, k_2 > 0$ and $k_1 + k_2 = k$. Using our assertion for k_1 and k_2, it is possible to fix c_1 with at most $k_1 - 1$ splits and c_2 with at most $k_2 - 1$ splits. Therefore, it is possible to fix c with at most $1 + (k_1 - 1) + (k_2 - 1) = k - 1$ splits.

Let $c_1, ..., c_p$ be the leaf clauses of P used respectively $k^1, k^2, ..., k^p$ times. Notice that $k^1 + k^2 + ... + k^p = |P| + 1$ since P has exactly $2|P|$ premises, i.e., uses of clauses, and $|P| - 1$ intermediate clauses (the empty clause is not used and we neglect the trivial cases where a non-empty intermediate clause is not used and where the proof produces several empty clauses). Using the previous induction, we need at most $k^1 - 1 + k^2 - 1 + ... + k^p - 1 \leq |P|$

splits to fix every non-read-once leaf clause of P. Consequently, $|T_1| \leq |P|$. On the other hand, the number of Max-SAT resolutions in T_2 is by construction equal to the number of resolution steps in P and, therefore, $|T_2| = |P|$. We conclude that the complete max-refutation contains at most $2|P|$ inference steps, which is in $O(|P|)$.

■

Example 5. *We consider the regular tree-like resolution refutation from Example 2 represented by the DAG in Figure 1. We observe that the original clause* (x_1) *is used two times as a premise of a resolution step. The junction point of the left and right branches eliminates variable* x_3 *such that the branch on the left leads to the premise containing literal* x_3 *and the branch on right leads to the premise containing literal* $\overline{x_3}$. *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 full max-refutation represented in Figure 5.*

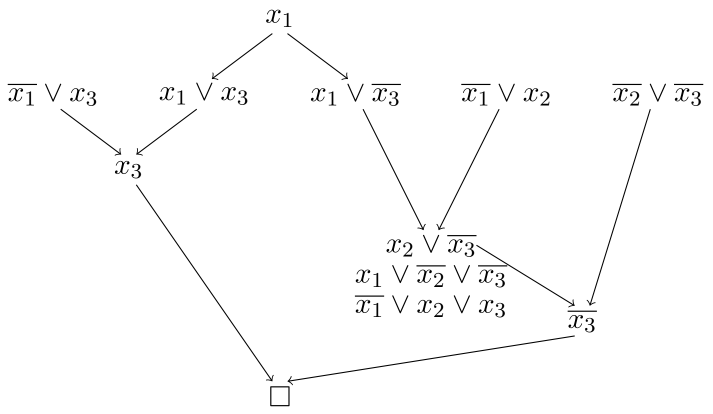

x_1
$\overline{x_1} \vee x_3$
$x_1 \vee x_3$
$x_1 \vee \overline{x_3}$
$\overline{x_1} \vee x_2$
$\overline{x_2} \vee \overline{x_3}$
x_3
$x_2 \vee \overline{x_3}$
$x_1 \vee \overline{x_2} \vee \overline{x_3}$
$\overline{x_1} \vee x_2 \vee x_3$
$\overline{x_3}$
$\square$

Figure 5: Applying the split rule to deal with a non-read once clause

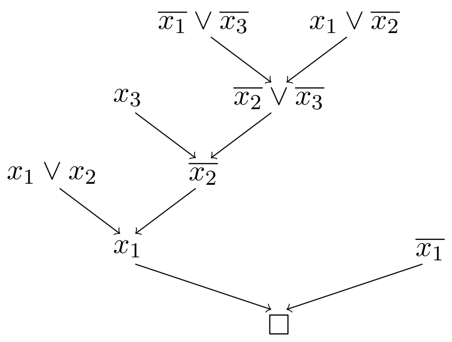
3.3 From Tree-Like Resolution Refutations to Max-Refutations

In the previous section, we proposed a linear adaptation from regular tree-like resolution refutations to max-refutations. Next, we extend the case where this adaptation guarantees linear size of the obtained max-refutation to tree-like resolution refutations. To this end, we simply exhibit a known transformation from any tree-like resolution refutation to a regular tree-like resolution refutation without increasing its size. This result was proved in (Urquhart, 1995) in the form of the following lemma (cf. Lemma 5.1 in (Urquhart, 1995)). The proof relies on a simple pruning argument which consists in iteratively discarding the first resolution in the case of an irregularity and updating the rest of the resolution proof accordingly, potentially discarding other resolution steps which are no longer necessary. The following lemma thus ensure that every tree-like resolution refutation can be made regular without increasing its size.

Lemma 3. *(Urquhart, 1995) A tree-like resolution refutation of minimal size is regular.*

Example 6. *We consider the tree-like resolution refutation represented in Figure 6. This refutation is not regular since* x_1 *is eliminated two times in the same branch. As shown in*

Figure 7, this refutation can be made regular by discarding the first resolution on variable x_1 in the irregularity and updating the rest of the proof. Note that after the transformation, clauses $(\overline{x_1} \vee \overline{x_3})$ and (x_3) are no longer used in the refutation.



$\overline{x_1} \vee \overline{x_3}$
$x_1 \vee \overline{x_2}$
x_3
$\overline{x_2} \vee \overline{x_3}$
$x_1 \vee x_2$
$\overline{x_2}$
x_1
$\overline{x_1}$
$\square$

Figure 6: Tree-Like resolution refutation containing an irregularity on variable x_1

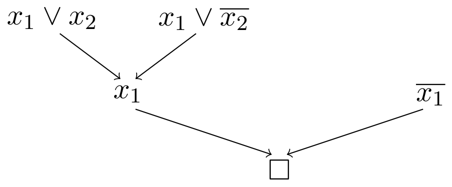


Figure 7: Regular tree-like resolution refutation after minimization

Since it is possible to make a tree-like resolution refutation regular without increasing the size of the proof, we can apply the adaptation in Theorem 2 to produce a max-refutation with linear size as established in the following theorem.

Theorem 3. *Given an unsatisfiable formula ϕ and a tree-like resolution refutation P of ϕ, there exists a max-refutation of ϕ containing $O(|P|)$ inference steps.*

Proof. Using Lemma 3, there exists a tree-like regular resolution refutation P_2 such that $|P_2| = O(|P|)$. By applying Theorem 2, we obtain a max-refutation containing $O(|P_2|)$ inference steps. Using the fact that $|P_2| = O(|P|)$, we conclude that this max-refutation contains $O(|P|)$ inference steps.

■

3.4 From Semi-Tree-Like Resolution Refutations to Max-Refutations

In Theorem 3, we proposed a linear adaptation from any tree-like resolution refutation to a max-refutation. In this section, we extend this linear result to semi-tree-like resolution refutations defined below. As shown in Proposition 1, this class of refutations extends tree-like resolution refutations, i.e., every tree-like resolution refutation is semi-tree-like.

