Read-Once Resolution for Unsatisfiability-Based Max-SAT Algorithms *

Federico Heras and Joao Marques-Silva
University College Dublin
Dublin, Ireland
fheras@ucd.ie and jpms@ucd.ie

Abstract

This paper proposes the integration of the resolution rule for Max-SAT with *unsatisfiability-based* Max-SAT solvers. First, we show that the resolution rule for Max-SAT can be safely applied as dictated by the resolution proof associated with an unsatisfiable core when such proof is *read-once*, that is, each clause is used at most once in the resolution process. Second, we study how this property can be integrated in an unsatisfiability-based solver. In particular, the resolution rule for Max-SAT is applied to read-once proofs or to read-once subparts of a general proof. Finally, we perform an empirical investigation on *structured* instances from recent Max-SAT evaluations. Preliminary results show that the use of read-once resolution substantially improves the performance of the solver.

1 Introduction

The *Satisfiability* problem in propositional logic (*SAT*) is the task of deciding whether a given propositional formula has a model. *Max-SAT* is an optimization variant of SAT and it can be seen as a generalisation of the SAT problem. Given a propositional formula in conjunctive normal form (*CNF*), the objective of the *Max-SAT* problem is to find a variable assignment that maximizes the number of satisfied clauses.

In *weighted Max-SAT*, each clause has an associated *weight* and the goal becomes maximizing the sum of the weights of the satisfied clauses. In many problems coming from real world domains, a subset of the (*hard*) clauses must be satisfied, and the remaining (*soft*) clauses may be satisfied or not.

Algorithms for Max-SAT have been the subject of significant improvements over the last years and several families of algorithms have been developed including *branch and bound* algorithms [Li *et al.*, 2007; Larrosa *et al.*, 2008], approaches based on reformulating Max-SAT as a *Weighted CSP* [de Givry *et al.*, 2003], algorithms based on iteratively calling a SAT solver [Berre and Parrain, 2010], *compilation-based* algorithms [Pipatsrisawat *et al.*, 2008], and algorithms based on computing *unsatisfiable cores* with a SAT solver [Fu and Malik, 2006]. In particular, two families of algorithms have been intensively investigated.

The first family of algorithms is based on a *branch and bound* algorithm and they perform quite well on *random* and *crafted* instances [Li *et al.*, 2007; Larrosa *et al.*, 2008; Heras *et al.*, 2008]. To reduce the search space, they compute a *lower bound* formed by an *underestimation* component and an *inference* component which applies *inference rules* based on the resolution rule for Max-SAT [Bonet *et al.*, 2007; Larrosa *et al.*, 2008] to transform the formula into an equivalent but simpler one.

The second family identifies *unsatisfiable* sub-formulas by performing successive calls to a SAT solver. Each call returns a *trace* from which a *resolution proof* of its unsatisfiability can be built [Zhang and Malik, 2003]. Each soft clause in the trace is extended with *relaxation variables* and *cardinality constraints* are added for the subsequent SAT call. These algorithms are highly competitive for *industrial* problems [Fu and Malik, 2006; Marques-Silva and Planes, 2008; Manquinho *et al.*, 2009; Ansótegui *et al.*, 2009; 2010] and we will refer to them as *unsatisfiability-based* algorithms.

Unsatisfiability-based Max-SAT algorithms perform well on industrial problem instances. However, the number of relaxation variables, and the number and size of cardinality constraints can represent a drawback for these algorithms. One approach to reduce the number of relaxation variables and (indirectly) the number and size of cardinality constraints would be to actually apply Max-SAT resolution [Bonet *et al.*, 2007; Larrosa *et al.*, 2008] on the resolution proofs generated by unsatisfiability-based algorithms.

Building on this idea, this paper proposes the integration of the techniques used in both families of algorithms. In particular, we study how the resolution rule for Max-SAT can be applied in unsatisfiability-based solvers. First, we show that the resolution rule for Max-SAT can be safely applied as dictated by the resolution proof associated with an unsatisfiable core when such proof is *read-once* [Iwama and Miyano, 1995], that is, each clause is used at most once in the whole resolution process. Second, we study how this property can be integrated in an unsatisfiability-based solver. Essentially, the resolution rule for Max-SAT is applied to read-once proofs or to read-once subparts of a general resolution proof. Note that we focus our study in the algorithm of [Fu and Malik, 2006] and extended to han-

*This work is partially supported by SFI grant BEACON (09/PI/I2618).

dle weighted partial Max-SAT in [Ansótegui *et al.*, 2009; Manquinho *et al.*, 2009], but read-once resolution could be also applied to other unsatisfiability-based Max-SAT solvers. Finally, we perform an empirical investigation on *crafted* and *industrial* instances of recent Max-SAT evaluations. The results show that the performance of an unsatisfiability-based Max-SAT solver is clearly improved in many benchmarks.

