To appear in the *Journal of Experimental & Theoretical Artificial Intelligence*
Vol. 00, No. 00, Month 20XX, 1–26

Improving Linear Search Algorithms with Model-based Approaches for MaxSAT Solving

Ruben Martins*, Vasco Manquinho and Inês Lynce

IST/INESC-ID, University of Lisbon, Portugal
{ruben,vmm,ines}@sat.inesc-id.pt

(*Received 00 Month 20XX; final version received 00 Month 20XX*)

Linear search algorithms have been shown to be particularly effective for solving partial Maximum Satisfiability (MaxSAT) problem instances. These algorithms start by adding a new relaxation variable to each soft clause and solving the resulting formula with a SAT solver. Whenever a model is found, a new constraint on the relaxation variables is added such that models with a greater or equal value are excluded. However, if the problem instance has a large number of relaxation variables, then adding a new constraint over these variables can lead to the exploration of a much larger search space.

This paper proposes new algorithms that use the models found by the SAT solver to partition the relaxation variables. These algorithms add a new constraint on a subset of relaxation variables, thus intensifying the search on that subspace. These algorithms are further enhanced by using incremental approaches where learned clauses are kept between calls to the SAT solver. Experimental results show that model-based algorithms can outperform a traditional linear search algorithm in several problem instances. Moreover, incremental approaches further improve the performance of linear search and model-based algorithms for MaxSAT. Overall, model-based algorithms improve the current state of the art in MaxSAT solving, in particular when dealing with problem instances with a large number of soft clauses.

Keywords: Maximum Satisfiability, Linear Search Algorithms, Model-based Approaches, Incrementality in MaxSAT

1. Introduction

Maximum Satisfiability (MaxSAT) is an optimization version of Boolean Satisfiability (SAT) where the goal is to find an assignment to the problem variables such that the number of satisfied clauses is maximized. Several algorithms have been recently proposed to solve MaxSAT (Ansótegui, Bonet, and Levy, 2009; Heras, Morgado, and Marques-Silva, 2011; Koshimura, Zhang, Fujita, and Hasegawa, 2012; Manquinho, Marques-Silva, and Planes, 2009; Martins, Manquinho, and Lynce, 2012). As a result of these algorithmic advances, MaxSAT solvers are nowadays powerful tools that can be used in many important application domains, such as software package upgrades (Janota, Lynce, Manquinho, and Marques-Silva, 2012), error localization in C code (Jose and Majumdar, 2011), debugging of hardware designs (Chen, Safarpour, Marques-Silva, and Veneris, 2010), bioinformatics (Lin and Khatri, 2012) or course timetabling (Asín and Nieuwenhuis, 2012).

Linear search algorithms for MaxSAT have shown to be effective for solving several classes of industrial MaxSAT problems. These algorithms work by iteratively finding models of a relaxed MaxSAT formula, such that the number of unsatisfied soft clauses

*Corresponding author. Email: ruben@sat.inesc-id.pt of the original MaxSAT formula is minimized. At each SAT call, a cardinality constraint over all relaxation variables is added to the working formula so that it excludes models with a greater or equal value. However, if the number of soft clauses is large, then the number of relaxation variables will be also large. Adding a cardinality constraint over a large number of relaxation variables can lead to the exploration of a much larger search space.

This paper describes algorithms that use the models found by the SAT solver to iteratively increase the set of relaxation variables that are used in the cardinality constraint. A preliminary version of this work was published at the RCRA 2013 workshop (Martins, Manquinho, and Lynce, 2013). In previous work, we have presented model-based algorithms for MaxSAT and have shown that model-based algorithms are able to improve the performance of linear search algorithms for MaxSAT problem instances with a large number of soft clauses. This paper extends previous work in the following directions: (1) different model-based algorithms have been evaluated and a new hybrid model-based algorithm has been proposed, (2) proofs have been included for the correction and termination of model-based algorithms, and (3) experimental results have been extended: (i) the benchmark set has been increased to include more instances, (ii) model-based algorithms are compared against several state-of-the-art MaxSAT solvers.

The organization of the paper is as follows. First, the MaxSAT problem is defined and linear search algorithms for MaxSAT are described. In section 3 different model-based algorithms for MaxSAT are presented. Next, section 4 describes how incremental approaches may be used in model-based algorithms. Section 5 presents an evaluation of the different model-based algorithms. Furthermore, a study on the impact of incremental approaches, as well as a comparison against state-of-the-art MaxSAT solvers is also presented in this section. Finally, the paper concludes and suggests future work.

2. Preliminaries

A Boolean formula in conjunctive normal form (CNF) is defined as a conjunction ($\wedge$) of clauses, where a clause is a disjunction ($\vee$) of literals and a literal is a Boolean variable x or its negation $\bar{x}$. A Boolean variable may be assigned truth values 1 (true) or 0 (false). A positive (negative) literal x ($\bar{x}$) is said to be satisfied if the respective variable is assigned value true (false). A positive (negative) literal x ($\bar{x}$) is said to be unsatisfied if the respective variable is assigned value false (true). A clause is said to be satisfied if at least one of its literals is satisfied. A clause is said to be unsatisfied if all of its literals are unsatisfied. A formula is satisfied if all of its clauses are satisfied. The Boolean Satisfiability (SAT) problem is to decide whether there exists an assignment that makes the formula satisfied. Such assignment is called a *solution* or *model*.

The Maximum Satisfiability (MaxSAT) problem is an optimization version of the SAT problem which consists in finding an assignment that minimizes (maximizes) the number of unsatisfied (satisfied) clauses. In the remainder of the paper, it is assumed that MaxSAT is defined as a minimization problem.

MaxSAT has several variants such as partial MaxSAT, weighted MaxSAT and weighted partial MaxSAT. In the partial MaxSAT problem, some clauses are declared as hard, while the rest are declared as soft. The objective in partial MaxSAT is to find an assignment to the problem variables such that all hard clauses are satisfied, while minimizing the number of unsatisfied soft clauses. Finally, in the weighted versions of MaxSAT, soft clauses can have weights greater than or equal to 1 and the objective is to satisfy all hard clauses while minimizing the total weight of unsatisfied soft clauses. For simplicity, in the remainder of the paper we will consider a MaxSAT formula as being a multiset of clauses.

In general, a MaxSAT formula has both hard and soft clauses. In this paper, we denote a MaxSAT formula as φ such that $\varphi = \varphi_h \cup \varphi_s$, where φ_h denotes the set of hard clauses and φ_s the set of soft clauses.

Definition 2.1 (Solution)**:** *A solution (or model) to a MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$ is a complete assignment ν such that all clauses in φ_h are satisfied.*

Definition 2.2 (Optimal solution)**:** *An optimal solution (or optimal model) to a MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$ is a complete assignment ν such that all clauses in φ_h are satisfied and the number of satisfied soft clauses is maximized, i.e. the number of unsatisfied clauses in φ_s is minimized.*

Definition 2.3 (Unsatisfiable MaxSAT formula)**:** *A MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$ is said to be unsatisfiable if φ_h is unsatisfiable, i.e. there is no solution to the formula.*

Example 2.4: *Consider the following MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$ such that φ_h denotes the set of hard clauses and φ_s the set of soft clauses.*

$$\begin{aligned} \varphi_h &= \{(\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4)\} \\ \varphi_s &= \{(x_1), (x_2 \vee \bar{x}_1), (x_3), (\bar{x}_3 \vee x_1), (x_4)\} \end{aligned} \tag{1}$$

An optimal solution to this formula would be $x_1 = x_2 = x_3 = 0, x_4 = 1$. This assignment satisfies all hard clauses and only two soft clauses are unsatisfied.

Note that $x_1 = x_4 = 0, x_2 = x_3 = 1$ is also a solution, but it is not optimal. Although it satisfies all hard clauses, three soft clauses are unsatisfied.

A generalization of clauses are cardinality constraints. These constraints define that a sum of n literals must be smaller than or equal to a given value k, i.e. $\sum_{i=1}^{n} x_i \leq k$. In other words, a cardinality constraint over n literals ensures that at most k literals can be assigned truth value 1. Although cardinality constraints do not occur in MaxSAT formulations, several algorithms for MaxSAT rely on these constraints (Ansótegui et al., 2009; Heras et al., 2011; Martins et al., 2012). Usually, cardinality constraints are encoded to CNF so that a SAT solver can handle the resulting formula (Asín, Nieuwenhuis, Oliveras, and Rodríguez-Carbonell, 2011; Bailleux and Boufkhad, 2003; Sinz, 2005).

2.1. *MaxSAT Algorithms*

In the last decade, several algorithms for solving MaxSAT have been proposed. They can be mostly categorized into branch and bound algorithms (Argelich, Li, and Manyà, 2007; Heras, Larrosa, and Oliveras, 2008; Li, Manyà, and Planes, 2007; Lin and Su, 2007), linear search algorithms (Koshimura et al., 2012; Le Berre and Parrain, 2010) and unsatisfiability-based algorithms (Ansótegui et al., 2009; Ansótegui, Bonet, and Levy, 2010; Heras et al., 2011; Manquinho et al., 2009).

Since this paper focus on new methods for linear search algorithms, next we provide a detailed description of linear search algorithms for partial MaxSAT. We refer to the literature for branch and bound (Li and Manyà, 2009) and unsatisfiability-based (Morgado, Heras, Liffiton, Planes, and Marques-Silva, 2013) algorithms.

One approach for solving MaxSAT is to make a linear search on the number of unsatisfied soft clauses. Algorithm 1 shows the traditional linear search algorithm for partial MaxSAT (Koshimura et al., 2012; Le Berre and Parrain, 2010). The algorithm starts by relaxing the partial MaxSAT formula. Initially, the working formula φ_W contains all hard clauses φ_h (line 1) and for each soft clause ω, a new variable is created and added to ω (lines 2-3). Next, the relaxed soft clause ω_R is added to the working formula φ_W.