Definition 8 (semi-tree-like resolution refutation)**.** *A resolution refutation is semi-tree-like if, for any branch of the refutation, at most one clause is non-read-once.*

Example 7. *We consider the resolution refutation P in Figure 8. P is clearly semi-tree-like since in each branch at most one clause is non-read-once. Notice also that P is not tree-like since (x_1) is an intermediate non-read-once clause.*

$$\begin{tikzcd}
 & x_1 \vee x_4 \arrow[dr] & & x_1 \vee \overline{x_4} \arrow[dl] & & \\
\overline{x_1} \vee x_3 \arrow[dr] & & x_1 \arrow[dl] \arrow[ddr] & \overline{x_1} \vee x_2 \arrow[dd] & & \overline{x_2} \vee \overline{x_3} \arrow[dddl] \\
 & x_3 \arrow[dddrr] & & & & \\
 & & & x_2 \arrow[dr] & & \\
 & & & & \overline{x_3} \arrow[dl] & \\
 & & & \square & &
\end{tikzcd}$$

Figure 8: Semi-tree-like resolution refutation

Proposition 1. *Let P be a resolution refutation. If P is tree-like then P is semi-tree-like.*

Proof. By definition of tree-like, each intermediate clause is read-once. In each branch, the only clause that can be non-read-once in P is by definition a leaf. Therefore, at most one clause is non-read-once in each branch and we conclude that P is semi-tree-like.

$\blacksquare$

To extend our result to semi-tree-like resolution refutations, we propose a method which relies on the fact that such refutations can be partitioned into two parts where the first part is a read-once sequence of resolutions and the second part is a tree-like resolution refutation. As the first part is a read-once sequence of resolutions, it is possible to adapt it for Max-SAT using a similar method to the one in (Heras & Marques-Silva, 2011), i.e., replacing each resolution by a Max-SAT resolution. Since the second part forms a tree-like resolution refutation, it is possible to adapt it for Max-SAT using our result in Theorem 3. After transforming the two parts, we glue them back to construct the full max-refutation.

Theorem 4. *Given an unsatisfiable formula ϕ and a semi-tree-like resolution refutation P of ϕ, there exists a max-refutation of ϕ containing $O(|P|)$ inference steps.*

Proof. As P is semi-tree-like, each branch of P contains at most one non-read-once clause. We partition P into two parts P_1 and P_2 as follows:

- For each branch containing one non-read-once clause, the transformations until this clause are put in P_1 and the transformations after this clause are put in P_2.
- For each branch not containing a non-read-once clause, the transformations are put in P_2.

By construction, P_1 is a read-once sequence of resolutions so it is possible to adapt it to obtain a Max-SAT transformation P_1' containing exactly $|P_1|$ inference steps by replacing resolutions with Max-SAT resolutions as in (Heras & Marques-Silva, 2011). Furthermore, P_2 is a tree-like resolution refutation because the non-read-once clauses of P are leaf clauses in P_2. Consequently, it is possible to adapt P_2 into a max-refutation P_2' containing $O(|P_2|)$

inference steps using the result in Theorem 3. Finally, we can combine P'_1 and P'_2 to obtain a Max-SAT refutation containing at most $O(|P_1|+|P_2|)$ and we conclude that the complete adaptation contains $O(|P|)$ inference steps.

■

Example 8. *We consider the semi-tree-like resolution refutation in Example 7, represented in Figure 8. To adapt this semi-tree-like resolution refutation to a max-refutation, we put aside the top resolution on variable x_1 taking clauses $(x_1 \vee x_4)$ and $(x_1 \vee \overline{x_4})$ and we obtain the tree-like resolution refutation in Example 2, represented in Figure 1. We adapt this tree-like resolution refutation as in Example 5 and we replace the resolution step on x_1 by a Max-SAT resolution step (in this case no compensation clauses are generated). We glue back the two parts to obtain the complete max-refutation represented in Figure 9.*

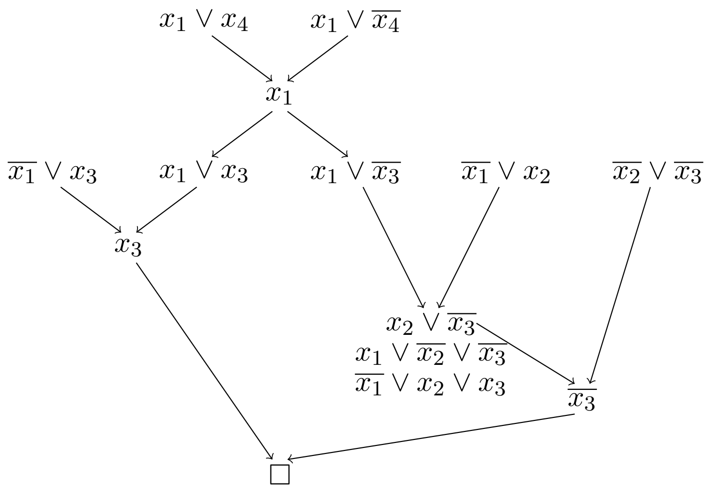

$x_1 \vee x_4$
$x_1 \vee \overline{x_4}$
x_1
$\overline{x_1} \vee x_3$
$x_1 \vee x_3$
$x_1 \vee \overline{x_3}$
$\overline{x_1} \vee x_2$
$\overline{x_2} \vee \overline{x_3}$
x_3
$x_2 \vee \overline{x_3}$
$x_1 \vee \overline{x_2} \vee \overline{x_3}$
$\overline{x_1} \vee x_2 \vee x_3$
$\overline{x_3}$
$\square$

Figure 9: Adapting a semi-tree-like resolution refutation to a max-refutation

3.5 From Unrestricted Resolution Refutations to Max-Refutations

In Theorems 1, 2, 3 and 4, we proposed linear adaptations from specific classes of resolution refutations to max-refutations. Now, we propose an adaptation from any resolution refutation to a max-refutation. The adaptation simply extends the one described in Theorem 3 by adding a prior transformation to make the initial resolution refutation tree-like as described in Lemma 4. Notice that we could make the initial resolution refutation semi-tree-like (instead of tree-like) but this choice does not affect the theoretical size of the obtained max-refutation.