The paper is organized as follows. Section 2 introduces preliminary definitions and notation. Section 3 presents an unsatisfiability-based algorithm for Max-SAT. Section 4 studies *read-once* resolution in the Max-SAT context and Section 5 proposes its integration on unsatisfiability-based Max-SAT solvers. The experimental investigation is presented in Section 6. Finally, Section 7 points out some concluding remarks and future work.

2 Definitions

In this Section we introduce the necessary definitions and notation related to the SAT and Max-SAT problems.

2.1 SAT

We define $X = \{x_1, x_2, \ldots, x_n\}$ as the set of Boolean variables. A *literal* is either a variable x_i or its negation $\bar{x}_i$. The variable to which a literal l refers is denoted by $var(l)$. Given a literal l, its negation $\bar{l}$ is $\bar{x}_i$ if l is x_i and it is x_i if l is $\bar{x}_i$.

A *clause* C is a disjunction of literals. Hereafter, capital letters will represent clauses. The *size* of a clause, noted $|C|$, is the number of literals that it has. A formula in *conjunctive normal form* (CNF) φ is a set of clauses.

An *assignment* is a set of literals $A = \{l_1, l_2, \ldots, l_k\}$ such that for all $l_i \in A$, its variable $var(l_i) = x_i$ is assigned value $true$ or $false$. If variable x_i is assigned to $true$, literal x_i is *satisfied* and literal $\bar{x}_i$ is *falsified*. Similarly, if variable x_i is assigned $false$, literal $\bar{x}_i$ is satisfied and literal x_i is falsified. If all variables in X are assigned, the assignment is called *complete*, otherwise it is called *partial*. An assignment *satisfies* a literal iff it belongs to the assignment, it satisfies a clause iff it satisfies one or more of its literals and it *falsifies* a clause iff it contains the negation of all its literals. The *empty clause* noted $\square$ has no literals (size 0) and cannot be satisfied. If a clause is falsified by an assignment, the clause is *conflicting* and it can be represented using the empty clause. A *model* is a complete assignment that satisfies all the clauses in a CNF formula φ. The SAT problem is the task of finding a model for a given formula.

The *Classical resolution* is a well-known sound and complete inference mechanism for SAT. Given two *clashing clauses* $(x \vee A)$ and $(\bar{x} \vee B)$ then the clause $(A \vee B)$ can be added to the formula. The new clause is usually referred to as *resolvent*. Clauses of size one are called unit clauses. When a formula contains a unit clause l , it can be simplied by removing all clauses containing l and removing $\bar{l}$ from all the clauses where it appears. The application of this rule until fixed-point is called *unit propagation* and it can be understood as a resolution-based simplification.

Given a CNF formula, a subset of clauses is *inconsistent* if all them cannot be simultaneously satisfied by any variable assignment. The clauses involved in an inconsistent subset of clauses can be determined with several steps of resolution. This process is called *resolution proof*. Let each clause C be identified by a label i and denote it as C_i. A resolution proof is an ordered set:

$$\mathcal{R} = \{C_i = (C_{i'} \bowtie C_{i''}), C_{i+1} = (C_{i'+1} \bowtie C_{i''+1}), \ldots, C_{i+k} = (C_{i'+k} \bowtie C_{i''+k})\}$$

Each clause C_j with $i \leq j \leq k$ is the resolvent of applying resolution (represented by symbol $\bowtie$) between clashing clauses $C_{j'}$ and $C_{j''}$. The last resolvent in the resolution proof is always the empty clause (i.e. $C_{i+k} = \square$). Clashing clauses generated in a previous resolution step of a resolution proof are called *derived* clauses, otherwise they are *original*. If all the clauses involved in a resolution proof are used at most once, it is called a *read-once resolution proof* [Iwama and Miyano, 1995], otherwise it is a *general resolution proof*.

Given an unsatisfiable SAT formula φ, a subset of clauses φ_C whose conjunction is still unsatisfiable is called an *unsatisfiable core* (or *trace*) [Zhang and Malik, 2003] of the original formula ($\varphi_C \subseteq \varphi$). Modern SAT solvers can be instructed to generate an unsatisfiable core or a general resolution proof for unsatisfiable formulas [Zhang and Malik, 2003]. Observe that an unsatisfiable core only contains the original clauses used in the related resolution proof.

2.2 Max-SAT

A *weighted* clause is a pair (C, w), where C is a clause and w is the cost of its falsification, also called its *weight*. Many real problems contain clauses that *must* be satisfied. We call these clauses *mandatory* or *hard* and we associate to them a special weight $\top$. Note that any weight $w \geq \top$ indicates that the associated clause must be necessarily satisfied. Thus, we can replace w by $\top$ without changing the problem. Consequently, we can assume all weights in the interval $[0..\top]$. Non-mandatory clauses are also called *soft* clauses. A *weighted* formula in *conjunctive normal form* (WCNF) φ is a set of weighted clauses.