Algorithm 1: Linear search algorithm for partial MaxSAT

```
Input: $\varphi = \varphi_h \cup \varphi_s$
Output: satisfying assignment to $\varphi$ or UNSAT
1  $(V_R, \mathsf{model}, \mu, \varphi_W) \leftarrow (\emptyset, \emptyset, +\infty, \varphi_h)$
2  foreach $\omega \in \varphi_s$ do
3      $V_R \leftarrow V_R \cup \{r\}$                          // r is a new variable
4      $\omega_R \leftarrow \omega \cup \{r\}$                       // relax soft clause
5      $\varphi_W \leftarrow \varphi_W \cup \omega_R$
6  while true do
7      $(st, \nu, \varphi_C) \leftarrow \mathtt{SAT}(\varphi_W)$
8      if $st = \mathsf{SAT}$ then
9          model $\leftarrow \nu$
10         $\mu \leftarrow |\{r \in V_R \mid \nu(r) = 1\}|$      // number of r variables assigned to 1
11         $\varphi_W \leftarrow \varphi_W \cup \{\mathtt{CNF}(\sum_{r \in V_R} r \leq \mu - 1)\}$
12     else
13         if model $= \emptyset$ then
14             return UNSAT                   // the MaxSAT formula is unsatisfiable
15         else
16             return model                     // return satisfying assignment to $\varphi$
```

In the remainder of the paper we denote the relaxation procedure described in lines 2-5 as `relaxFormula(`$\varphi_W, \varphi_s, V_R$`)`.

The goal of this algorithm is to find an assignment to the problem variables such that it minimizes the number of relaxation variables that are assigned truth value 1. In any optimal solution, if a relaxation variable is assigned truth value 1, it corresponds to the unsatisfiability of a soft clause in the original partial MaxSAT formula.

Linear search algorithms work by iteratively calling a SAT solver over a working formula φ_W. A SAT solver returns a triple (st, ν, φ_C), where st denotes the outcome of the solver: satisfiable (SAT) or unsatisfiable (UNSAT). If the solver returns SAT, then the model that satisfies all clauses is stored in ν. On the other hand, if the solver returns UNSAT, then φ_C contains an unsatisfiable subformula.

The working formula φ_W is then given to the SAT solver. While the working formula remains satisfiable, the model ν provided by the SAT solver is stored and we compute the upper bound value μ corresponding to the model ν. This upper bound value corresponds to the number of soft clauses that are unsatisfied in the original partial MaxSAT formula and consequently to the number of relaxation variables that are assigned truth value 1 (line 10). Next, the working formula φ_W is updated by adding a cardinality constraint that excludes models with value greater than or equal to μ. This procedure is repeated until the SAT solver returns unsatisfiable. When this occurs, we have found an optimal solution to φ (line 16). If there was no satisfiable call to the SAT solver, the original formula is unsatisfiable and the algorithm returns UNSAT (line 14).

Example 2.5: *Consider the partial MaxSAT formula φ as defined in Equation (1). Algorithm 1 starts by relaxing the partial MaxSAT formula by introducing a new relaxation variable in each soft clause. As a result, the working formula φ_W will be updated as follows:*

$$\begin{aligned}\varphi_W = \{&(\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4),\\ &(x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5)\}\end{aligned} \tag{2}$$

The set of relaxation variables is $V_R = \{r_1, r_2, r_3, r_4, r_5\}$. Next, φ_W is given to a SAT solver. Suppose that the SAT solver returns the following satisfying assignment ν: $\langle x_2 = x_3 = x_4 = r_1 = r_4 = 0,\ x_1 = r_2 = r_3 = r_5 = 1\rangle$. We store ν in model *and compute the upper bound value μ. Since r_2, r_3 and r_5 were assigned truth value 1, we have found a solution that unsatisfies three soft clauses. Therefore, we can update the upper bound value μ to 3.*

The working formula is now updated with a cardinality constraint over all relaxation variables, such that assignments corresponding to the unsatisfiability of 3 or more soft clauses are excluded. As a result, the working formula is now as follows:

$$\begin{aligned}\varphi_W = \{&(\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4),\\ &(x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5),\\ &CNF(r_1 + r_2 + r_3 + r_4 + r_5 \leq 2)\}\end{aligned} \tag{3}$$

After updating the working formula, the algorithm makes another call to the SAT solver. Consider that the SAT solver returns another satisfying assignment ν: $\langle x_1 = x_2 = x_3 = r_2 = r_4 = r_5 = 0,\ x_4 = r_1 = r_3 = 1\rangle$. Since two relaxation variables were assigned truth value 1, then $\mu = 2$. Next, we update the cardinality constraint of the working formula to $CNF(r_1 + r_2 + r_3 + r_4 + r_5 \leq 1)$.

Finally, we make another call to the SAT solver which returns unsatisfiable. This means that there is no satisfying assignment such that only one of the relaxation variables is assigned truth value 1. Therefore, the optimal solution is given by the previous model that corresponds to unsatisfying two soft clauses.

3. Model-based Approaches for MaxSAT

Traditional linear search algorithms add cardinality constraints over all relaxation variables. If the number of relaxation variables is large, then this can lead to the exploration of a large search space. In contrast with traditional linear search algorithms, model-based algorithms use models given by the SAT solver to iteratively increase the set of relaxation variables that are used in the cardinality constraint.

In this section we present three model-based algorithms for solving partial MaxSAT problems. The first algorithm uses the models given by the SAT solver to iteratively increase the set of relaxation variables that are used in the cardinality constraint. The second algorithm extends the first algorithm by inducing a clear partitioning between the relaxation variables that are used in the cardinality constraint and the ones that are not used. The third algorithm presents a variation of the second algorithm, where we only induce a partial partitioning on the relaxation variables that are not used in the cardinality constraint.

3.1. *Model-based Algorithms*

Model-based algorithms have been first proposed for solving the Minimum Satisfiability (MinSAT) problem (Heras, Morgado, Planes, and Marques-Silva, 2012). The MinSAT problem consists in finding an assignment to the problem variables such that it minimizes the number of satisfied clauses. Even tough this algorithm was not proposed for MaxSAT, the authors mention that it can be adapted to solve MaxSAT problems (Heras et al., 2012).

Algorithm 2 shows our adaptation of this model-based algorithm for solving partial MaxSAT problems. Similarly to the linear search algorithm, the model-based algorithm

Algorithm 2: Model-based algorithm for partial MaxSAT

```
Input: $\varphi = \varphi_h \cup \varphi_s$
Output: satisfying assignment to $\varphi$ or UNSAT
1  $(V_R, V_A, \mathsf{model}, \mu, \mu', \varphi_W) \leftarrow (\emptyset, \emptyset, \emptyset, +\infty, +\infty, \varphi_h)$
2  relaxFormula($\varphi_W, \varphi_s, V_R$)                         // relax soft clauses
3  while true do
4      $(st, \nu, \varphi_C) \leftarrow$ SAT($\varphi_W$)
5      if $st$ = SAT then
6          $V_A \leftarrow V_A \cup \{r \in V_R \mid \nu(r) = 1\}$          // update active r variables
7          $\mu' \leftarrow |\{r \in V_R \mid \nu(r) = 1\}|$
8          if $\mu' < \mu$ then
9              model $\leftarrow \nu$
10             $\mu \leftarrow \mu'$                                // update the upper bound value
11         $\varphi_W \leftarrow \varphi_W \cup \{\mathtt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
12     else
13         if model $= \emptyset$ then
14             return UNSAT
15         else
16             return model
```

starts by relaxing the soft clauses of the MaxSAT formula (line 2). The working formula φ_W is then given to the SAT solver.

The model-based algorithms described in this paper are based on partitioning the relaxation variables in φ_R into two sets of active and inactive relaxation variables.

Definition 3.1 (Active Variables): *A relaxation variable r is considered active if the algorithm has already found at least one model where r is assigned value 1.*

Definition 3.2 (Inactive Variables): *A relaxation variable r is considered inactive if the algorithm did not found any model where r is assigned value 1.*

Note that the two sets of active and inactive relaxation variables are disjoint. Moreover, their union is equivalent to the set of relaxation variables (V_R). At each call of the solver, if the working formula remains satisfiable, then the set of active relaxation variables V_A is updated (line 6).

If the value μ' of the new satisfying assignment is less than the best known upper bound value μ, then μ is updated and the current model is stored. Notice that, since we do not add a cardinality constraint over all relaxation variables, it is possible to obtain models that do not improve on the ones already found. Next, we add a cardinality constraint over the active relaxation variables that excludes models using μ or more active relaxation variables (line 11).

This procedure is repeated until the formula becomes unsatisfiable. When this occurs, we have found an optimal solution to φ (line 16). Similarly to the linear search algorithm, if there was no satisfiable call to the SAT solver, the original formula is unsatisfiable and the algorithm returns UNSAT (line 14).

The main difference between Algorithm 1 and Algorithm 2 is the set of relaxation variables that is being used in the cardinality constraint. The model-based algorithm uses the models to iteratively increase the number of active relaxation variables used in the cardinality constraint. This allows the search to focus on a subset of the relaxation variables. As a side effect, Algorithm 2 may require additional SAT calls, since some of those calls may not improve on the current upper bound but only increase the set of active relaxation variables.

Example 3.3: *Consider again the same partial MaxSAT formula φ as defined in Equa-*

tion (1). Similarly to the linear search algorithm, Algorithm 2 starts by relaxing the partial MaxSAT formula. As a result, the working formula φ_W will be updated as follows:

$$\begin{aligned}\varphi_W = \{&(\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4),\\ &(x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5)\}\end{aligned} \tag{4}$$

Next, φ_W is given to a SAT solver. Suppose it returns the following satisfying assignment ν: $\langle x_2 = x_3 = x_4 = r_1 = r_4 = 0,\ x_1 = r_2 = r_3 = r_5 = 1\rangle$.