To achieve this prior transformation, we iteratively search the proof for the first non-read-once intermediate clause c. If this clause is used $k > 1$ times as a premise of another resolution step, we consider the part of the proof leading to c and we duplicate it $k-1$ times in order to get a tree-like sequence of resolutions generating k resolvents each containing exactly the same literals as c and generated by a similar sequence of resolution steps. Consequently, c is no longer used several times as a premise of a resolution step, the input clauses are. Repeating this treatment on intermediary non-read-once clauses forces the resolution refutation to become tree-like. Fixing a non-read-once intermediate clause can, in the worst case, double the size of the current resolution refutation. As such, the size of the obtained tree-like resolution refutation is exponentially bounded by $O(|P| \times 2^{|P|})$, where P is the initial unrestricted resolution refutation. To polish this upper bound, we introduce a new parameter defined below, which is the number of multi-uses of intermediate clauses. Notice how, in the definition, we subtract one use for each clause. Intuitively, we consider the first use of any non-read-once intermediate clause as authorized.

Definition 9. *Let P be a resolution refutation. The number of multi-uses of intermediate non-read-once clauses, denoted $\mu(P)$, is defined as follows:*

$$\mu(P) = \sum_{c \text{ intermediate non-read-once in } P} (d^+(c) - 1)$$

where $d^+(c)$ denotes the number of uses of the clause c, i.e., the number of outgoing arcs from c in the DAG representation of P.

Lemma 4. *Given an unsatisfiable formula ϕ and a resolution refutation P of ϕ, there exists a tree-like resolution refutation of ϕ containing $O(2^{\mu(P)} \times |P|)$ resolution steps.*

Proof. Let P be a resolution refutation of ϕ. We iteratively make the intermediate non-read-once clauses read-once. Each time, we pick the first intermediate non-read-once clause c and duplicate the sub-proof deriving c. Each iteration decrements the number of intermediate non-read-once clauses by 1 until the resolution refutation becomes tree-like. Clearly, for each duplication, the size of the proof is doubled in the worst case and we perform exactly $\mu(P)$ duplications. We conclude that the size of the obtained tree-like resolution refutation is bounded by $O(2^{\mu(P)} \times |P|)$.

■

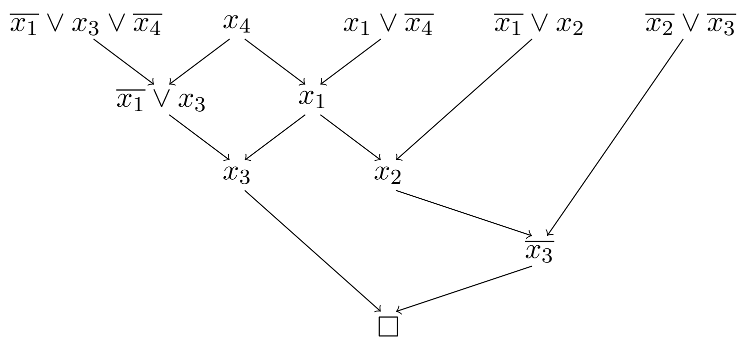
Now that we can transform any resolution refutation to a tree-like resolution refutation, we just have to adapt the obtained tree-like resolution refutation with the method described in Theorem 3 as established in the following theorem.

Theorem 5. *Given an unsatisfiable formula ϕ and an unrestricted resolution refutation P of ϕ, there exists a max-refutation of ϕ with $O(2^{\mu(P)} \times |P|)$ inference steps.*

Proof. Let P be an unrestricted resolution refutation of ϕ. We adapt P to obtain a tree-like resolution refutation P_t of size $O(2^{\mu(P)} \times |P|)$ using Lemma 4. Then, using Theorem 3, we obtain a max-refutation of size $O(2^{\mu(P)} \times |P|)$.

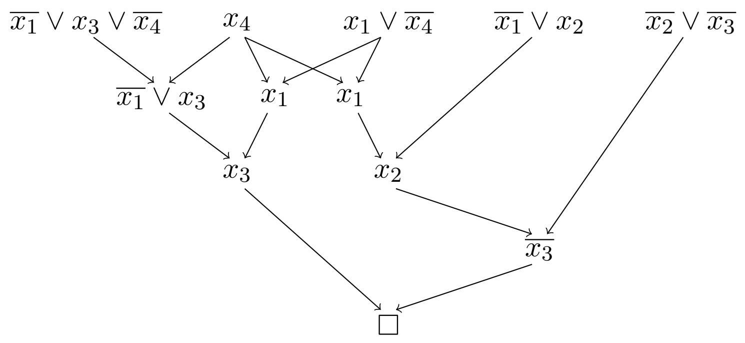
■

Example 9. *We consider the resolution refutation represented in Figure 10. This refutation is not semi-tree-like since the clauses (x_1) and (x_4) are two non read-once clauses in the same branch. First, we duplicate the resolutions leading to (x_1) and we obtain the tree-like resolution refutation represented in Figure 11. Then, we apply the transformations described in Theorem 3 to get the max-refutation represented in Figure 12.*



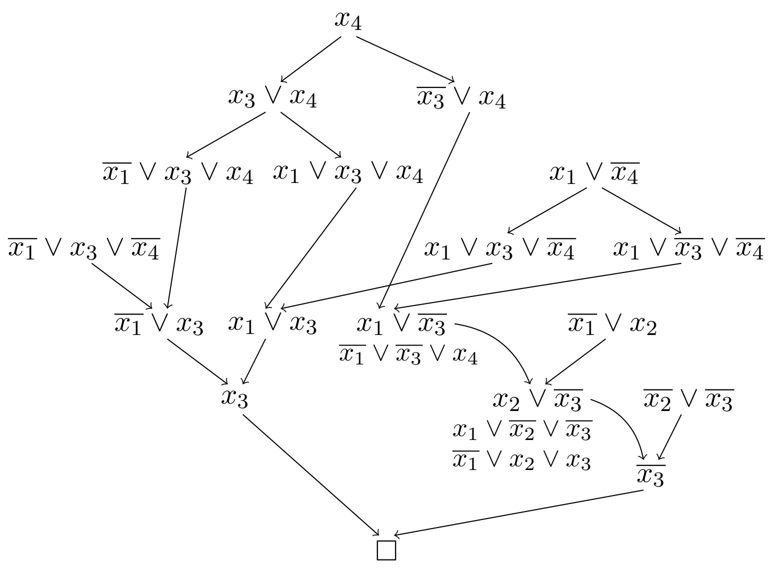
$\overline{x_1} \vee x_3 \vee \overline{x_4}$
x_4
$x_1 \vee \overline{x_4}$
$\overline{x_1} \vee x_2$
$\overline{x_2} \vee \overline{x_3}$
$\overline{x_1} \vee x_3$
x_1
x_3
x_2
$\overline{x_3}$
$\square$

Figure 10: Unrestricted resolution refutation



$\overline{x_1} \vee x_3 \vee \overline{x_4}$
x_4
$x_1 \vee \overline{x_4}$
$\overline{x_1} \vee x_2$
$\overline{x_2} \vee \overline{x_3}$
$\overline{x_1} \vee x_3$
x_1
x_1
x_3
x_2
$\overline{x_3}$
$\square$