A *model* is a complete assignment that satisfies all mandatory clauses. The *cost of an assignment* is the sum of weights of the clauses that it falsifies. Given a WCNF formula, *Weighted* Max-SAT is the problem of finding a model of minimum cost. Note that if a formula contains only mandatory clauses, weighted Max-SAT is equivalent to classical SAT. If all the clauses have weight 1, we have the (unweighted) Max-SAT problem. Hereafter, we will assume weighted Max-SAT.

A weighted formula φ' is a *relaxation* of φ (noted $\varphi' \sqsubseteq \varphi$) if the optimal cost of φ' is less than or equal to the optimal cost in φ (non-models are considered to have cost infinity). Two weighted formulas φ' and φ are *equivalent* (denoted as $\varphi' \equiv \varphi$) if $\varphi' \sqsubseteq \varphi$ and $\varphi \sqsubseteq \varphi'$.

Let u and w be two weights. Their sum is defined as, $u \oplus w = min(u + w, \top)$ in order to keep the result within the interval $[0..\top]$. If $u \geq w$, their subtraction is defined as,

$$u \ominus w = \begin{cases} u - w & : \quad u \neq \top \\ \top & : \quad u = \top \end{cases}$$

The De Morgan laws cannot be used in Max-SAT [Bonet *et al.*, 2007; Larrosa *et al.*, 2008]. Instead, the following rule should be repeatedly used until the conjunctive normal form is achieved: $(A \vee \overline{l \vee C}, w) \equiv$

$\{(A \vee \bar{C}, w), (A \vee \bar{l} \vee C, w)\}$. If a formula contains clauses (C, u) and (C, v), they can be replaced by $(C, u \oplus v)$. If a formula contains the clause $(C, 0)$, such clause can be removed. The empty clause may appear in a formula. If its weight is $\top$, i.e. $(\square, \top)$, it is clear that the formula does not have any model. If its weight is w, i.e. $(\square, w)$, the cost of any assignment will include that weight, therefore w is a *lower bound* of the formula optimal cost.

The resolution rule can be extended from SAT to Max-SAT [Bonet *et al.*, 2007; Larrosa *et al.*, 2008] as $\{(x \vee A, u), (\bar{x} \vee B, w)\} \equiv \{(A \vee B, m), (x \vee A, u \ominus m), (\bar{x} \vee B, w \ominus m), (x \vee A \vee B, m), (\bar{x} \vee \bar{A} \vee B, m)\}$ where $m = \min(u, w)$. Briefly, $(x \vee A, u)$ and $(\bar{x} \vee B, w)$ are the *clashing* clauses, $(A \vee B, m)$ is the *resolvent*, $(x \vee A, u \ominus m)$ and $(\bar{x} \vee B, w \ominus m)$ are the *posterior clashing* clauses, and $(x \vee A \vee \bar{B}, m)$ and $(\bar{x} \vee \bar{A} \vee B, m)$ are the *compensation* clauses.

The resolution step i of a resolution proof $\mathcal{R}$ including weighted clauses is noted $(C_i, w_i) = (C_{i'}, w_{i'}) \bowtie (C_{i''}, w_{i''})$, where (C_i, w_i) is the resolvent and $(C_{i'}, w_{i'})$ and $(C_{i''}, w_{i''})$ are the clashing clauses. The set of compensation clauses produced by such resolution step will be noted as $[(C_{i'}, w_{i'}) \bowtie (C_{i''}, w_{i''})]$.

3 An Unsatisfiability-Based Max-SAT Solver

Algorithm 1 summarizes a unsatisfiability-based Max-SAT algorithm to handle weighted partial Max-SAT formulas [Manquinho *et al.*, 2009; Ansótegui *et al.*, 2009]. The algorithm iteratively finds unsatisfiable cores by calling a SAT solver (line 4). Lines 11-21 should be omitted as they introduce read-once resolution that will be explained later.

The input of Algorithm 1 is a Max-SAT formula φ. Besides, it maintains a *working formula* φ_W and a *lower bound* λ of the optimal solution which are initialized to φ and 0, respectively. For each unsatisfiable sub-formula φ_C, the minimum weight m of the soft clauses (i.e. (C, w) with $w < \top$) in the sub-formula (line 9) is computed. Then, the minimum weight is used to update the lower bound (line 10). Soft clauses in the unsatisfiable sub-formula are relaxed (lines 22 to 30) by adding new relaxation variables and requiring that exactly one of the relaxation variables must be assigned to true. This is usually expressed by adding cardinality constraints (line 31). Observe that if the weight of a clause is larger than m, then a copy clause of the original clause with weight m is created (and relaxed), and the weight of the original clause is decreased by m.