We update the set of active relaxation variables V_A by extending it with the relaxation variables that were assigned truth value 1. Therefore, the set of active variables is now updated to $V_A = \{r_2, r_3, r_5\}$. At each SAT call, if the value of the new satisfying assignment μ' is less than the best known upper bound value μ, then we update μ to μ'. Moreover, if μ is updated we also store the corresponding model ν in model. *Since the new satisfying assignment improves our best known upper bound, we update the upper bound μ to 3.*

The working formula is now updated with a cardinality constraint over the active relaxation variables. As a result, the working formula is now as follows:

$$\begin{aligned}\varphi_W = \{&(\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4),\\ &(x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5),\\ &\mathit{CNF}(r_2 + r_3 + r_5 \leq 2)\}\end{aligned} \tag{5}$$

The SAT solver is called again on the working formula. Suppose it returns the satisfying assignment ν: $\langle x_1 = x_2 = x_3 = x_4 = r_2 = r_4 = 0,\ r_1 = r_3 = r_5 = 1\rangle$.

Since r_1 is assigned truth value 1 and does not occur in V_A, we must add r_1 to V_A. The value μ' for the new satisfying assignment is 3, since three relaxation variables are assigned truth value 1. However, μ' does not improve the best known upper bound value μ. Therefore, μ does not change, but the cardinality constraint over V_A must be updated. As a result, the cardinality constraint of the working formula is updated to $\mathit{CNF}(r_1 + r_2 + r_3 + r_5 \leq 2)$.

Consider that in the next call to the SAT solver a new satisfying assignment ν is found: $\langle x_1 = r_2 = r_3 = r_5 = 0,\ x_2 = x_3 = x_4 = r_1 = r_4 = 1\rangle$. Since r_4 is assigned truth value 1 and does not occur in V_A, r_4 is added to V_A. The value μ' of the new satisfying assignment is 2. Hence, μ is updated to 2 and the current model is saved.

The cardinality constraint of the working formula is now $\mathit{CNF}(r_1+r_2+r_3+r_4+r_5 \leq 1)$ and another call to the SAT solver is made. Finally, the SAT solver returns unsatisfiable. The optimal solution was found and is given by the previous model that corresponds to unsatisfying 2 soft clauses.

Proof sketch Algorithm 2

We start by defining the relaxation of a MaxSAT formula. Let φ_R be the relaxation of MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$ such that for each soft clause ω_i in φ_s a new relaxation variable r_i is added. As a result, we have $\varphi_R = \varphi_h \cup \{\omega_i \in \varphi_s : \omega_i \cup \{r_i\}\}$

Lemma 3.4: *φ_R is a satisfiable formula, iff φ_h is satisfiable.*

Proof. If φ_h is unsatisfiable, then φ_R is also unsatisfiable since $\varphi_h \subseteq \varphi_R$. Otherwise, if there is an assignment ν that satisfies φ_h, then one can easily define a satisfying assignment to φ_R extending ν by assigning all relaxation variables to value 1. □

Lemma 3.5: *Given an optimal solution for MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$, a satisfying assignment for φ_R can be built such that the number of relaxation variables assigned value 1 is minimum. The converse is also true.*

Proof. Let ν be an optimal assignment to φ. Let φ_s^U be the set of soft clauses in φ_s that are unsatisfied by ν. Since ν denotes an optimal assignment, the cardinality of φ_s^U is minimum. Notice that ν is not a complete assignment to φ_R. However, one can easily extend ν to build a satisfying assignment to φ_R.

Let ν_R be the extension of ν such that relaxation variables in soft clauses satisfied by ν are assigned value 0 and relaxation variables in soft clauses unsatisfied by ν are assigned value 1. Clearly, ν_R is a solution for φ_R, since clauses not already satisfied by ν become satisfied by assigning to 1 the respective relaxation variables. Moreover, ν_R is also a solution for φ_R where the number of relaxation variables assigned value 1 is minimum.

Suppose that there was another satisfying assignment ν'_R for φ_R with a smaller number of relaxation variables assigned value 1. In that case, one could build an assignment ν' from ν'_R where the relaxation variables are removed. As a result, ν' would be a complete assignment for φ where the number of unsatisfied soft clauses would be smaller than $|\varphi_s^U|$. However, this contradicts ν being an optimal solution for φ. Hence, ν_R is an optimal solution for φ_R.

The proof of the converse is straightforward. □

Lemma 3.6: *When a relaxation variable is considered active it never becomes inactive.*

Proof. The set V_A starts as an empty set (line 1). Whenever a new model for φ_R is found, new relaxation variables may become active (line 6). The set V_A is not modified elsewhere. □

Lemma 3.7: *A relaxation variable r only becomes active if the algorithm finds one model of φ_R where r is assigned value 1.*

Proof. A variable r only becomes active in line 6 of Algorithm 2 when the newly found model has r assigned value 1. □

Theorem 3.8: *Given a partial MaxSAT formula φ, model-based Algorithm 2 terminates and returns the correct solution.*

Proof. Algorithm 2 starts by building φ_R from φ as the initial working formula (line 2). If φ_R is unsatisfiable, then the algorithm returns UNSAT (line 14).

Otherwise, the algorithm iterates through the models of φ_R in order to find an optimal solution. The approach is similar to the classic linear search algorithm for solving MaxSAT. Each time a new model is found, a new constraint is added such that the newly found model is excluded in order to guarantee that the next SAT call returns a different model. Moreover, some models with an equal or higher value are also excluded.[1] Therefore, the algorithm uses a sequence of SAT calls to iterate through valid models of φ_R and the model to be returned is only saved if it improves on the previously found models (lines 8-10).

The constraint added to exclude the current model only uses active relaxation variables (V_A). However, this is sufficient since it excludes all models with greater or equal number of active relaxation variables assigned value 1. No model using fewer relaxation variables

[1]Note that in the classic linear search approach, the constraint added in each iteration excludes *a*ll models with an equal or higher value.

assigned value 1 is excluded. Note that since V_A only increases, its size is always larger or equal to the value of the best solution found μ.

Considering that models are being excluded in each iteration of the algorithm, the working formula eventually becomes unsatisfiable. When this occurs, the algorithm terminates and returns the best model found (an optimal solution to φ). □

3.2. *Model-based Algorithms with Disabled Variables*

The model-based algorithm presented in Algorithm 2 does not impose restrictions on inactive relaxation variables. This may lead to calls to the SAT solver returning new satisfying assignments that do not improve on the best solution previously found. However, if we impose restrictions over the inactive relaxation variables, then we can further reduce the search space.

Algorithm 3 shows the model-based algorithm with disabled relaxation variables. This algorithm extends the model-based algorithm by disabling inactive relaxation variables. The goal is to make an optimistic assumption that inactive relaxation variables can be assigned value 0. If this is not the case, the working formula becomes unsatisfiable. However, in case the formula is unsatisfiable, current SAT solvers are able to provide certificates of unsatisfiability. In our algorithm, these certificates of unsatisfiability are then used to extend the set of active relaxation variables until the working formula becomes satisfiable.

Definition 3.9 (Disabled variables)**:** *A relaxation variable r is considered disabled if it is inactive and the unit clause $(\bar{r})$ is added to the working formula to disable the assignment of truth value 1 to r.*

As with previous algorithms, Algorithm 3 also starts by relaxing the MaxSAT formula (line 2). However, notice that the initial working formula φ_{init} is stored for latter use (line 3).

Next, the SAT solver is called in order to find the first model for the formula. If the initial formula is unsatisfiable, then the algorithm terminates and returns UNSAT. Otherwise, all relaxation variables that are assigned truth value 1 in the first model are considered active, while the others are considered disabled. Unit clauses are added to enforce the assignment of value 0 to disabled relaxation variables in the next model. Moreover, a cardinality constraint over the active relaxation variables is also added (line 11).

After the initial split of relaxation variables, the algorithm iterates over the working formula. Whenever the working formula is satisfiable, a new better solution has been found and we update the cardinality constraint as in the linear search algorithm (line 17). However, if the SAT solver returns unsatisfiable, then we need to analyze the unsatisfiable subformula φ_C. If φ_C contains unit clauses with disabled relaxation variables, then those relaxation variables are added to V_A (line 22) and removed from V_D (line 23). Note that each unsatisfiable iteration enumerates disjoint unsatisfiable subformulas. Therefore, the lower bound value (λ) can be increased at each unsatisfiable call of the SAT solver (line 24). Afterwards, the working formula is rebuilt from φ_{init} together with the cardinality constraint over the updated set of active relaxation variables and the unit clauses of the updated set of disabled relaxation variables.

The algorithm proceeds until an optimal solution is found. The algorithm can terminate due to two reasons: if the unsatisfiable subformula does not depend on any disabled relaxation variables (line 19) or if the lower bound value is the same as the upper bound value (lines 26).

A similar approach to the one shown in Algorithm 3 has been previously pre-

Algorithm 3: Model-based algorithm with disabled variables for partial MaxSAT

```
Input: $\varphi = \varphi_h \cup \varphi_s$
Output: satisfying assignment to $\varphi$ or UNSAT
1  $(V_R, V_A, V_D, \mathsf{model}, \lambda, \mu, \varphi_W) \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, 0, +\infty, \varphi_h)$
2  relaxFormula($\varphi_W, \varphi_s, V_R$)                          // relax soft clauses
3  $\varphi_{init} \leftarrow \varphi_W$                            // store the original relaxed formula
4  $(st, \nu, \varphi_C) \leftarrow$ SAT($\varphi_W$)                      // get the first model
5  if $st$ = SAT then
6      $V_A \leftarrow \{r \in V_R \mid \nu(r) = 1\}$                    // set of active variables
7      $V_D \leftarrow V_R \setminus V_A$                              // set of disabled variables
8      $(\mathsf{model}, \mu) \leftarrow (\nu, |V_A|)$
9  else
10     return UNSAT
11 $\varphi_W \leftarrow \varphi_W \cup \{(\bar{r}) \mid r \in V_D\} \cup \{\mathtt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
12 while true do
13     $(st, \nu, \varphi_C) \leftarrow$ SAT($\varphi_W$)
14     if $st$ = SAT then
15         model $\leftarrow \nu$
16         $\mu \leftarrow |\{r \in V_A \mid \nu(r) = 1\}|$
17         $\varphi_W \leftarrow \varphi_W \cup \{\mathtt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
18     else
19         if $\varphi_C \cap \{(\bar{r}) \mid r \in V_D\} = \emptyset$ then
20             return model
21         else
22             $V_A \leftarrow V_A \cup \{r \mid (\bar{r}) \in \varphi_C \cap r \in V_D\}$     // update active variables
23             $V_D \leftarrow V_D \setminus V_A$                          // update disabled variables
24             $\lambda \leftarrow \lambda + 1$
25             $\varphi_W \leftarrow \varphi_{init} \cup \{(\bar{r}) \mid r \in V_D\} \cup \{\mathtt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
26     if $\lambda = \mu$ then
27         return model
```

sented (Marques-Silva and Planes, 2008). In their approach, the authors propose to start with all relaxation variables disabled, i.e. $V_D = V_R$ and $V_A = \emptyset$. Additionally, at each unsatisfiable iteration, they add an extra clause over the disabled relaxation variables that appear in the unsatisfiable subformula. Even though this extra clause is not necessary, it guarantees that one of the these relaxation variables is assigned truth value 1 in the next iteration. Moreover, their approach does not relax all soft clauses at the beginning of the search but instead only relax them when they appear in an unsatisfiable subformula. However, this is equivalent to relax all soft clauses and then to disable the relaxation variables through unit clauses.