Figure 11: Adapting a resolution refutation to a tree-like resolution refutation



x_4
$x_3 \vee x_4$
$\overline{x_3} \vee x_4$
$\overline{x_1} \vee x_3 \vee x_4$
$x_1 \vee x_3 \vee x_4$
$x_1 \vee \overline{x_4}$
$\overline{x_1} \vee x_3 \vee \overline{x_4}$
$x_1 \vee x_3 \vee \overline{x_4}$
$x_1 \vee \overline{x_3} \vee \overline{x_4}$
$\overline{x_1} \vee x_3$
$x_1 \vee x_3$
$x_1 \vee \overline{x_3}$
$\overline{x_1} \vee \overline{x_3} \vee x_4$
$\overline{x_1} \vee x_2$
x_3
$x_2 \vee \overline{x_3}$
$x_1 \vee \overline{x_2} \vee \overline{x_3}$
$\overline{x_1} \vee x_2 \vee x_3$
$\overline{x_2} \vee \overline{x_3}$
$\overline{x_3}$
$\square$

Figure 12: Adapting an unrestricted resolution refutation to a max-refutation

3.6 Unrestricted Resolution Refutations and Diamond Patterns

We finish this section by exhibiting resolution refutations whose adaptations as in Theorem 5 is exponential. To this end, we introduce in the following definition a new pattern which we will use to build such refutations.

Definition 10 (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 13.*

$$\begin{tikzcd}
\overline{x} \lor y \arrow[dr] & & x \lor A \arrow[dl] \arrow[dr] & & \overline{x} \lor \overline{y} \arrow[dl] \\
& y \lor A \arrow[dr] & & \overline{y} \lor A \arrow[dl] & \\
& & A & &
\end{tikzcd}$$

Figure 13: Diamond pattern (x, y, A)

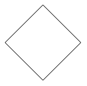


Figure 14: Simplified representation of a diamond pattern

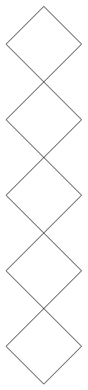


Figure 15: Simplified representation of a 5-stacked diamond pattern

We can represent this pattern by a diamond as in Figure 14. Notice that the particular 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 11 (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 $A_i = (x_1 \lor \cdots \lor x_{i-1})$ 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 15 for $k = 5$. Clearly, k-stacked diamonds are resolution refutations as they deduce the empty clause $\square$. In particular, when $k > 2$, a k-stacked diamond is not semi-tree-like. The size of a k-stacked diamond P is $|P| = 3k$. Furthermore, we have $\mu(P) = k - 1$. Therefore, after the application of the adaptation described in Theorem 4, we obtain a max-refutation whose size is at least 2^{k-1} showing that the proposed adaptation can be exponential in the worst case.

4. MS-Builder & MS-Checker

In this section, we propose to use the adaptation of any resolution refutation for Max-SAT, presented in Section 3, to generate certificates for the Max-SAT problem.

4.1 MS-Builder

MS-Builder generates certificates for the Max-SAT Problem in the particular form of a Max-SAT-equivalence-preserving transformation from the initial formula into a formula composed of a set of empty clauses and a satisfiable sub-formula. The treatment applied by MS-Builder is similar to SAT-based Max-SAT algorithms and particularly core-guided ones (Ansótegui et al., 2013; Morgado, Heras, Liffiton, Planes, & Marques-Silva, 2013). The tool also relies on the adaptation of resolution refutations proposed in Section 3. Given an initial formula, MS-Builder iteratively calls a SAT oracle to get a resolution refutation, adapts it for Max-SAT and applies it to the current formula as shown in Algorithm 1. When the SAT oracle returns that the current formula is now satisfiable (with a model), the algorithm terminates. The complete sequence of transformations generating k empty clauses is a proof that the Max-SAT optimum is at least k while the model is a proof that it is possible to falsify exactly k clauses and therefore that the Max-SAT optimum is k.

Algorithm 1 MS-Builder

```
Require: Formula $\phi$
Ensure: Max-SAT certificate $c$ for $\phi$
 1: $(T, opt) \leftarrow (\emptyset, 0)$
 2: while $\phi$ is inconsistent do
 3:     $RP \leftarrow$ compute_resolution_refutation($\phi$)
 4:     $MRP \leftarrow$ adapt_resolution_refutation_for_Max-SAT($RP$)
 5:     $\phi \leftarrow$ apply_max-refutation($\phi, MRP$)
 6:     $(\phi, opt) \leftarrow$ remove_empty_clauses($\phi, opt$)
 7:     $T \leftarrow$ concatenate($T, MRP$)
 8: end while
 9: $I \leftarrow$ compute_model($\phi$)
10: return $(T, opt, I)$
```

MS-Builder also works on weighted (partial) formulas in similar fashion. The only difference is that the inconsistent core returned by the oracle is treated differently. The minimum weight over all the clauses in the core is computed and is used to partition the clauses using the unfold rule. Once all the clauses in the core have the same weight, the obtained residual clauses are put back into the formula. Note that non-read-once hard clauses are not fixed because a hard clause can be used several times by duplicating as many copies as needed using the unfold rule. After the transformation, an empty clause is generated and its weight is added to *opt*. Note that this treatment is similar to the one performed by weighted core-guided algorithms for Max-SAT (Ansótegui et al., 2013; Morgado et al., 2013).

MS-Builder receives a file containing a formula in the standard WCNF format[1] and it returns a certificate like in Figure 17. The proof file must start with a sequence of Max-SAT transformation steps, each step being a line of the file. A Max-SAT transformation line must start with 't' and must include the name of the inference rule (`msres` for Max-SAT resolution, `split` for the split rule, `fold` for the fold rule and `unfold` for the unfold rule) and its premise(s) (between '< >' and separated by '|'). For the split rule, the variable to split on must be specified as a parameter after its name, while, for the unfold rule, the weight of the first conclusion must be specified in the same way. Then, the proof file must contain a line (starting with 'o') with the announced optimum cost of the formula. Finally, it must contain a line (starting with 'v') with a truth assignment satisfying the final formula (without the empty clauses). Both files may contains commentary lines, which must be started with 'c'.