The first unsatisfiability-based algorithm for Max-SAT [Fu and Malik, 2006] was designed to handle unweighted partial Max-SAT, it added several relaxation variables to the same clause and it used the *pairwise encoding* for the cardinality constraints. More sophisticated algorithms have been developed in recent years, which add only one relaxation variable per clause, use alternative encodings for the cardinality constraints, and are able to handle weighted soft clauses and hard clauses [Marques-Silva and Planes, 2008; Manquinho *et al.*, 2009; Ansótegui *et al.*, 2009; 2010].

Theorem 1 *The value returned by Algorithm 1 (excluding lines 11-21) is the minimum cost of unsatisfied clauses in φ.*

Proof 1 *Proof included in [Ansótegui* et al.*, 2009].*

Algorithm 1 Weighted partial Max-SAT algorithm enhanced with read-once resolution

```
SOLVE(φ)
 1  φ_W ← φ
 2  λ ← 0
 3  while true
 4      do (status, φ_C) ← SAT(φ_W)
 5         if status = true
 6           then ▷ Solution to Weighted Max-SAT problem
 7                return λ
 8         φ_W ← φ_W − φ_C
 9         m ← min{w|(C, w) ∈ φ_C ∧ w < ⊤}
10         λ ← λ + m
11         R_C ← getProof(φ_C)
12         for each (C_i, w_i) = (C_i', w_i') ⋈ (C_i'', w_i'') ∈ R_C
13             do
14                if ROR((C_i, w_i), R_C)
15                  then (C_i', w_i') ← (C_i', w_i' ⊖ m)
16                       (C_i'', w_i'') ← (C_i'', w_i'' ⊖ m)
17                       φ_C ← φ_C − {(C_i', w_i'), (C_i'', w_i'')}
18                       φ_W ← φ_W ∪ {(C_i', w_i'), (C_i'', w_i'')}
19                       φ_W ← φ_W ∪ {[(C_i', w_i') ⋈ (C_i'', w_i'')]}
20                       if C_i ≠ □
21                         then φ_C ← φ_C ∪ {(C_i, m)}
22         V_R ← ∅
23         for each (C, w) ∈ φ_C with w < ⊤
24             do
25                r is a new relaxation variable
26                V_R ← V_R ∪ {r}
27                φ_C ← φ_C ∪ {(C ∪ {r}, m)}
28                if w > m
29                  then (C, w) ← (C, w ⊖ m)
30                  else φ_C ← φ_C − {(C, w)}
31         φ_C ← CNF(∑_{r∈V_R} r = 1)
32         φ_W ← φ_W ∪ φ_C
```

4 Read-Once Resolution in Max-SAT

For each unsatisfiable core, unsatisfiability-based Max-SAT algorithms add a set of relaxation variables and cardinality constraints to the formula. Recall that we can always retrieve a resolution proof from any unsatisfiable core. A different approach would be to apply Max-SAT resolution as dictated by such a resolution proof in order to increase the weight of the empty clause. As a result, the relaxation variables and cardinality constraints would not be necessary. However, this approach cannot be applied in general:

Example 1 *Let $\varphi = \{(x_1 \vee x_2, 1)_{i1}, (\bar{x}_1 \vee x_2, 1)_{i2}, (\bar{x}_2 \vee x_3, 1)_{i3}, (\bar{x}_2 \vee \bar{x}_4, 1)_{i4}, (\bar{x}_3 \vee x_4, 1)_{i5}\}$ and a SAT solver returned the following resolution proof $\mathcal{R} = \{(i6 = i1 \bowtie i2), (i7 = i6 \bowtie i3), (i8 = i7 \bowtie i5), (i9 = i6 \bowtie i4), (i10 = i8 \bowtie i9)\}$. The soft clause with identifier $i6$ is used twice in the proof. In the Max-SAT context, the application of Max-SAT resolution between two clashing soft clauses produces a new resolvent but also the clashing clauses may* disappear *(ie. their weight equals 0). Hence, if we apply Max-SAT resolution as dictated by $\mathcal{R}$, we cannot continue at step $(i9 = i6 \bowtie i4)$ because clause $i6$ has been* consumed *at step $(i7 = i6 \bowtie i3)$. Then, the empty clause cannot be increased.*

In other words, when soft clauses with weights greater than 0 are resolved more than once, Max-SAT resolution does not ensure to produce resolvents with weight greater than 0. Hence, it would be interesting to study what happens if we restrict the application of resolution to the case in which each clause is used *at most once* in a proof. This process is referred to as *Read-once resolution* (*ROR*) [Iwama and Miyano, 1995]. ROR is an incomplete calculus in which pairs of clashing clauses $(x \vee A)$ and $(\bar{x} \vee B)$ are resolved into $(A \vee B)$ and then both clashing clauses disappear from the formula.