In contrast to the previous approach, we propose a different partitioning strategy based on finding an initial model. The relaxation variables that are assigned truth value 1 in that model are put in the set of active variables. On the other hand, the relaxation variables that are assigned truth value 0 in that model are put in the set of disabled variables.

Example 3.10: *Consider again the same partial MaxSAT formula φ as defined in Equation (1). Similarly to the previous algorithms, the initial working formula φ_W is the relaxed partial MaxSAT formula.*

$$\begin{aligned}\varphi_W = \{&(\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4),\\ &(x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5)\}\end{aligned} \tag{6}$$

Suppose we are using a partitioning strategy based on finding an initial model. The SAT *solver is called on the relaxed formula and returns the following satisfying assignment $\nu : \langle x_2 = x_3 = x_4 = r_1 = r_4 = 0,\ x_1 = r_2 = r_3 = r_5 = 1\rangle$.*

We update the set of active relaxation variables to $V_A = \{r_2, r_3, r_5\}$ and the set of disabled relaxation variables to $V_D = \{r_1, r_4\}$. Moreover, the current model is saved and μ is set to 3.

The working formula is now updated with a cardinality constraint over the active relaxation variables. Moreover, we also add to the working formula the unit clauses that do not allow the assignment of truth value 1 to the disabled relaxation variables. As a result, the working formula is now as follows:

$$\begin{aligned}\varphi_W = \{\ &(\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4),\\ &(x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5),\\ &(\bar{r}_1), (\bar{r}_4), \mathit{CNF}(r_2 + r_3 + r_5 \leq 2)\}\end{aligned} \tag{7}$$

After updating the working formula, the algorithm makes another call to the SAT solver. Consider that the SAT solver returns unsatisfiable and φ_C contains $(\bar{r}_1)$. We update the set of active relaxation variables to $V_A = \{r_1, r_2, r_3, r_5\}$ and the set of disabled relaxation variables to $V_D = \{r_4\}$. Next, we rebuild the working formula from the initial relaxed MaxSAT formula (φ_{init}), together with the cardinality constraint over the active relaxation variables and the unit clauses from the disabled relaxation variables. As a result, the working formula is now as follows:

$$\begin{aligned}\varphi_W = \{\ &(\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4),\\ &(x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5),\\ &(\bar{r}_4), \mathit{CNF}(r_1 + r_2 + r_3 + r_5 \leq 2)\}\end{aligned} \tag{8}$$

Next, the SAT solver is called. Suppose it returns a satisfying assignment ν: $\langle x_1 = x_2 = x_3 = r_2 = r_4 = r_5 = 0,\ x_4 = r_1 = r_3 = 1\rangle$. A new better solution has been found and μ is set to 2. The cardinality constraint of the working formula is updated to $\mathit{CNF}(r_1 + r_2 + r_3 + r_5 \leq 1)$.

In the next call, the formula is unsatisfiable and φ_C contains $(\bar{r}_4)$. The sets of active and disabled relaxation variables are updated to $V_A = \{r_1, r_2, r_3, r_4, r_5\}$ and $V_D = \emptyset$. The working formula is rebuilt as previously described.

Finally, the next call to the SAT solver returns unsatisfiable and φ_C does not contain any unit clauses from the disabled relaxation variables. Therefore, an optimal solution was already found and is given by the previous model that corresponds to unsatisfying 2 soft clauses.

Proof sketch Algorithm 3

Lemma 3.11: *If there is an optimal solution for φ in which soft clause ω_i is satisfied, then r_i can be safely assigned value 0 when solving φ_R.*

Proof. The optimal model ν for φ that satisfies soft clause ω_i can be trivially extended to ν_R such that ν_R satisfies φ_R and $r_i = 0$ in ν_R. □

Corollary 3.12: *If there is an optimal solution for φ in which the set of soft clauses φ'_S is satisfied, then all relaxation variables for clauses in φ'_S can be safely assigned value 0 when solving φ_R.*

Proof. Follows from the previous proof. □

In Algorithm 3 the working formula φ_W is defined as $\varphi_W = \varphi_R \cup \{(\neg r) \mid r \in V_D\} \cup \{\texttt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$, where V_A denotes the set of active variables and $V_D = V_R \setminus V_A$ denotes the set of *disabled* variables. Moreover, μ denotes the number of relaxation variables assigned value 1 in the best model already found.

Lemma 3.13: *Suppose that a model ν using μ relaxation variables assigned to 1 is the best solution found so far. If φ_W is satisfiable, then a model ν' of φ_W improves on ν.*

Proof. Note that in this algorithm, *disabled* variables correspond to all relaxation variables that are not active variables. Since all *disabled* variables are assigned value 0 in φ_W and at most $\mu - 1$ active variables can be assigned value 1, then any model ν' of φ_W will have at most $\mu - 1$ relaxation variables assigned value 1. As a result, one can safely replace ν by ν' as the best solution found so far. □

Lemma 3.14: *Suppose that a model ν using μ relaxation variables assigned to 1 is the best solution found so far. If φ_W is unsatisfiable and the unsatisfiable core returned by the SAT solver does not depend on disabled relaxation variables V_D, then model ν is an optimal solution.*

Proof. φ_W contains unit clauses such that all *disabled* relaxation variables V_D must be assigned value 0. If the unsatisfiable core returned by the SAT solver does not contain any of those unit clauses, then $\varphi_R \cup \{\texttt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$ is unsatisfiable. As a result, at least μ relaxation variables must be assigned value 1 for φ_R to be satisfiable. Considering that model ν uses μ relaxation variables assigned to 1, then ν is an optimal solution. □

Lemma 3.15: *If φ_W is unsatisfiable and the unsatisfiable core φ_C returned by the SAT solver depends on a subset V_D' of disabled relaxation variables, then at least one of the corresponding soft clauses is unsatisfied in an optimal solution. Otherwise, an optimal solution was already found.*

Proof. Suppose there is an optimal solution where all V_D' variables can be assigned value 0. In that case, the unsatisfiability of φ_W would be due to the unsatisfiability of $\varphi_R \cup \{\texttt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$. In that case, μ corresponds to the value of the optimal solution that was previously found (lemma 3.14).

Otherwise, at least one of the variables in V_D' must be assigned value 1. As a result, at least one of the corresponding soft clauses will be unsatisfied in an optimal solution. □

Lemma 3.16: *Let λ denote a lower bound on the number of unsatisfied clauses in an optimal solution. Each time that φ_W is unsatisfiable, λ can be increased by 1.*

Proof. If the unsatisfiability of φ_W does not depend on the unit clauses on *disabled* relaxation variables, then an optimal solution was already found and the algorithm terminates (lemma 3.14).

Otherwise, the unsatisfiable core φ_C returned by the SAT solver contains a subset of unit clauses on *disabled* relaxation variables. Note that these variables become active and the corresponding unit clauses are removed from φ_W. Hence, in each iteration that φ_W is unsatisfiable, the subset of unit clauses on *disabled* relaxation variables is disjoint from the previous ones. Therefore, in each iteration, a disjoint subset of soft clauses is identified such that at least one will be unsatisfied in an optimal solution. As a result, the lower bound λ can be safely increased by 1 when φ_W is unsatisfiable. □

Theorem 3.17: *Given a partial MaxSAT formula φ, model-based Algorithm 3 terminates and returns the correct solution.*

Proof. Algorithm 3 starts by building φ_R from φ as the initial working formula (line 2). Next, a SAT call is made to obtain a first model for φ_R. However, if φ_R is unsatisfiable, then the algorithm terminates (line 10) and returns UNSAT (lemma 3.4). Otherwise, relaxation variables are partitioned into active (V_A) and *disabled* (V_D).

Next, the algorithm iterates through the models of working formula φ_W in order to find an optimal solution. If a new solution is found, the model is saved and the working formula is updated (lines 15-17). Notice that the newly found model always improves on the previous one (lemma 3.13).

In working formula φ_W, one assumes that *disabled* relaxation variables can be assigned value 0 and corresponding unit clauses are added. As a result, φ_W may become unsatisfiable. Whenever this occurs, the SAT solver provides an unsatisfiable subformula φ_C of φ_W. If φ_C does not contain any unit clause that forces *disabled* relaxation variables to be assigned value 0, then an optimal model was already found and the algorithm terminates (lemma 3.14).

Otherwise, the assumption that all *disabled* relaxation variables can be assigned value 0 is not correct. The unsatisfiable subformula φ_C provides information on a subset of *disabled* relaxation variables where at least one of them must be assigned value 1. Therefore, these variables are moved from V_D to V_A and the working formula is rebuilt accordingly (lines 22-25). Moreover, the lower bound λ is also updated (lemma 3.16). Eventually, all *disabled* relaxation variables may become active and φ_C does not contain any unit clause on *disabled* relaxation variables. As a result, the algorithm terminates and an optimal solution is returned.