Example 10. *We consider the formula $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2}) \wedge (\overline{x_3}) \wedge (x_2 \vee x_3)$ corresponding to the input file represented in Figure 16. MS-Builder calls a SAT oracle for the first time and it returns that the initial formula is inconsistent alongside the resolution refutation represented in Figure 18. This refutation is read-once and is adapted into the max-refutation in Figure 19 by simply replacing each resolution step with a Max-SAT resolution. After applying the max-refutation in Figure 19, the current formula is now $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge \Box$. MS-Builder calls a SAT oracle for the second time and it returns that the current formula (without the empty clause) is inconsistent. The oracle also provides the semi-read-once resolution refutation represented in Figure 20, which is adapted into the max-refutation in Figure 21. Applying the max-refutation of Figure 21 produces the current formula $\phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \Box \wedge \Box$. MS-Builder calls a SAT oracle for the third and last time. The SAT oracle returns that the current formula (without the empty clauses) is satisfiable and provides the model $I = \{\overline{x_1}, \overline{x_2}, \overline{x_3}\}$. Finally, MS-Builder returns the Max-SAT certificate described in Figure 17.*

```
c formula with opt = 2
1 -1 3 0
1 1 0
1 -1 2 0
1 -2 -3 0
1 -2 0
1 -3 0
1 2 3 0
```

Figure 16: Formula file

```
c certificate of example 10
c first refutation is read-once
t msres < 1 2 3 | 1 -2 >
t msres < 1 3 | 1 -3 >
c second refutation is semi-read-once
t msres < 1 -1 2 | 1 -2 -3 >
t msres < 1 -1 3 | 1 -1 -3 >
t msres < 1 1 | 1 -1 >
o 2
v 000
```

Figure 17: Max-SAT certificate

1. our tools support both the old and new formats described in `https://maxsat-evaluations.github.io/2020/format.html`

$$\begin{tikzcd}
x_2 \lor x_3 \arrow[dr] & & \overline{x_2} \arrow[dl] & & x_3 \arrow[ddl] \\
& \overline{x_3} \arrow[drr] & & & \\
& & & \square &
\end{tikzcd}$$

Figure 18: First resolution refutation

$$\begin{tikzcd}
x_2 \lor x_3 \arrow[dr] & & \overline{x_2} \arrow[dl] & & x_3 \arrow[ddl] \\
& \begin{array}{c}\overline{x_3}\\ \overline{x_2} \lor \overline{x_3}\end{array} \arrow[drr] & & & \\
& & & \square &
\end{tikzcd}$$

Figure 19: First max-refutation

$$\begin{tikzcd}
\overline{x_1} \lor x_3 \arrow[dr] & & x_1 \arrow[dl] \arrow[ddr] & & \overline{x_1} \lor x_2 \arrow[ddl] & & \overline{x_2} \lor \overline{x_3} \arrow[dddl] \\
& x_3 \arrow[dddrr] & & & & & \\
& & & x_2 \arrow[drr] & & & \\
& & & & & \overline{x_3} \arrow[dll] & \\
& & & \square & & &
\end{tikzcd}$$

Figure 20: Second resolution refutation

$$\begin{tikzcd}
\overline{x_1} \lor x_3 \arrow[ddrrr] & x_1 \arrow[dddr, bend right] & & & \overline{x_1} \lor x_2 \arrow[dr] & & \overline{x_2} \lor \overline{x_3} \arrow[dl] \\
& & & & & \begin{array}{c}\overline{x_1} \lor \overline{x_3}\\ \overline{x_1} \lor x_2 \lor x_3\\ x_1 \lor \overline{x_2} \lor \overline{x_3}\end{array} \arrow[dll, bend right] & \\
& & & \overline{x_1} \arrow[dl] & & & \\
& & \square & & & &
\end{tikzcd}$$

Figure 21: Second max-refutation

4.2 MS-Checker

MS-Checker receives two files corresponding respectively to the formula and to the computed certificate. For each inference rule applied in the certificate, MS-Checker verifies that the transformation is valid and that the premises are in the current formula, then applies it. It also checks that the number of computed empty clauses is equal to the claimed optimum of the certificate and that the provided assignment is a model of the final formula (without the empty clauses). MS-Checker can be described by Algorithm 2.

Algorithm 2 MS-Checker

```
Require: Formula $\phi$, Max-SAT certificate $C$ for $\phi$
Ensure: True if $C$ is a valid certificate for $\phi$, False otherwise
 1: for each inference step $s \in C$ do
 2:     if a premise in $s$ is missing in $\phi$ then
 3:         return False
 4:     else
 5:         $\phi \leftarrow$ apply_inference_step($s, \phi$)
 6:     end if
 7: end for
 8: $(\phi, opt) \leftarrow$ remove_empty_clauses($\phi$)
 9: if opt(C) $\neq opt$ or model(C) does not satisfy $\phi$ then
10:     return False
11: end if
12: return True
```

4.3 Experiments

We have implemented MS-Builder and MS-Checker in C++[2]. Resolution Refutations are computed using Booleforce (Biere, 2010; Eén & Sörensson, 2003) and Tracecheck (Biere, 2006, 2008). We considered the benchmark of the unweighted partial track and of the weighted partial track of the 2020 Max-SAT Evaluation (Bacchus et al., 2020). The experiments are performed on Dell PowerEdge M620 servers with Intel XeonSilver E5-2609 processors (clocked at 2.5∽2.6 GHz) under Ubuntu 18.04. Each solving process is allocated a slot of 1 hour and at most 16 GB of memory per instance.

MS-Builder has succeeded to construct full proofs for 163 instances over 576 unweighted partial instances and for 144 instances over 600 weighted partial instances. The running time for building Max-SAT certificates is in Figure 22. We can see that the weighted benchmark seems to be more difficult than the unweighted one although we added core minimization techniques (Ignatiev et al., 2019; Silva, 2010).

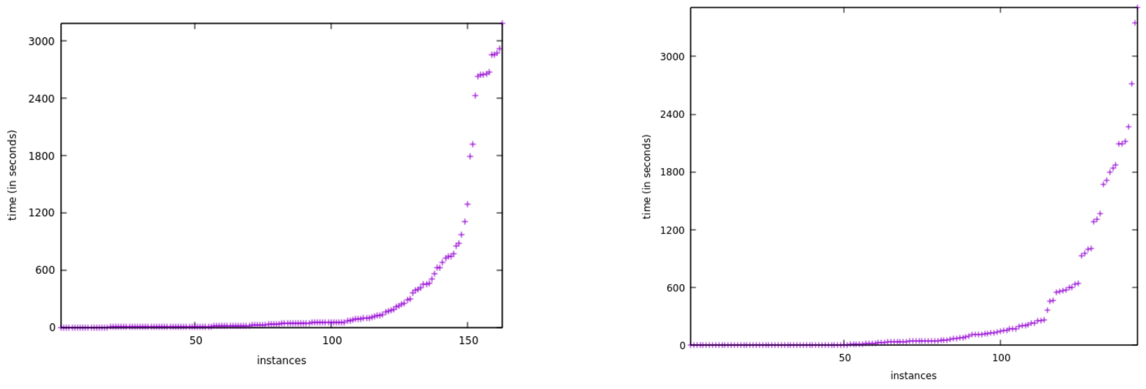


Figure 22: Running time for building complete proofs for unweighted and weighted instances