Remark 1 *ROR is an incomplete calculus, that is, it cannot generate resolution proofs for some unsatisfiable formulas. This is a known result [Iwama and Miyano, 1995].*

When we look at the recent Max-SAT resolution, we observe that it has precisely a similar behavior to read-once resolution. In particular, when both clashing clauses are soft, they may disappear after the resolution step. The main difference is that Max-SAT resolution adds new compensation clauses to recover an equivalent Max-SAT formula.

Observe that current branch and bound Max-SAT solvers apply the Max-SAT resolution rule when different unsatisfiable formulas describing a specific pattern are detected [Li *et al.*, 2007; Larrosa *et al.*, 2008; Heras and Larrosa, 2008; Li *et al.*, 2010]. Such patterns include a small set of binary and unary clauses and each clause is resolved once. In [Heras *et al.*, 2008], it is shown that for each unsatisfiable sub-formula detected by unit propagation, a resolution proof using each clause once can be always built. However, there are unsatisfiable sub-formulas with ROR proofs that cannot be detected by solely applying unit propagation:

Example 2 *Let be* $\varphi = \{(x_1 \vee x_2)_{i1}, (\bar{x}_1 \vee x_2)_{i2}, (x_1 \vee \bar{x}_2)_{i3}, (\bar{x}_1 \vee \bar{x}_2)_{i4}\}$ *and its associated ROR proof* $\mathcal{R} = \{(i5 = i1 \bowtie i2), (i6 = i3 \bowtie i4), (i7 = i5 \bowtie i6)\}$. *Clearly, unit propagation cannot prove the unsatisfiability of* φ.

In [Cooper *et al.*, 2010], the same observation reported in Example 1 is studied in the context of the *Weighted Constraint Satisfaction Problem* (*WCSP*). Recall that the Max-SAT problem can be seen as a WCSP in which variables are boolean and clauses are *forbidden tuples* of *weighted constraints*. The authors propose using *rational weights* in order to produce equivalent WCSP problems which remain in the *arc level*, that is, all transformations produce constraints of at most size two. Differently, our approach considers natural weights but it allows resulting clauses of size greater than 2.

5 Unsatisfiability-Based Max-SAT with ROR

Algorithm 1 contains the necessary code to apply Max-SAT resolution when the read-once resolution (*ROR*) criterion is executed between lines 11-21. The resolution proof $\mathcal{R}_C$ associated to each unsatisfiable core is built in line 11 [Zhang and Malik, 2003]. Then, the algorithm iterates over the resolution steps as dictated by $\mathcal{R}_C$. For each read-once step, Max-SAT resolution is actually applied (line 14 to 21). In particular, the weights of the clashing clauses $(C_{i'}, w_{i'})$ and $(C_{i''}, w_{i''})$ are decreased by m (lines 15 and 16). If the clashing clauses are soft (this detail is omitted from the pseudocode), they are removed from the unsatisfiable core φ_C (line 17) but the resolvent is added to φ_C (line 21). If the clashing clauses are hard they are just maintained in the core as they may be used again in a different resolution step. Finally, the clashing and compensation clauses are added to the working formula φ_W (lines 18 and 19).

Algorithm 2 Determines if a clause is hard or not.

```
HARD((C_i, w_i), R)
1  if Input((C_i, w_i), R) ∧ w_i = ⊤
2     then
3          return true
4  if Input((C_i, w_i), R) ∧ w_i ≠ ⊤
5     then
6          return false
7  {(C_i', w_i'), (C_i'', w_i'')} ← ancestors((C_i, w_i), R)
8  return Hard((C_i', w_i'), R) ∧ Hard((C_i'', w_i''), R)
```

Algorithm 3 Determines if a clause is hard or it and its ancestors are used at most once.

```
ROR((C_i, w_i), R)
1  if Hard((C_i, w_i), R)
2     then
3          return true
4  if Input((C_i, w)_i, R) ∧ Used(C_i, w), R) = 1
5     then
6          return true
7  if Used(C_i, w_i), R) > 1
8     then
9          return false
10 {(C_i', w_i'), (C_i'', w_i'')} ← ancestors((C_i, w_i), R)
11 return ROR((C_i', w_i'), R) ∧ ROR((C_i'', w_i''), R)
```