Finally, if the lower bound value λ is equal to μ, then we are sure that a better solution does not exist and the algorithm terminates (lines 26-27). □

3.3. *Hybrid Model-based Algorithms*

The model-based algorithm presented in Algorithm 3 uses the unsatisfiable subformulas returned by the SAT solver to move relaxation variables from disabled to active . However, the unsatisfiable subformulas returned by the SAT solver may be unnecessarily large. This may lead to many relaxation variables becoming active which can have a detrimental effect on the solver.

Algorithm 4 shows a hybrid model-based algorithm that combines model-based algorithms 2 and 3. The motivation is to change the state of disabled relaxation variables to inactive and only afterwards to active . As a result, we are reducing the number of relaxation variables that become active at each iteration.

The differences from Algorithm 3 are highlighted in Algorithm 4. When the formula is unsatisfiable and the unsatisfiable subformula φ_C contains unit clauses with disabled relaxation variables, then these variables are put in the set of inactive relaxation variables (line 25) and removed from the set of disabled relaxation variables (line 26).

Notice that the cardinality constraint is still over the active relaxation variables. As a side effect, Algorithm 4 may require additional SAT calls, since some of those calls may not improve the current upper bound value but only increase the set of active relaxation variables. This is similar to the scenario that occurs in Algorithm 2. The set of active relaxation variables is only updated when a model is found. All inactive relaxation variables that are assigned truth value 1, are added to the set of active variables (line 15) and removed from the set of inactive variables (line 16). Moreover, similarly to Algorithm 2 the upper bound value is only updated (line 19) if the current model improves on the best solution already found.

Example 3.18: *Consider again the same partial MaxSAT formula φ as defined in Equa-*

Algorithm 4: Hybrid model-based algorithm for partial MaxSAT

```
Input: $\varphi = \varphi_h \cup \varphi_s$
Output: satisfying assignment to $\varphi$ or UNSAT
1  $(V_R, V_A, V_D, V_I, \mathsf{model}, \lambda, \mu, \varphi_W) \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, 0, +\infty, \varphi_h)$
2  relaxFormula($\varphi_W, \varphi_s, V_R$)                          // relax soft clauses
3  $\varphi_{init} \leftarrow \varphi_W$                              // store the original relaxed formula
4  $(st, \nu, \varphi_C) \leftarrow \mathtt{SAT}(\varphi_W)$          // get the first model
5  if $st =$ SAT then
6      $V_A \leftarrow \{r \in V_R \mid \nu(r) = 1\}$                 // set of active variables
7      $V_D \leftarrow \{r \in V_R \mid \nu(r) = 0\}$                 // set of disabled variables
8      $(\mathsf{model}, \mu) \leftarrow (\nu, |V_A|)$
9  else
10     return UNSAT
11 $\varphi_W \leftarrow \varphi_W \cup \{(\neg r) \mid r \in V_D\} \cup \{\mathtt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
12 while true do
13     $(st, \nu, \varphi_C) \leftarrow \mathtt{SAT}(\varphi_W)$
14     if $st =$ SAT then
15         $V_A \leftarrow V_A \cup \{r \in V_I \mid \nu(r) = 1\}$        // update active variables
16         $V_I \leftarrow V_I \setminus V_A$                            // update inactive variables
17         $\mu' \leftarrow |\{r \in V_R \mid \nu(r) = 1\}|$
18         if $\mu' < \mu$ then
19             $(\mathsf{model}, \mu) \leftarrow (\nu, \mu')$            // update the upper bound value
20         $\varphi_W \leftarrow \varphi_W \cup \{\mathtt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
21     else
22         if $\varphi_C \cap \{(\neg r) \mid r \in V_D\} = \emptyset$ then
23             return model
24         else
25             $V_I \leftarrow V_I \cup \{r \mid (\neg r) \in \varphi_C \cap r \in V_D\}$   // update inactive variables
26             $V_D \leftarrow V_D \setminus \{V_A \cup V_I\}$                          // update disabled variables
27             $\lambda \leftarrow \lambda + 1$
28             $\varphi_W \leftarrow \varphi_{init} \cup \{(\neg r) \mid r \in V_D\} \cup \{\mathtt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
29     if $\lambda = \mu$ then
30         return model
```

tion (1). Similarly to the previous algorithms, the initial working formula φ_W is the relaxed partial MaxSAT formula.

$$\begin{aligned}\varphi_W = \{&(\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4),\\ &(x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5)\}\end{aligned} \tag{9}$$

Suppose we are using a partitioning strategy based on finding an initial model. The SAT solver is called on the relaxed formula and returns the following satisfying assignment $\nu : \langle x_2 = x_3 = x_4 = r_1 = r_4 = 0,\ x_1 = r_2 = r_3 = r_5 = 1\rangle$.

We update the set of active relaxation variables to $V_A = \{r_2, r_3, r_5\}$ and the set of disabled relaxation variables to $V_D = \{r_1, r_4\}$. Moreover, the current model is saved and μ is set to 3.

The working formula is now updated with a cardinality constraint over the active relaxation variables. Moreover, we also add to the working formula the unit clauses that do not allow the assignment of truth value 1 to the disabled relaxation variables. As a result, the working formula is now as follows:

$$\begin{aligned}\varphi_W = \{\ & (\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4), \\ & (x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5), \\ & (\bar{r}_1), (\bar{r}_4), \mathit{CNF}(r_2 + r_3 + r_5 \leq 2)\}\end{aligned} \tag{10}$$

After updating the working formula, the algorithm makes another call to the SAT solver. Consider that the SAT solver returns unsatisfiable and φ_C contains $(\bar{r}_1)$. We update the set of inactive relaxation variables to $V_I = \{r_1\}$ and the set of disabled relaxation variables to $V_D = \{r_4\}$. Next, we rebuild the working formula from the initial relaxed MaxSAT formula (φ_{init}), together with the cardinality constraint over the active relaxation variables and the unit clauses from the inactive relaxation variables. As a result, the working formula is now as follows:

$$\begin{aligned}\varphi_W = \{\ & (\bar{x}_2 \vee \bar{x}_1), (x_2 \vee \bar{x}_3), (\bar{x}_1 \vee \bar{x}_4), \\ & (x_1 \vee r_1), (x_2 \vee \bar{x}_1 \vee r_2), (x_3 \vee r_3), (\bar{x}_3 \vee x_1 \vee r_4), (x_4 \vee r_5), \\ & (\bar{r}_4), \mathit{CNF}(r_2 + r_3 + r_5 \leq 2)\}\end{aligned} \tag{11}$$

Next, the SAT solver is called. Suppose it returns a satisfying assignment ν: $\langle x_1 = x_2 = x_3 = x_4 = r_2 = r_4 = 0,\ r_1 = r_3 = r_5 = 1\rangle$. Since r_1 is assigned truth value 1 and does not occur in V_A, we must add r_1 to V_A and remove it from V_I. The value μ' for the new satisfying assignment is 3, since three relaxation variables are assigned truth value 1. However, μ' does not improve the best known upper bound value μ. Therefore, μ does not change, but the cardinality constraint over V_A must be updated. As a result, the cardinality constraint of the working formula is updated to $\mathit{CNF}(r_1 + r_2 + r_3 + r_5 \leq 2)$.

In the next call, the SAT solver returns unsatisfiable and φ_C contains $(\bar{r}_4)$. The sets of disabled and inactive relaxation variables are updated to $V_D = \emptyset$ and $V_I = \{r_4\}$ and the working formula is rebuilt as previously described.

Consider that in the next call to the SAT solver a new satisfying assignment ν is found: $\langle x_1 = r_2 = r_3 = r_5 = 0,\ x_2 = x_3 = x_4 = r_1 = r_4 = 1\rangle$. Since r_4 is assigned truth value 1 and does not occur in V_A, r_4 is added to V_A and remove from V_I. The value μ' of the new satisfying assignment is 2. Hence, μ is updated to 2 and the current model is saved.

The cardinality constraint of the working formula is now $\mathit{CNF}(r_1 + r_2 + r_3 + r_4 + r_5 \leq 1)$ and another call to the SAT solver is made. Finally, the next call to the SAT solver returns unsatisfiable and φ_C does not contain any unit clauses from the disabled relaxation variables. Hence, the optimal solution was found and is given by the previous model that corresponds to unsatisfying 2 soft clauses.

4. Incrementality in MaxSAT

Linear search algorithms for MaxSAT work by making a sequence of calls to the SAT solver. After each call to the SAT solver, we can rebuild the working formula by discarding all learned clauses from the previous iteration.

Definition 4.1 (Learned Clause (Marques-Silva and Sakallah, 1996; Zhang, Madigan, Moskewicz, and Malik, 2001))**:** *After detecting a conflict, i.e. a sequence of assignments that make a clause unsatisfiable, a new clause is learned to prevent the same conflict from occurring in the subsequent search. The new clause is called a learned clause and results from the analysis of the implication graph which represents the dependencies between assignments.*

Discarding all learned clauses between calls to the SAT solver is denoted by *non-incremental MaxSAT solving*. However, some of those learned clauses may be kept between iterations. Keeping learned clauses between calls to the SAT solver is denoted by *incremental MaxSAT solving*.

Incremental MaxSAT solving is essential for boosting the performance of MaxSAT solvers. However, to the best of our knowledge, no study has been made to evaluate the impact of incrementality on linear search algorithms for MaxSAT. Incrementality in MaxSAT is tightly related to the result of each call of the SAT solver and to the set of active relaxation variables. The following scenarios can occur when considering incrementality in linear search algorithms: (i) the SAT solver returns satisfiable and the set of active relaxation variables remains the same, (ii) the SAT solver returns satisfiable and the set of active relaxation variables is updated, and (iii) the SAT solver returns unsatisfiable.

