To appear in the *Journal of Experimental & Theoretical Artificial Intelligence*
Vol. 00, No. 00, Month 20XX, 1–28

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 to improve on the performance of linear search algorithms for MaxSAT by using 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. Additionally, the proposed 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 classic 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 are competitive with state-of-the-art MaxSAT solvers, and can improve their performance when solving 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

*Corresponding author. Email: ruben@sat.inesc-id.pt 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 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.

The main goal of this paper is to improve the performance of classic linear search algorithms for MaxSAT instances with a large number of soft clauses, while maintaining their performance for MaxSAT instances with a small number of soft clauses. 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 the RCRA 2013 workshop paper, 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 the RCRA 2013 workshop paper 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*. A formula is satisfiable if there exists a solution to the formula. Otherwise, if there is no solution solution to the formula, then the formula is unsatisfiable.

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 formula variables such that all hard clauses are satisfied, while minimizing the number of unsatisfied soft clauses. For simplicity, in the remainder of the paper we will consider the MaxSAT problem as being a particular case of the partial MaxSAT problem, where the set of hard clauses is empty. 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 set of clauses, where we allow multiple occurrences of the same clause.

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. In contrast to SAT formulas, note that a solution (or model) of a MaxSAT formula does not have to satisfy all clauses. A solution to a MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$ is any assignment ν that satisfies all clauses in φ_h. A solution ν is preferred (or improves on) another solution ν' if ν unsatisfies less soft clauses than ν'. An optimal solution (or optimal model) is an assignment ν that satisfies all clauses in φ_h while minimizes the number of unsatisfied clauses in φ_s Finally, a MaxSAT formula is said to be unsatisfiable if φ_h is unsatisfiable., i.e. there is no solution to the formula.

Example 2.1: *Consider the following MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$:*

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

An optimal solution to this formula is $x_1 = x_2 = x_3 = 0, x_4 = 1$. This assignment satisfies all hard clauses and only two 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. $\sum_{i=1}^{n} l_i \leq k$. The sum of n literals, l_i, corresponds to the number of literals that are satisfied. In other words, a cardinality constraint over n literals ensures that at most k literals can be satisfied.

Cardinality constraints occur naturally in different problems. In order to use MaxSAT formulations in those problems, cardinality constraints are encoded to CNF so that a MaxSAT solver can handle the resulting formula (Asín, Nieuwenhuis, Oliveras, and Rodríguez-Carbonell, 2011; Bailleux and Boufkhad, 2003; Sinz, 2005). Moreover, several algorithms for MaxSAT solving also rely on these constraints (Ansótegui et al., 2009; Heras et al., 2011; Martins et al., 2012).

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), and hybrid algorithms that combine Mixed Integer Programming (MIP) and SAT technology (Davies and Bacchus, 2011, 2013a,b).

Since this paper focuses on new methods for linear search algorithms, next we provide a detailed description of linear search algorithms for partial MaxSAT. We refer to (Li and Manyà, 2009) for branch and bound algorithms and to (Morgado, Heras, Liffiton,

Algorithm 1: Linear search algorithm for partial MaxSAT

```
Input: $\varphi = \varphi_h \cup \varphi_s$
Output: model to $\varphi$ or UNSAT
1  $(V_R, \text{model}, \mu, \varphi_W) \leftarrow (\emptyset, \emptyset, +\infty, \varphi_h)$
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      $(st, \nu, \varphi_C) \leftarrow \texttt{SAT}(\varphi_W)$
8      if $st$ = 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         $\varphi_W \leftarrow \varphi_W \cup \{\texttt{CNF}(\sum_{r \in V_R} r \leq \mu - 1)\}$
12     else
13         if model $= \emptyset$ then
14             return UNSAT                          // the MaxSAT formula is unsatisfiable
15         else
16             return model                                  // return model to $\varphi$
```

Planes, and Marques-Silva, 2013a) for unsatisfiability-based algorithms, respectively.

One approach for solving MaxSAT is to make a linear search over the number of unsatisfied soft clauses. The classic linear search algorithm for partial MaxSAT (Koshimura et al., 2012; Le Berre and Parrain, 2010) is presented as Algorithm 1. 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. 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 formula 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 status 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. If the working formula is unsatisfiable, then the original formula is also unsatisfiable and Algorithm 1 returns UNSAT (line 14). Otherwise, while the working formula remains satisfiable, the model ν provided by the SAT solver is stored and the algorithm computes 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, the algorithm has found an optimal solution to φ (line 16).

Example 2.2: *Consider the partial MaxSAT formula φ as defined in Equation (1). Algorithm 1 starts by relaxing the partial MaxSAT formula and the working formula φ_W is 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 that returns the following assignment ν: $\langle x_2 = x_3 = x_4 = r_1 = r_4 = 0,\ x_1 = r_2 = r_3 = r_5 = 1\rangle$. Since r_2, r_3 and r_5 were assigned truth value 1, the algorithm 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 the working formula is now as follows:

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

The algorithm makes another call to the SAT solver that returns another assignment ν: $\langle x_1 = x_2 = x_3 = r_2 = r_4 = r_5 = 0,\ x_4 = r_1 = r_3 = 1\rangle$. As a result, the cardinality constraint of the working formula is updated to $CNF(r_1 + r_2 + r_3 + r_4 + r_5 \leq 1)$.

Finally, the algorithm makes another call to the SAT solver which returns unsatisfiable. Therefore, the optimal solution is given by the previous model that corresponds to unsatisfying two soft clauses.

3. Model-based Approaches for MaxSAT

Classic 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 to classic linear search algorithms, model-based approaches do not use all relaxation variables in the cardinality constraint. The relaxation variables in the cardinality constraint are selectively increased considering the information provided by the SAT solver.