In what follows we introduce the read-once criterion. Let be $(C_i, w_i) = (C_{i'}, w_{i'}) \bowtie (C_{i''}, w_{i''})$ a resolution step in a proof $\mathcal{R}$. The function $ancestors((C_i, w_i), \mathcal{R})$ returns the pair of clauses $(C_{i'}, w_{i'})$ and $(C_{i''}, w_{i''})$ from which (C_i, w_i) was derived as dictated by $\mathcal{R}$. Predicate $Input((C_i, w_i), \mathcal{R})$ returns true if clause (C_i, w_i) is an original clause in proof $\mathcal{R}$ (i.e. it is not a resolvent of any step in $\mathcal{R}$), otherwise it returns false. Function $Used((C_i, w_i), \mathcal{R})$ returns how many times clause (C_i, w_i) is resolved in proof $\mathcal{R}$. Predicate $Hard((C_i, w_i), \mathcal{R})$ (see Algorihm 2) returns true if clause (C_i, w_i) is an input hard clause or if all its ancestors are hard, otherwise it returns false. Finally, predicate $ROR((C_i, w), \mathcal{R})$ (see Algorihm 3) returns true if clause (C_i, w_i) is hard or if it and all of its soft ancestors have been used at most once in the resolution proof $\mathcal{R}$.

Consider $(C_k, w_k) = (C_{k'}, w_{k'}) \bowtie (C_{k''}, w_{k''})$, where (C_k, w_k) is the last resolvent in a resolution proof $\mathcal{R}$. Clearly, if $ROR((C_k, w_k))$ is true it means that the *entire* resolution proof $\mathcal{R}$ is read-once. When this happens the algorithm does not need to add relaxation variables neither cardinality constraints. This could be a key advantage to improve the efficiency of the original algorithm. More concretely, when the last step of resolution is ROR the related resolvent is equal to the empty clause $(\square, m)$. As we have increased the lower bound in line 10 we don't add the empty clause to the for-

mula as it would mean adding twice the same contribution to the lower bound (line 20).

Observe that Algorithms 2 and 3 are in practice implemented with a standard *depth first search*, that visits each resolution proof node at most once.

Theorem 2 *Algorithm 1 with ROR (including lines 11-21) returns the minimum cost of unsatisfied clauses in φ.*

Proof 2 *(Sketch) The proof shows that* any *transformation to the working formula φ_W results in a Max-SAT equivalent formula [Ansótegui* et al.*, 2009]. This invariant needs to be maintained in the whole algorithm. The general algorithm (i.e. by excluding lines 11-21) is known to be correct, again because at each iteration φ_W is transformed into an equivalent formula [Ansótegui* et al.*, 2009]. What remains to be shown is that the lines 11-21 respect the invariant. Suppose any subtree of a general resolution proof where each clause is used at most once (ie. ROR subtree). Max-SAT resolution can be applied to such subtree and then the resulting formula is guaranteed to be equivalent and the last resolvent has weight greater than 0 [Bonet* et al.*, 2007; Larrosa* et al.*, 2008] (otherwise the subtree is not ROR). Let be R the resulting resolvent of the last resolution step after applying resolution to such ROR subtree. Now, the general resolution proof can be transformed by replacing the whole ROR subtree with the resulting clause R (and compensation clauses are properly added to the working formula). Finally, the modifications made in 11-21 provide an equivalent formula and unsatisfiable core for the line 22, hence the termination is guaranteed from [Ansótegui* et al.*, 2009].*

6 Experimental Results

In order to evaluate the use of read-once resolution inside an unsatisfiability-based Max-SAT solver, we extended the C++ MSUNCORE [Manquinho *et al.*, 2009] Max-SAT solver (shortly WMSU1) which interfaces PICOSAT [Biere, 2008] as SAT solver. WMSU1 is similar to the original [Fu and Malik, 2006] solver but it is extended to handle weighted partial Max-SAT [Ansótegui *et al.*, 2009; Manquinho *et al.*, 2009] and it uses the *bitwise encoding* for the cardinality constraints [Prestwich, 2007].

Due to clause minimization techniques in PICOSAT, it returns traces which contain only the original clauses involved in the core, but the order of clauses does not represent the actual resolutions steps. To integrate ROR into the Max-SAT solver, these unordered traces need to be reordered into actual resolution proofs [Beame *et al.*, 2004]. The reordeing is performed with the TRACECHECK tool [1].

In our experiments, we considered all *industrial* and *partial crafted* instances coming from recent Max-SAT Evaluations [2]. We considered all the unweighted Max-SAT industrial and unweighted partial Max-SAT industrial instances from the 2009 Max-SAT Evaluation since in the 2010 edition only a subset was considered. For the remaining categories, we considered the instances in the 2010 Max-SAT Evaluation. All experiments were conducted on a HPC cluster with 50 nodes, each node is a CPU Xeon E5450 3GHz, 32GB RAM and Linux. A time limit of 1200 seconds and 2GB of memory were given for each execution.