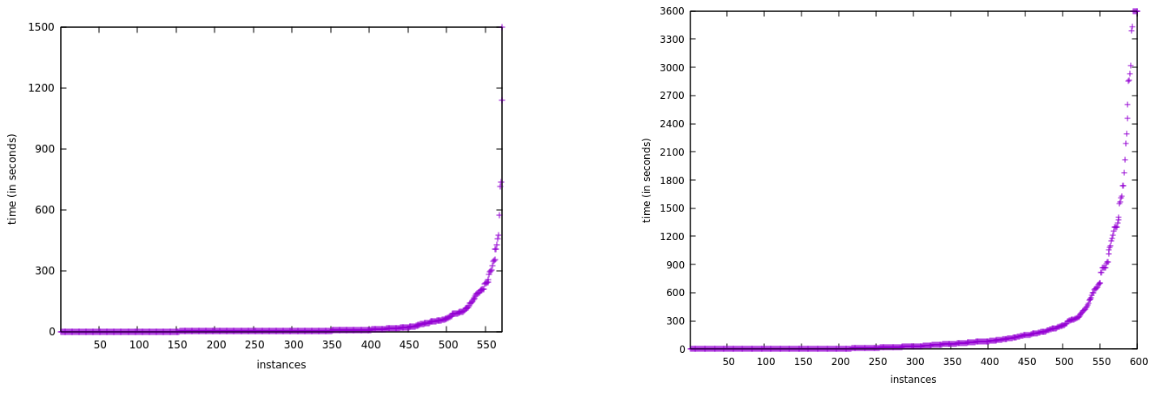


Figure 23: Running time for checking proofs for unweighted and weighted instances

2. the source code is available on `https://pageperso.lis-lab.fr/matthieu.py/en/software.html`

MS-Checker has succeeded to check 575 complete or partial certificates over 576 for the unweighted benchmarks and 593 complete or partial certificates over 600 for the unweighted benchmarks. Proof checking is obviously much easier than proof building except on rare formulas with a large number of clauses which can make difficult the operation of extracting a premise or exceed the memory limit. The running time for checking certificates is in Figure 23. MS-Builder has also succeed to build at least half of the proofs (with respect to the number of empty clauses) of 302 instances over 463 unweighted instances and of 326 instances over 489 weighted instances for which the optimum cost is known. This is illustrated in Figure 24 which reports the percentage of empty clauses built per instance. The sizes of the computed proofs vary from a few bytes to 1 Gigabyte as illustrated in Figure 25. Notice how empty (incomplete) proofs are computed for some very hard instances for which the timeout is not sufficient to even compute the first Max-SAT refutation. On the other hand, there are some unweighted instances, usually with an optimum cost of 1, which have very small proofs.

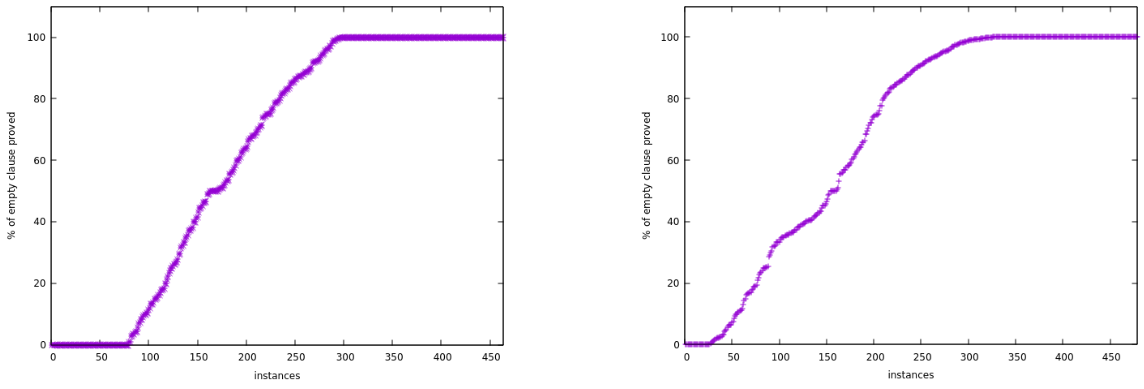


Figure 24: Percentage of proved □ per instance for unweighted and weighted instances

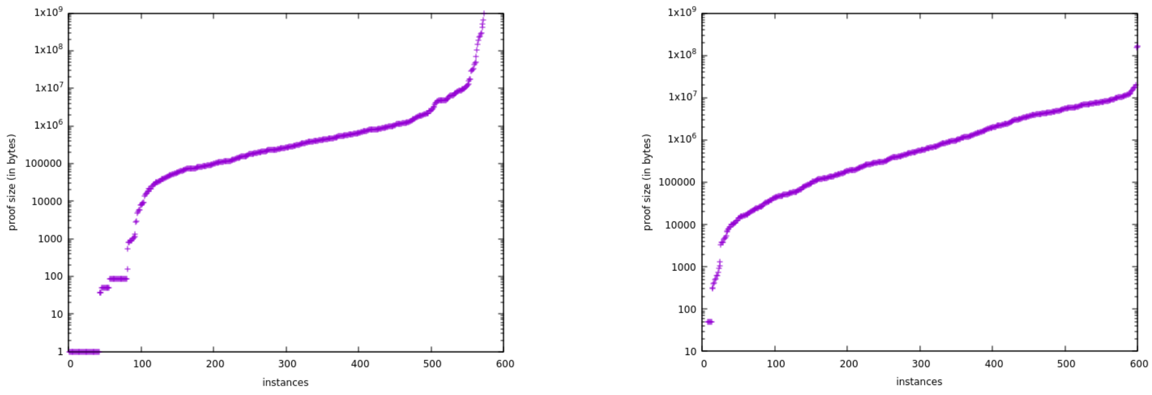


Figure 25: Size of proof per instance (in logarithmic scale) for unweighted and weighted instances

Type of resolution refutation	Unweighted instances		Weighted instances	
	Number	Percentage	Number	Percentage
read-once	169,239	83.7 %	135,594	35.60 %
semi-read-once	24,556	12.1 %	87,748	23.04 %
tree-like regular	2,879	1.4 %	23,612	6.20 %
tree-like	1,795	0.9 %	87,529	22.99 %
unrestricted	3,799	1.9 %	46,337	12.17 %

Table 2: Encountered types of resolution refutations in the whole benchmark

Finally, we can observe in Table 2 how the resolution refutation types are different for unweighted and weighted instances (semi-tree-like refutations are merged with tree-like refutations using a preprocessing technique). Indeed, while the percentage of read-once and semi-read-once resolution refutations is 83.7 % in the unweighted benchmark, it is only 35.60 % in the weighted benchmark. Such a difference can explain why weighted instances are harder to solve than unweighted instances. We further observed experimentally that, when MS-Builder is working on an instance, the first resolution refutations are often easy (i.e., read-once) and small while the last ones are often hard (i.e., unrestricted) and huge, that is why the timeout often stops on the last resolution refutations.