Updating the cardinality constraint

Consider the first scenario where the SAT solver returns satisfiable and the set of active relaxation variables does not change. If this occurs, it means a new upper bound value has been found. Hence, we need to update the right-hand side of the cardinality constraint on the active variables. However, for several cardinality encodings, when updating the right-hand side we do not need to re-encode the cardinality constraint. Instead, we can update the encoding by setting some specific literals to false. This update procedure is denoted by *incremental strengthening* (Asín et al., 2011). In this scenario, *all* learned clauses from the previous SAT call are sound and may be reused in the next call to the SAT solver. In the remainder of the paper, this incrementality case is denoted by *Update* since we only need to update the cardinality encoding.

Updating the set of active relaxation variables

Assume that the SAT solver returns satisfiable but the set of active relaxation variables has been updated. In this case, incremental strengthening cannot be used since the set of relaxation variables being used in the cardinality constraint has changed. Therefore, we must rebuilt the working formula and re-encode the cardinality constraint. When rebuilding the formula we cannot keep all learned clauses that were created during the previous SAT call.

Definition 4.2 (Unsafe learned clauses)**:** *A learned clause is declared unsafe if it was created by using at least one of the following clauses: (i) clauses from the encoding of the cardinality constraint, (ii) unit clauses from the disabled relaxation variables, or (iii) previously declared unsafe learned clauses.*

Learned clauses that are unsafe must be discarded since they may not be valid in the context of the current working formula. On the other hand, learned clauses that are not marked as unsafe may be kept since they remain valid for the next call to the SAT solver. In the remainder of the paper, this incrementality case is denoted by *Re-Encode* since we need to re-encode the cardinality encoding.

Unsatisfiable call to the SAT solver

Finally, consider the scenario where the SAT solver returns unsatisfiable. In this scenario we proceed as in the *Re-Encode* case. The working formula is rebuilt and the cardinality constraint is re-encoded. Moreover, only learned clauses that are not marked as unsafe can be kept for the next call to the SAT solver. In the remainder of the paper, this

	Incrementality		
Algorithm	*Update*	*Re-Encode*	*Unsat*
`Linear`	✓		
`Model`	✓	✓	
`Disabled`	✓		✓
`Hybrid`	✓	✓	✓

Table 1. Incrementality in linear search algorithms

incrementality case is denoted by *Unsat* since it occurs due to an unsatisfiable call of the SAT solver.

4.1. *Incrementality in Linear Search algorithms*

Table 1 shows which incrementality cases (*Update*, *Re-Encode*, *Unsat*) occur in the linear search algorithms that have been presented in the previous sections, namely:

- `Linear`: uses the traditional linear search algorithm presented in Algorithm 1.
- `Model`: uses the model-based algorithm presented in Algorithm 2.
- `Disabled`: uses the model-based algorithm with disabled relaxation variables presented in Algorithm 3.
- `Hybrid`: uses the hybrid model-based algorithm presented in Algorithm 4.

The `Linear` algorithm only needs to encode the cardinality constraint once (when the first model is found). In the next iterations, the cardinality constraint is always updated through incremental strengthening. Therefore, *all* learned clauses may be kept between calls to the SAT solver.

In the `Model` algorithm occur incremental cases of *Update* and *Re-Encode*. If the set of active relaxation variables does not change between calls to the SAT solver, then we can reuse all learned clauses as in the `Linear` algorithm. However, if the set of active relaxation variables changes, then we need to rebuild the working formula and re-encode the cardinality constraint. While rebuilding the formula, we keep learned clauses that are not marked as unsafe.

In the `Disabled` algorithm occur incremental cases of *Update* and *Unsat*. If the SAT solver returns satisfiable, then we can reuse all learned clauses like in the `Linear` algorithm. Notice that, in this algorithm, the set of active relaxation variables is only updated when the solver returns unsatisfiable. Therefore, in the former case we need to rebuild the working formula and re-encode the cardinality constraint. When the formula is rebuilt, we keep learned clauses that are not marked as unsafe.

Finally, in the `Hybrid` algorithm occur all possible incremental cases. In this algorithm, the set of active relaxation variables is updated like in `Model`. Hence, if the set of active relaxation variables does not change between calls to the SAT solver, then we can reuse all learned clauses. Otherwise, we need to rebuild the working formula and re-encode the cardinality constraint. Moreover, if the solver returns unsatisfiable then we also need to rebuild the working formula and re-encode the cardinality constraint. Learned clauses that are not marked as unsafe are kept when rebuilding the formula.

5. Experimental Results

In this section we evaluate the performance of model-based algorithms in a selection of benchmark instance sets. All experiments were run on two AMD Opteron 6276 processors

Benchmark	#Instances	Avg. #Soft
ms_industrial	121	1,357,041
pms_industrial	972	1,120
close_solution	486	166,105

Table 2. Average number of soft clauses per instance

(2.3 GHz) running Linux Fedora Core 18 with a timeout of 1,800 seconds and a memory limit of 16 GB.

In what follows, we compare the solvers that implement the different algorithms presented in this paper, namely: `Linear`, `Model`, `Disabled[All]`, `Disabled[Model]`, `Hybrid[All]`, and `Hybrid[Model]`.

`Linear` denotes the solver that implements the linear search algorithm described in section 2.1. `Model`, `Disabled` and `Hybrid` denote the solvers that implement the model-based algorithms described in section 3. Moreover, `[All]` and `[Model]` denote the partitioning strategy for the initial set of active and disabled relaxation variables. `[All]` corresponds to the partitioning strategy where all relaxation variables are initially declared as disabled, whereas `[Model]` uses the first model to partition the relaxation variables between active and disabled.

Our solvers were implemented using the AUSTIN framework[2]. AUSTIN is based on `Glucose 2.3` (Audemard and Simon, 2009) and uses the cardinality networks encoding (Asín et al., 2011) to encode cardinality constraints into CNF. The unsatisfiable subformulas for the `Disabled` and `Hybrid` solvers were extracted using an assumption-based approach (Asín, Nieuwenhuis, Oliveras, and Rodríguez-Carbonell, 2010). Additionally, the extraction of unsatisfiable subformulas has been further enhanced by considering the improvements proposed for `Glucose` when using assumptions (Audemard, Lagniez, and Simon, 2013).

5.1. *Benchmarks*

The evaluation was performed on 121 industrial MaxSAT instances and on 972 partial MaxSAT instances of the MaxSAT evaluations of 2009-2012[3].

Additionally, we have also considered 486 instances coming from the *close solution* problem[4] (Abío and Stuckey, 2012; Abío, Deters, Nieuwenhuis, and Stuckey, 2011). The close solution problem is as follows. Consider a SAT formula φ and a model ν. If some new clauses are added to φ, then the goal is to find a new model ν' that is as similar as possible to ν. Similarity is measured as the number of assignments to the variables that are common to ν and ν'. This problem can be encoded into partial MaxSAT and may contain a large number of soft clauses.

Table 2 shows the total number of instances for each benchmark set and the average number of soft clauses per instance for each benchmark set.

Notice that the MaxSAT industrial benchmarks (ms_industrial) have a very large number of soft clauses. On average, each instance has over 1 million soft clauses. Moreover, note that these instances do not have hard clauses. Therefore, they are only composed of soft clauses and in some cases solvers can run out of memory when handling such huge MaxSAT formulas. If each soft clause is relaxed by adding a new relaxation variable,

[2]The AUSTIN framework (**a**lgorithms **u**sing **st**ructural **in**formation for MaxSAT solving) will be publicly available for download at `http://sat.inesc-id.pt/austin/`
[3]Available at `http://maxsat.ia.udl.cat/`
[4]We would like to thank Ignasi Abío for his assistance with these benchmarks.

then encoding a cardinality constraint into CNF using a huge set of relaxation variables can easily lead to memory problems.

In contrast, partial industrial MaxSAT benchmarks (pms_industrial) have both hard and soft clauses. Each instance has only a small number of soft clauses, being on average around 1,000 soft clauses per instance. Finally, the number of soft clauses in the close solution problem (close_solution) is not as large as in the industrial MaxSAT benchmarks but it is much larger than in the partial industrial benchmarks. On average, each close solution problem instance has over 160,000 soft clauses.

5.2. *Model-based Algorithms*

Table 3 shows the number of instances solved by each solver for the different benchmarks.

For the partial industrial MaxSAT benchmarks, model-based algorithms do not perform as well as the traditional linear search algorithm. Since the average number of soft clauses is small, model-based algorithms tend to use the majority of the relaxation variables in the cardinality constraint to find the optimal solution. Therefore, starting with a small subset of the relaxation variables and iteratively increasing this set represents an overhead that deteriorates the performance of the solver. Even though model-based algorithms have to deal with an overhead when solving instances with a small number of soft clauses, the model-based algorithms that use the `[All]` strategy for partitioning the initial set of disabled relaxation variables exhibit a performance which is comparable to the traditional linear search algorithm. For example, both `Disabled[All]` and `Hybrid[All]` solve only 1 less instance than `Linear`.

For the industrial MaxSAT benchmarks, `Disabled` and `Hybrid` clearly outperform the remaining solvers. These solvers can solve instances in this benchmark set by using a small fraction of the relaxation variables. `Disabled` solvers exhibit the best performance for this benchmark set since the relaxation variables that do not belong to the cardinality constraint are disabled. In contrast, the `Hybrid` solvers disable only some of the relaxation variables that do not belong to the cardinality constraint. For a very large number of relaxation variables, disabling all relaxation variables that do not belong to the cardinality constraint leads to a better performance of the solver. This may explain why `Disable` solvers are able to outperform `Hybrid` solvers for this set of benchmarks. Since `Model` does not impose any restriction on the inactive relaxation variables, it does not prune the search space efficiently for this huge number of soft clauses. As a result, `Model` deteriorates the performance of the traditional linear search algorithms for the industrial MaxSAT benchmarks.