Preliminary experiments with ROR transformations showed that traces (input files for TRACECHECK) and actual proofs (output files of TRACECHECK) can be very large for some industrial benchmarks (up to hundreds of megabytes). Besides, for too large proofs, the unrestricted application of resolution could increase substantially the size of the formula. For these reasons, we restricted the application of ROR to traces and proofs up to a maximum file size (2 MB for traces and 20 MB for proofs). We manually set such limits with respect to a small set of industrial instances and such limit was fairly good in general.

Table 1 shows the number of instances solved by each solver within the time limit. The first column shows the name of the benchmark set. The second column presents the total number of instances for each set. The third and fourth columns show the number of solved instances by WMSU1 and the average time for the solved instances of each set, respectively. Similarly, the fifth and sixth columns show the number of solved instances by WMSU1-ROR (ie. WMSU1 with ROR) and the average time for each set, respectively.

We observe that ROR improve the performance of WMSU1 in many of the 31 benchmark sets. In particular, there is an improvement in 14 sets and a slight worsening in 2 sets. Significant improvements are reported in some industrial benchmarks such as Bcp-fir, Bcp-hipp-su, and Bcp-msp but also in crafted benchmarks such as Scheduling, Warehouses and RandomNet. Observe that 211 unsolved instances by WMSU1 become solved by WMSU1-ROR, that is, 19% of the instances are solved by adding ROR.

Figure 1 presents a scatter plot comparing the cpu time in seconds required by WMSU1 and WMSU1-ROR in all the instances that can be solved by at least one of them. The scatter plot shows that WMSU1 requires less time for instances that can be solved by both solvers. But, WMSU1-ROR aborts much fewer instances than WMSU1.

Finally, Table 2 shows the number of solved instances by state-of-the-art Max-SAT solvers including WPM1 [Ansótegui *et al.*, 2009] , MINIM [Heras *et al.*, 2008], SAT4J [Berre and Parrain, 2010], PM2 [Ansótegui *et al.*, 2009] and WPM2 [Ansótegui *et al.*, 2010]. For unweighted Max-SAT, WMSU1-ROR notably improves the performance of WMSU1 but there are more efficient solvers. For weighted Max-SAT, WMSU1 is the most robust solver.

7 Conclusions and Future Work

This paper proposes to apply Max-SAT resolution in an unsatisfiability-based Max-SAT solver when the unsatisfiable core has an associated ROR proof or to subparts of a general proof which are also read-once. Future work includes integrating read-once resolution in other unsatisfiability-based Max-SAT solvers, including WMSU4 [Marques-Silva and Planes, 2008], PM2 [Ansótegui *et al.*, 2009] and WPM2[Ansótegui *et al.*, 2010]. Moreover, we will study under which conditions general proofs can be converted into ROR proofs for Max-SAT.

[1] http://fmv.jku.at/booleforce/

[2] http://www.maxsat.udl.cat/

Benchmark Set	$\#Ins.$	WMSU1	TIME	WMSU1-ROR	TIME
1. ms/ind/CircuitDebugg.	9	9	104.45	9	13.03
2. ms/ind/SeanSafarpour	112	92	38.38	**93**	41.93
3. pms/ind/Bcp-fir	59	50	62.78	**57**	30.51
4. pms/ind/Bcp-hipp-simp	138	131	1.87	**132**	3.65
5. pms/ind/Bcp-hipp-su	38	12	113.44	**17**	138.34
6. pms/ind/Bcp-msp	148	20	20.19	**62**	3.76
7. pms/ind/Bcp-mtg	215	172	11.84	**175**	12.5
8. pms/ind/Bcp-syn	74	30	39.7	**37**	68.88
9. pms/ind/CircuitTraceComp.	4	0	-	0	-
10. pms/ind/Haplotype-Asse.	6	5	9.15	5	52.04
11. pms/ind/Pbo-mqc-nencdr	128	**61**	54.89	59	73.39
12. pms/ind/Pbo-mqc-nlogen.	128	**70**	45.77	65	37.67
13. pms/ind/Pbo-routing	15	15	0.18	15	1.11
14. pms/ind/Protein	12	0	-	1	22.39
15. pms/craft/JobShop	25	0	-	0	-
16. pms/craft/Frb	4	2	204.42	2	148.47
17. pms/craft/MaxCliqueRnd	96	0	0	**14**	11.23
18. pms/craft/MaxCliqueStr	62	4	0.5	**19**	1.82
19. pms/craft/MaxOneRnd	80	39	34.75	**47**	14.49
20. pms/craft/MaxOneStr	60	0	-	2	2.34
21. pms/craft/Kbtree	54	13	9.59	13	27.35
22. pms/craft/MipLib	4	2	3.12	2	0.51
23. wpms/craft/Paths	88	0	-	**5**	2.76
24. wpms/craft/Scheduling	84	0	-	**66**	1.16
25. wpms/craft/Planning	56	29	1.48	**34**	3.7
26. wpms/craft/Warehouses	18	1	0.01	**18**	33.67
27. wpms/craft/MipLib	12	1	0.02	1	0.06
28. wpms/craft/RandomNet	78	15	38.13	**32**	83.34
29. wpms/craft/Spot5	21	3	21.83	**4**	0.2
30. wpms/ind/Upgrade	100	100	50.97	100	66.59
31. wpms/ind/TimeTabling	32	11	199.91	**12**	161.9
Total	1960	887	-	**1098**	-

Table 1: Solved instances by WMSU1 and WMSU1-ROR.