5. Conclusion

In this paper, we contributed to the generation of certificates for Max-SAT. First, we proposed an adaption from any resolution refutation into a max-refutation The showed that the proposed adaptation is linear in terms of the number of inference steps of the computed max-refutation with respect to the initial resolution refutation for the following cases: semi-read-once resolution, tree-like regular resolution, tree-like resolution and semi-tree-like resolution. These results are achieved by augmenting Max-SAT resolution with the split rule which enabled us to duplicate clauses by adding literals when necessary. However, our generic adaptation for unrestricted resolution refutations has a worst-case exponential blow-up in the size of the proofs. Then, we devised two tools, called MS-Builder and MS-Checker, to respectively generate and check certificates for the Max-SAT problem. The certificates are represented in the form of a Max-SAT-equivalence-preserving transformation from the initial formula into a formula composed of a set of empty clauses and a satisfiable sub-formula. To compute the transformation, MS-Builder iteratively calls a SAT oracle to get a resolution refutation, adapts it for Max-SAT and applies it to the current formula. These tools were evaluated on the unweighted and weighted instances of the 2020 Max-SAT Evaluation (Bacchus et al., 2020).

As future work, it would be relevant to study if there exists an adaptation of resolution refutations for Max-SAT which is polynomial in the unrestricted case, or to prove that such an adaptation is not possible. Moreover, it would be interesting to study cases where we can extend the treatment provided for unit clauses and the semi-read-once case. In particular, the diamond patterns can be transformed using the unit-propagation-fixing preprocessing (applied on a non-unit clause), showing that this preprocessing can sometimes be generalized for non-unit clauses. It would be also relevant to improve the implementation of both our tools with more adapted data structures or more advanced techniques such as stratification (Ansótegui, Bonet, Gabàs, & Levy, 2012). Finally, it would be interesting to study whether efficient heuristics can be devised to encourage a SAT oracle to generate specific types of refutations, for which a cost-effective adaptation is possible.

Acknowledgments

This work was conducted in the Computer Science and Systems Laboratory (LIS - UMR 7020) of Aix-Marseille University. We thank the anonymous reviewers for their efforts and valuable input.

References

Abramé, A., & Habet, D. (2014). Ahmaxsat: Description and Evaluation of a Branch and Bound Max-SAT Solver. *J. Satisf. Boolean Model. Comput.*, *9*(1), 89–128.

Achá, R. J. A., & Nieuwenhuis, R. (2014). Curriculum-based course timetabling with SAT and MaxSAT. *Ann. Oper. Res.*, *218*(1), 71–91.

Ansótegui, C., Bonet, M. L., Gabàs, J., & Levy, J. (2012). Improving SAT-Based Weighted MaxSAT Solvers. In Milano, M. (Ed.), *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, Vol. 7514 of *Lecture Notes in Computer Science*, pp. 86–101. Springer.

Ansótegui, C., Bonet, M. L., & Levy, J. (2009). Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In Kullmann, O. (Ed.), *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, Vol. 5584 of *Lecture Notes in Computer Science*, pp. 427–440. Springer.

Ansótegui, C., Bonet, M. L., & Levy, J. (2013). SAT-based MaxSAT algorithms. *Artif. Intell.*, *196*, 77–105.

Atserias, A., & Lauria, M. (2019). Circular (yet sound) proofs. In Janota, M., & Lynce, I. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, Vol. 11628 of *Lecture Notes in Computer Science*, pp. 1–18. Springer.

Bacchus, F., Järvisalo, M., & Martins, R. (2020). MaxSAT Evaluation. `https://maxsat-evaluations.github.io/2020/`.

Ben-Sasson, E., Impagliazzo, R., & Wigderson, A. (2004). Near Optimal Separation Of Tree-Like And General Resolution. *Comb.*, *24*(4), 585–603.

Biere, A. (2006). TraceCheck. `http://fmv.jku.at/tracecheck/`.

Biere, A. (2008). PicoSAT Essentials. *J. Satisf. Boolean Model. Comput.*, *4*(2-4), 75–97.

Biere, A. (2010). Booleforce. `http://fmv.jku.at/booleforce/`.

Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.). (2021). *Handbook of Satisfiability* (2 edition)., Vol. 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Bofill, M., Garcia, M., Suy, J., & Villaret, M. (2015). MaxSAT-Based Scheduling of B2B Meetings. In Michel, L. (Ed.), *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, Vol. 9075 of *Lecture Notes in Computer Science*, pp. 65–73. Springer.

Bonet, M. L., & Levy, J. (2020). Equivalence Between Systems Stronger Than Resolution. In Pulina, L., & Seidl, M. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, Vol. 12178 of *Lecture Notes in Computer Science*, pp. 166–181. Springer.

Bonet, M. L., Levy, J., & Manyà, F. (2006). A Complete Calculus for Max-SAT. In Biere, A., & Gomes, C. P. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, Vol. 4121 of *Lecture Notes in Computer Science*, pp. 240–251. Springer.

Bonet, M. L., Levy, J., & Manyà, F. (2007). Resolution for Max-SAT. *Artif. Intell.*, *171*(8-9), 606–618.

Cherif, M. S., Habet, D., & Abramé, A. (2020). Understanding the power of Max-SAT resolution through UP-resilience. *Artif. Intell.*, *289*, 103397.

Cherif, M. S., Habet, D., & Py, M. (2022). From Crossing-Free Resolution to Max-SAT Resolution. In Solnon, C. (Ed.), *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*, Vol. 235 of *LIPIcs*, pp. 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

D'Almeida, D., & Grégoire, É. (2012). Model-based diagnosis with default information implemented through MAX-SAT technology. In Zhang, C., Joshi, J., Bertino, E., & Thuraisingham, B. (Eds.), *IEEE 13th International Conference on Information Reuse & Integration, IRI 2012, Las Vegas, NV, USA, August 8-10, 2012*, pp. 33–36. IEEE.

Davies, J., & Bacchus, F. (2011). Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. In Lee, J. H. (Ed.), *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, Vol. 6876 of *Lecture Notes in Computer Science*, pp. 225–239. Springer.

Demirovic, E., & Musliu, N. (2017). MaxSAT-based large neighborhood search for high school timetabling. *Comput. Oper. Res.*, *78*, 172–180.