For the close solution problem, all model-based algorithms exhibited a better performance than `Linear`. Model-based algorithms are able to solve most of the instances by using only a small subset of the relaxation variables. Since `Disabled` and `Hybrid` enumerate disjoint unsatisfiable subformulas, they are able to maintain a lower bound on the value of the optimal solution. For the majority of the instances on the close solution problem, `Disabled` and `Hybrid` are able to find the optimal solution due to the lower bound value being the same as the upper bound value. This scenario occurs particularly in the `[All]` partitioning strategy, i.e. when starting the search with all relaxation variables disabled. This may explain why the `Disabled` and `Hybrid` algorithms outperform the `Model` algorithm and why the `[All]` partitioning strategy outperforms the `[Model]` partitioning strategy.

Since both model-based algorithms improve the performance of linear search algorithms for the close solution problem, we will analyze in more detail its performance for this benchmark set. Figure 1 shows a cactus plot with the running times of the different linear search algorithms for the close solution problem. The cactus plot shows the

	ms_industrial	close_solution	pms_industrial	Total
Linear	48	196	874	1118
Model	25	208	851	1084
Disabled[Model]	89	214	869	1172
Disabled[All]	93	249	873	1215
Hybrid[Model]	81	225	868	1174
Hybrid[All]	89	262	873	1224

Table 3. Number of instances solved by each solver

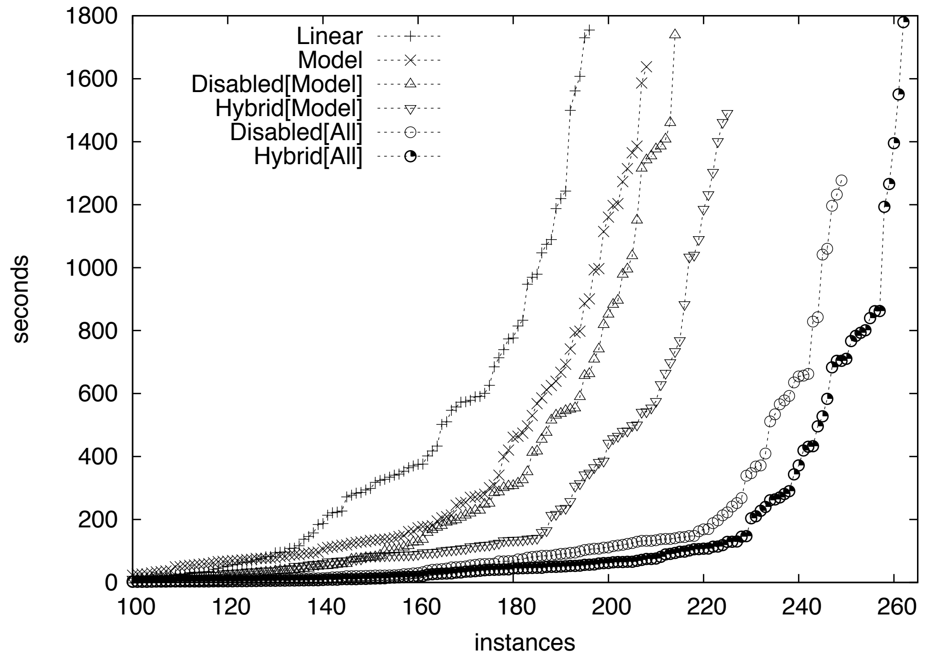


Figure 1. Running times of the different linear search algorithms for the close solution problem

sorted run times for each solver. Each point in the plot corresponds to a problem instance, where the y-axis corresponds to the time required by the solver and the x-axis corresponds to the accumulative number of instances solved until that time. All model-based algorithms clearly outperform the traditional linear search algorithm in terms of number of instances solved and in running time. The `[All]` partitioning strategy outperforms the `[Model]` partitioning strategy for the close solution problem. The `Model` algorithm performed the worst among the model-based algorithms. However, the `Hybrid` algorithm combines the `Model` and `Disabled` algorithms and improves the performance of the `Disabled` algorithm when using both `[All]` and `[Model]` partitioning strategies. Moreover, `Hybrid[All]` clearly outperforms all other model-based algorithms and significantly improves on traditional linear search algorithms.

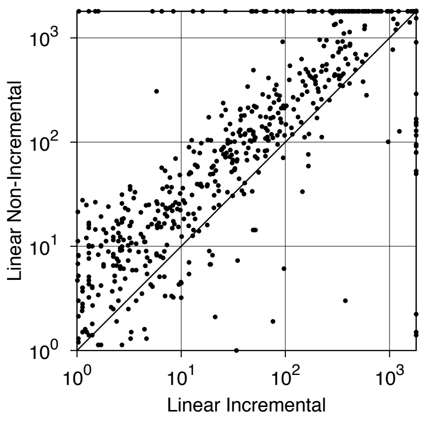
5.3. *Incrementality in MaxSAT*

Table 4 shows the number of instances solved with and without incrementality. Overall, incrementality always improves the performance of the solver. This is particularly evident in the `Linear` algorithm. Notice that the `Linear` algorithm can reuse all learned clauses. This may explain why incrementality is critical to the performance of this algorithm since it allows the solver to solve more 55 instances.

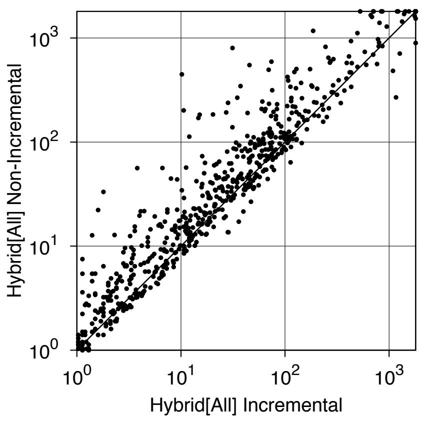
The `Model` also significantly improves its performance when using an incremental ap-

	Non-Incremental	Incremental
Linear	1063	1118
Model	1064	1084
Disabled[Model]	1165	1172
Disabled[All]	1211	1215
Hybrid[Model]	1167	1174
Hybrid[All]	1216	1224

Table 4. Number of instances solved with and without incrementality



(a) Linear search algorithm



(b) Hybrid model-based algorithm

Figure 2. Incrementality in MaxSAT

proach, since it is able to solve more 20 instances than without incrementality. Notice that on iterations where the set of active relaxation variables does not change, all learned clauses may be reused. On the remaining iterations, only learned clauses that are not unsafe are kept.

For algorithms that have unsatisfiable iterations, the incremental approach only slightly increases the number of solved instances. This is as expected, since only learned clauses that are not unsafe are kept for the next iteration when the working formula is found to be unsatisfiable.

Figure 2 shows a scatter plot between the non-incremental and incremental approaches for the `Linear` and `Hybrid[All]` algorithms. Each point in the plot corresponds to a problem instance, where the x-axis corresponds to the run time required by the incremental approach and the y-axis corresponds to the run time required by the non-incremental approach. If a point is above the diagonal, then it means that the incremental approach outperforms the non-incremental approach. Otherwise, it means that the non-incremental approach outperforms the incremental approach.

Figure 2 (a) shows that the incremental approach improves the solving time on the majority of the instances for the linear search algorithm. Moreover, the incremental approach clearly outperforms the non-incremental approach in terms of number of instances solved. However, there are some instances solved only when the non-incremental approach is used. Notice that the incremental approach encodes the cardinality constraint only once. In the next calls to the SAT solver, the cardinality encoding is updated by setting some specific literals to false. The number of auxiliary variables and clauses that are necessary to encode the cardinality constraint depend on the initial upper bound value[5]. If the number of relaxation variables is large and the initial upper bound value is also large, then the encoding may generate a large number of auxiliary variables and clauses. On the other hand, in the non-incremental approach, every time a new upper bound value is found the formula is rebuilt and the cardinality constraint is re-encoded. Hence, the size of the encoding will become smaller at each call to the SAT solver. Therefore, there is a trade off between rebuilding the encoding and using incrementality. Even though incrementality is better for the majority of the cases, for some instances rebuilding the encoding may lead to better results.

Figure 2 (b) shows that the incremental approach significantly reduces the solving time of the hybrid model-based algorithm. Note that the hybrid model-based algorithm rebuilds the cardinality constraint every time the set of active relaxation variables changes or when the SAT solver returns unsatisfiable. Thus, in the hybrid model-based algorithm the problem that was observed when using an incremental approach for the linear search algorithm does not occur.

5.4. *State-of-the-art MaxSAT solvers*

In this section we compare austin with our best performing algorithm (`Hybrid[All]`) against the following state-of-the-art MaxSAT solvers:

- qmaxsat[6] (Koshimura et al., 2012): uses the traditional linear search algorithm described in Algorithm 1. qmaxsat is based on `Glucose 2.0` (Audemard and Simon, 2009) and uses the totalizer encoding (Bailleux and Boufkhad, 2003) to encode cardinality constraints. qmaxsat is an incremental solver that uses incremental strengthening to update the cardinality encoding. qmaxsat was the winner of the industrial category of partial MaxSAT in the MaxSAT evaluation of 2012.
- wpm1[6] (Ansótegui et al., 2009): uses an unsatisfiability-based algorithm. wp1 is a non-incremental solver based on `PicoSAT` (Biere, 2008) and uses the regular encoding (Ansótegui and Manyà, 2005) to encode cardinality constraints.
- msu4 (Marques-Silva and Planes, 2008): uses a similar algorithm to the model-based algorithm with disabled relaxation variables described in Algorithm 4. msu4 is a non-incremental solver that uses the `MSUnCore` framework (Morgado et al., 2013) and is based on `PicoSAT` (Biere, 2008). The cardinality network encoding (Asín et al., 2011) is used to encode cardinality constraints.
- msu-bin-c-d (Heras et al., 2011; Morgado, Heras, and Marques-Silva, 2012): uses a binary search algorithm that is guided by unsatisfiable subformulas. msu-bin-c-d is a non-incremental solver that uses the `MSUnCore` framework (Morgado et al., 2013) and is based on `PicoSAT` (Biere, 2008). The cardinality network encoding (Asín et al., 2011) is used to encode cardinality constraints.