Solver	$\#Ins.$	WMSU1	WMSU1-ROR	WPM1	MINIMAXSAT	SAT4J	PM2+WPM2
Unweighted	1471	727	826	782	969	1037	**1167**
Weighted	489	160	**272**	138	236	212	113

Table 2: Number of solved instances.

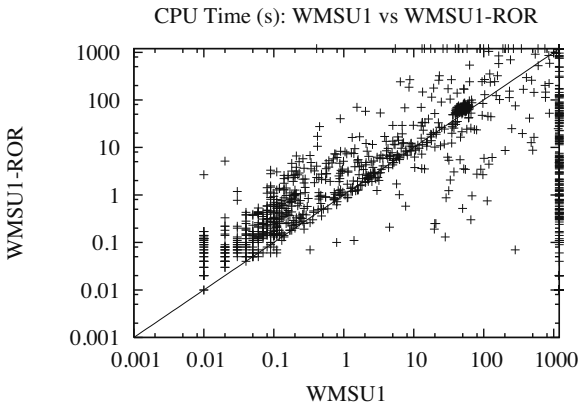

Figure 1: Time scatter plot for WMSU1 vs WMSU1-ROR.

References

[Ansótegui *et al.*, 2009] C. Ansótegui, M. L. Bonet, and J. Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In *SAT*, pages 427–440, July 2009.

[Ansótegui *et al.*, 2010] C. Ansótegui, M. L. Bonet, and J. Levy. A new algorithm for weighted partial MaxSAT. In *AAAI*, 2010.

[Beame *et al.*, 2004] P. Beame, H. A. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *JAIR*, 22:319–351, 2004.

[Berre and Parrain, 2010] D. Le Berre and A. Parrain. The Sat4j library, release 2.2. *JSAT*, 7:59–64, 2010.

[Biere, 2008] Armin Biere. Picosat essentials. *JSAT*, 4(2-4):75–97, 2008.

[Bonet *et al.*, 2007] M. L. Bonet, J. Levy, and F. Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8-9):606–618, 2007.

[Cooper *et al.*, 2010] M. C. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artif. Intell.*, 174(7-8):449–478, 2010.

[de Givry *et al.*, 2003] S. de Givry, J. Larrosa, P. Meseguer, and T. Schiex. Solving Max-SAT as Weighted CSP. In *CP*, pages 363–376, 2003.

[Fu and Malik, 2006] Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *SAT*, pages 252–265, 2006.

[Heras and Larrosa, 2008] F. Heras and J. Larrosa. A max-sat inference-based pre-processing for Max-Clique. In *SAT*, pages 139–152, 2008.

[Heras *et al.*, 2008] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: An efficient weighted Max-SAT solver. *JAIR*, 31:1–32, January 2008.

[Iwama and Miyano, 1995] K. Iwama and E. Miyano. Intractability of read-once resolution. In *Structure in Complexity Theory Conference*, pages 29–36, 1995.

[Larrosa *et al.*, 2008] J. Larrosa, F. Heras, and S. de Givry. A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, 172(2-3):204–233, 2008.

[Li *et al.*, 2007] C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *JAIR*, 30:321–359, 2007.

[Li *et al.*, 2010] C. M. Li, F. Manyà, N. O. Mohamedou, and J. Planes. Resolution-based lower bounds in MaxSAT. *Constraints*, 15(4):456–484, 2010.

[Manquinho *et al.*, 2009] V. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for weighted Boolean optimization. In *SAT*, pages 495–508, July 2009.

[Marques-Silva and Planes, 2008] J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *DATE*, pages 408–413, 2008.

[Pipatsrisawat *et al.*, 2008] K. Pipatsrisawat, A. Palyan, M. Chavira, A. Choi, and A. Darwiche. Solving weighted Max-SAT problems in a reduced search space: A performance analysis. *JSAT*, 4:191–217, 2008.

[Prestwich, 2007] S. D. Prestwich. Variable dependency in local search: Prevention is better than cure. In *SAT*, pages 107–120, 2007.

[Zhang and Malik, 2003] L. Zhang and S. Malik. Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In *DATE*, pages 10880–10885, 2003.