Eén, N., & Sörensson, N. (2003). An Extensible SAT-solver. In Giunchiglia, E., & Tacchella, A. (Eds.), *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, Vol. 2919 of *Lecture Notes in Computer Science*, pp. 502–518. Springer.

Filmus, Y., Mahajan, M., Sood, G., & Vinyals, M. (2020). MaxSAT Resolution and Subcube Sums. In Pulina, L., & Seidl, M. (Eds.), *Theory and Applications of Satisfiability*

Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings, Vol. 12178 of *Lecture Notes in Computer Science*, pp. 295–311. Springer.

Fu, Z., & Malik, S. (2006). On Solving the Partial MAX-SAT Problem. In Biere, A., & Gomes, C. P. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, Vol. 4121 of *Lecture Notes in Computer Science*, pp. 252–265. Springer.

Guerra, J., & Lynce, I. (2012). Reasoning over Biological Networks Using Maximum Satisfiability. In Milano, M. (Ed.), *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, Vol. 7514 of *Lecture Notes in Computer Science*, pp. 941–956. Springer.

Heras, F., & Marques-Silva, J. (2011). Read-Once Resolution for Unsatisfiability-Based Max-SAT Algorithms. In Walsh, T. (Ed.), *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 572–577. IJCAI/AAAI.

Hertel, A., & Urquhart, A. (2009). Algorithms and Complexity Results for Input and Unit Resolution. *J. Satisf. Boolean Model. Comput.*, *6*(1-3), 141–164.

Ignatiev, A., Morgado, A., & Marques-Silva, J. (2019). RC2: an Efficient MaxSAT Solver. *J. Satisf. Boolean Model. Comput.*, *11*(1), 53–64.

Iwama, K., & Miyano, E. (1995). Intractability of Read-Once Resolution. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pp. 29–36. IEEE Computer Society.

Küegel, A. (2012). Improved Exact Solver for the Weighted MAX-SAT Problem. In *POS-10. Pragmatics of SAT*, Vol. 8 of *EPiC Series in Computing*, pp. 15–27. EasyChair.

Larrosa, J., & Heras, F. (2005). Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pp. 193–198. Professional Book Center.

Larrosa, J., Heras, F., & de Givry, S. (2008). A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, *172*(2), 204–233.

Larrosa, J., & Rollon, E. (2020a). Augmenting the Power of (Partial) MaxSat Resolution with Extension. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 1561–1568. AAAI Press.

Larrosa, J., & Rollon, E. (2020b). Towards a Better Understanding of (Partial Weighted) MaxSAT Proof Systems. In Pulina, L., & Seidl, M. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, Vol. 12178 of *Lecture Notes in Computer Science*, pp. 218–232. Springer.

Li, C. M., Manyà, F., & Planes, J. (2007). New Inference Rules for Max-SAT. *J. Artif. Intell. Res.*, *30*, 321–359.

Li, C. M., Manyà, F., & Soler, J. R. (2016). A Clause Tableau Calculus for MaxSAT. In Kambhampati, S. (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 766–772. IJCAI/AAAI Press.

Li, C., Xu, Z., Coll, J., Manyà, F., Habet, D., & He, K. (2022). Boosting branch-and-bound MaxSAT solvers with clause learning. *AI Commun.*, *35*(2), 131–151.

Loveland, D. (1970). A linear format for resolution. In *Symposium on Automatic Demonstration*, pp. 147–162.

Manquinho, V. M., Silva, J. P. M., & Planes, J. (2009). Algorithms for Weighted Boolean Optimization. In Kullmann, O. (Ed.), *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, Vol. 5584 of *Lecture Notes in Computer Science*, pp. 495–508. Springer.

Manyà, F., Negrete, S., Roig, C., & Soler, J. R. (2020). Solving the Team Composition Problem in a Classroom. *Fundam. Informaticae*, *174*(1), 83–101.

Martins, R., Manquinho, V. M., & Lynce, I. (2014). Open-WBO: A Modular MaxSAT Solver. In Sinz, C., & Egly, U. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, Vol. 8561 of *Lecture Notes in Computer Science*, pp. 438–445. Springer.

Morgado, A., Heras, F., Liffiton, M. H., Planes, J., & Marques-Silva, J. (2013). Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints An Int. J.*, *18*(4), 478–534.

Muise, C. J., Beck, J. C., & McIlraith, S. A. (2016). Optimal Partial-Order Plan Relaxation via MaxSAT. *J. Artif. Intell. Res.*, *57*, 113–149.

Narodytska, N., & Bacchus, F. (2014). Maximum Satisfiability Using Core-Guided MaxSAT Resolution. In Brodley, C. E., & Stone, P. (Eds.), *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pp. 2717–2723. AAAI Press.

Py, M., Cherif, M. S., & Habet, D. (2020). Towards Bridging the Gap Between SAT and Max-SAT Refutations. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*, pp. 137–144. IEEE.

Py, M., Cherif, M. S., & Habet, D. (2021a). A Proof Builder for Max-SAT. In Li, C., & Manyà, F. (Eds.), *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, Vol. 12831 of *Lecture Notes in Computer Science*, pp. 488–498. Springer.

Py, M., Cherif, M. S., & Habet, D. (2021b). Computing Max-SAT Refutations using SAT Oracles. In *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021*, pp. 404–411. IEEE.

Py, M., Cherif, M. S., & Habet, D. (2021c). Inferring Clauses and Formulas in Max-SAT. In *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021*, pp. 632–639. IEEE.

Robinson, J. A. (1965). A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, *12*(1), 23–41.

Silva, J. P. M. (2010). Minimal Unsatisfiability: Models, Algorithms and Applications (Invited Paper). In *40th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2010, Barcelona, Spain, 26-28 May 2010*, pp. 9–14. IEEE Computer Society.

Silva, J. P. M., & Sakallah, K. A. (1996). GRASP - a new search algorithm for satisfiability. In Rutenbar, R. A., & Otten, R. H. J. M. (Eds.), *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*, pp. 220–227. IEEE Computer Society / ACM.

Urquhart, A. (1995). The complexity of propositional proofs. *Bull. Symb. Log.*, *1*(4), 425–467.

Urquhart, A. (2011). A Near-Optimal Separation of Regular and General Resolution. *SIAM J. Comput.*, *40*(1), 107–121.

Xu, H., Rutenbar, R. A., & Sakallah, K. A. (2003). Sub-SAT: a formulation for relaxed Boolean satisfiability with applications in routing. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, *22*(6), 814–820.

Zhang, L., & Bacchus, F. (2012). MAXSAT Heuristics for Cost Optimal Planning. In Hoffmann, J., & Selman, B. (Eds.), *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press.