Table 5 compares the number of instances solved by state-of-the-art MaxSAT solvers.

wpm1 shows a good performance on instances where the number of soft clauses is large, namely on industrial MaxSAT instances and the close solution problem. On the other hand, it shows a poor performance on instances where the number of soft clauses is small, i.e. on partial industrial MaxSAT instances.

msu-bin-c-d is the most robust of the state-of-the-art MaxSAT solvers since it shows the best performance when the number of soft clauses is small and it does not underperform when the number of soft clauses increases.

[5]The cardinality networks encoding requires $O(n\ log^2\ k)$ additional variables and clauses, where n is the number of relaxation variables in the cardinality constraint and k is the initial upper bound value.
[6]Version from the MaxSAT evaluation 2012.

Solver	ms_industrial	close_solution	pms_industrial	Total
AUSTIN	89	262	873	1224
MSU-BIN-C-D	92	189	883	1164
QMAXSAT	61	183	876	1120
MSU4	93	195	822	1110
WPM1	102	221	722	1045

Table 5. Number of instances solved by state-of-the-art MaxSAT solvers

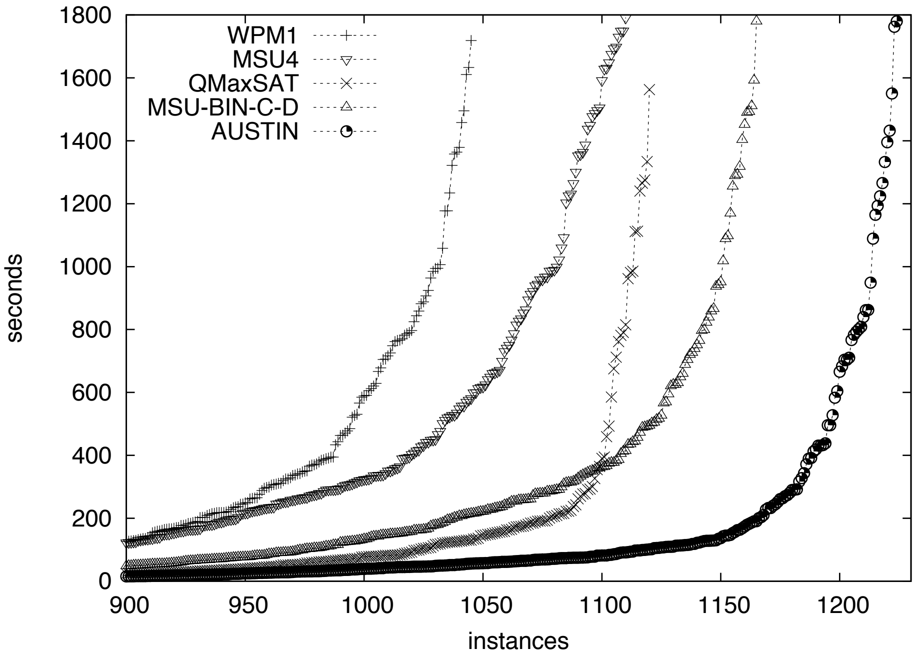


Figure 3. Running times of state-of-the-art MaxSAT solvers on all instances

AUSTIN significantly improves linear search algorithms on solving instances where the number of soft clauses is large while preserving their performance on instances where the number of soft clauses is small. AUSTIN is the best performing solver for the close solution instances and is the most robust solver thus outperforming MSU-BIN-C-D.

Figure 3 shows a cactus plot with the running times of state-of-the-art MaxSAT solvers on all instances. The results are clear: AUSTIN outperforms all other MaxSAT solvers in terms of solving time and number of instances solved. AUSTIN is able to improve the current state of the art in MaxSAT solving, in particular when dealing with problems with a large number of soft clauses.

6. Conclusions

Linear search algorithms for MaxSAT have shown to be particularly effective when the number of soft clauses is small. However, if the number of soft clauses increases, then the performance of linear search algorithms tends to deteriorate. This is due to the fact that the cardinality constraint is expressed over all relaxation variables. When the number of soft clauses increases, the cardinality encoding that enforces an upper bound on the relaxation variables also increases in the number of auxiliary clauses and variables. Therefore, the search space can grow significantly for problem instances with a large number of soft clauses.

In this paper, we have described different model-based algorithms that start by imposing a cardinality constraint over a subset of the relaxation variables. Even though model-based algorithms may require a larger number of calls to the SAT solver than traditional linear search algorithms, these calls are usually easier to solve and may be able to find better upper bound values. Moreover, since we are only considering a subset of the relaxation variables, the number of auxiliary clauses and variables needed for the cardinality encoding can be much smaller than in the original linear search algorithm.

Model-based algorithms have shown to be effective for problem instances where the number of soft clauses is large. Disabling relaxation variables allows model-based algorithms to further improve their performance on problem instances where the number of soft clauses is large while preserving their performance on instances where the number of soft clauses is small.

Experimental results also show that incremental approaches outperform non-incremental approaches. This shows the importance of keeping learned clauses between calls to the SAT solver. Furthermore, even on cases where not all learned clauses may be kept, it is important to keep learned clauses that remain valid for the next call to the SAT solver.

Our new hybrid model-based algorithm outperforms state-of-the-art MaxSAT solvers and improves the current state of the art in MaxSAT solving, in particular when solving problems with a large number of soft clauses.

Due to the success of model-based algorithm in partial MaxSAT problems, as future work, we propose to extend the model-based algorithms presented in this paper for weighted MaxSAT problems by using appropriate CNF encodings.

Acknowledgements

This work was partially supported by national funds through FCT - Fundação para a Ciência e a Tecnologia, under projects PEst-OE/EEI/LA0021/2013, iExplain (PTDC/EIA-CCO/102077/2008) and ASPEN (PTDC/EIA-CCO/110921/2009).

References

I. Abío and P. J. Stuckey. Conflict Directed Lazy Decomposition. In *Principles and Practice of Constraint Programming*, pages 70–85, 2012.

I. Abío, M. Deters, R. Nieuwenhuis, and P. J. Stuckey. Reducing Chaos in SAT-Like Search: Finding Solutions Close to a Given One. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 273–286, 2011.

C. Ansótegui and F. Manyà. Mapping Many-Valued CNF Formulas to Boolean CNF Formulas. In *International Symposium on Multiple-Valued Logic*, pages 290–295, 2005.

C. Ansótegui, M. L. Bonet, and J. Levy. Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 427–440, 2009.

C. Ansótegui, M. L. Bonet, and J. Levy. A New Algorithm for Weighted Partial MaxSAT. In *AAAI Conference on Artificial Intelligence*, pages 3–8, 2010.

J. Argelich, C. M. Li, and F. Manyà. An improved exact solver for Partial Max-SAT. In *Proceedings of the International Conference on Nonconvex Programming: Local and Global Approaches (NCP-2007)*, pages 230–231, 2007.

R. Asín and R. Nieuwenhuis. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research*, pages 1–21, 2012.

R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Practical algorithms for unsatisfiability proof and core generation in SAT solvers. *AI Communications*, 23 (2-3):145–157, 2010.

R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality Networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

G. Audemard and L. Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *International Joint Conferences on Artificial Intelligence*, pages 399–404, 2009.

G. Audemard, J.-M. Lagniez, and L. Simon. Improving glucose for incremental sat solving with assumptions: Application to mus extraction. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 309–317, 2013.

O. Bailleux and Y. Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming*, pages 108–122, 2003.

A. Biere. PicoSAT Essentials. *JSAT*, 4(2-4):75–97, 2008.

Y. Chen, S. Safarpour, J. Marques-Silva, and A. G. Veneris. Automated Design Debugging With Maximum Satisfiability. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 29(11):1804–1817, 2010.

F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSAT: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

F. Heras, A. Morgado, and J. Marques-Silva. Core-Guided Binary Search Algorithms for Maximum Satisfiability. In *AAAI Conference on Artificial Intelligence*, pages 36–41, 2011.

F. Heras, A. Morgado, J. Planes, and J. Marques-Silva. Iterative SAT Solving for Minimum Satisfiability. In *International Conference on Tools with Artificial Intelligence*, pages 922–927, 2012.

M. Janota, I. Lynce, V. Manquinho, and J. Marques-Silva. PackUp: Tools for Package Upgradability Solving. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2):89–94, 2012.

M. Jose and R. Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *Conference on Programming Language Design and Implementation*, pages 437–446. ACM Press, 2011.

M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. QMaxSAT: A Partial Max-SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:95–100, 2012.

D. Le Berre and A. Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–6, 2010.

C. M. Li and F. Manyà. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, pages 613–631. IOS Press, 2009.

C. M. Li, F. Manyà, and J. Planes. New Inference Rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.

H. Lin and K. Su. Exploiting Inference Rules to Compute Lower Bounds for MAX-SAT Solving. In *International Joint Conferences on Artificial Intelligence*, pages 2334–2339, 2007.

P.-C. K. Lin and S. P. Khatri. Application of Max-SAT-based ATPG to optimal cancer therapy design. *BMC Genomics*, 13((Suppl 6):S5), 2012.

V. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for Weighted Boolean Optimization. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 495–508, 2009.

J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Design, Automation, and Test in Europe Conference*, pages 408–413, 2008.

J. Marques-Silva and K. A. Sakallah. GRASP: A New Search Algorithm for Satisfiability. In *International Conference on Computer-Aided Design*, pages 220–227, 1996.

R. Martins, V. Manquinho, and I. Lynce. Parallel Search for Maximum Satisfiability. *AI*

Communications, 25(2):75–95, 2012.

R. Martins, V. Manquinho, and I. Lynce. Model-based Partitioning for MaxSAT Solving . In *RCRA International Workshop on Experimental Evaluation of Algorithms for solving problems with combinatorial explosion*, 2013.

A. Morgado, F. Heras, and J. Marques-Silva. Improvements to Core-Guided Binary Search for MaxSAT. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 284–297, 2012.

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

C. Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming*, pages 827–831, 2005.

L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In *International Conference on Computer-Aided Design*, pages 279–285, 2001.