In this section we describe three model-based algorithms for solving partial MaxSAT problems. The first algorithm follows the basic model-based approach where the set of relaxation variables used in the cardinality constraint is increased each time a new model is found. The second algorithm extends the first algorithm by inducing a clear partitioning between the relaxation variables that are used in the cardinality constraint and those that are not used. In this algorithm, relaxation variables not used in the cardinality constraint are disabled between calls to the SAT solver. Finally, the third algorithm presents a variation of the second algorithm, where we only induce a partial partition of the relaxation variables that are not used in the cardinality constraint.

3.1. *Basic 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 formula variables such that it minimizes the number of satisfied clauses. Even though this algorithm was not proposed for

Algorithm 2: Model-based algorithm for partial MaxSAT

```
Input: $\varphi = \varphi_h \cup \varphi_s$
Output: model 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      $(st, \nu, \varphi_C) \leftarrow \mathtt{SAT}(\varphi_W)$
5      if $st = \mathsf{SAT}$ 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              $\mathsf{model} \leftarrow \nu$
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 $\mathsf{model} = \emptyset$ then
14             return UNSAT
15         else
16             return model
```

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. We adapted the model-based algorithm from MinSAT to MaxSAT by changing the relaxation procedure to the one used in linear search algorithms for MaxSAT. Similarly to the linear search algorithm, the model-based algorithm 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. At each stage of the algorithm, these sets are updated accordingly.

Definition 3.1 (Algorithm Stage)**:** *A new stage s of the algorithm starts when the SAT solver is called by the s^{th} time.*

Definition 3.2 (Active Variables)**:** *A relaxation variable r is considered active at stage s if the algorithm has already found at least one model where r is assigned value 1.*

Definition 3.3 (Inactive Variables)**:** *A relaxation variable r is considered inactive at stage s if the algorithm has not yet returned a model where r is assigned value 1.*

At each stage of the algorithm, 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 model is less than the best known upper bound value μ, then μ is updated and the current model is stored. Notice that, since the algorithm does not add a cardinality constraint over all relaxation variables, it is possible to obtain models that do not improve on those already found. Next, the algorithm adds 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, the algorithm has 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 3.4: *Consider again the same partial MaxSAT formula φ as defined in Equation (1). Similarly to the linear search algorithm, Algorithm 2 starts by relaxing the partial MaxSAT formula 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 that returns the following assignment ν: $\langle x_2 = x_3 = x_4 = r_1 = r_4 = 0,\ x_1 = r_2 = r_3 = r_5 = 1\rangle$. The set of active variables is now updated to $V_A = \{r_2, r_3, r_5\}$ and the upper bound μ to 3.

The working formula is now as follows:

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

The SAT solver is called again returning the assignment ν: $\langle x_1 = x_2 = x_3 = x_4 = r_2 = r_4 = 0,\ r_1 = r_3 = r_5 = 1\rangle$. As a result, r_1 is added to V_A, μ does not change and the cardinality constraint of the working formula is updated to $CNF(r_1 + r_2 + r_3 + r_5 \leq 2)$.

In the next call to the SAT solver a new model ν is found: $\langle x_1 = r_2 = r_3 = r_5 = 0,\ x_2 = x_3 = x_4 = r_1 = r_4 = 1\rangle$. and r_4 is added to V_A. μ is updated to 2 and the current model is saved.

The cardinality constraint of the working formula is now $CNF(r_1+r_2+r_3+r_4+r_5 \leq 1)$ 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 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 $\varphi_R = \varphi_h \cup \{\omega_i \cup \{r_i\} \mid \omega_i \in \varphi_s\}$.

Lemma 3.5: *φ_R is satisfiable 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 define a model to φ_R extending ν by assigning all relaxation variables to value 1. □

Lemma 3.6: *Given an optimal solution for MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$, a model for φ_R can be built such that the number of relaxation variables assigned value 1 is minimum with respect to the total number of relaxation variables.*

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 extend ν to build a model 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 model 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. □

Lemma 3.7: *Given a model for φ_R such that the number of relaxation variables assigned value 1 is minimum with respect to the total number of relaxation variables, an optimal solution for MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$ can be built.*

Proof. Let ν_R be a model for φ_R where the number of relaxation variables assigned value 1 is minimum. In this case, one can build ν from ν_R by removing the assignments to the relaxation variables.

Note that ν satisfies all hard clauses φ_h and all soft clauses where the relaxation variables are assigned value 0 in ν_R. On the other hand, all soft clauses where the relaxation variables are assigned value 1 in ν_R are unsatisfied by ν. Notice that the number of unsatisfied soft clauses is minimal and ν is an optimal solution for MaxSAT formula φ.

If ν is not an optimal solution for φ, then it must exist another satisfiable assignment ν' such that the number of unsatisfiable soft clauses is smaller. In that case, one could build another assignment ν'_R that would satisfy φ_R with less relaxation variables assigned value 1 than ν_R. However, that contradicts the fact that the number of relaxation variables assigned value 1 in ν_R is minimum. □

Lemma 3.8: *When a relaxation variable is considered active at stage s of the algorithm, it remains active in all subsequent stages s' until the algorithm ends.*

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

Lemma 3.9: *A relaxation variable r only becomes active if Algorithm 2 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 3.10: *Given a partial MaxSAT formula φ, Algorithm 2 terminates and returns a 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. At each stage of the algorithm, if 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 assigned value 1 is excluded. Note that at each stage of the algorithm, the set V_A only increases and 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. *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 models 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 3.11 (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 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 later 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

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

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

```
Input: $\varphi = \varphi_h \cup \varphi_s$
Output: model 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 $\varphi_W \leftarrow \varphi_W \cup \{(\bar{r}) \mid r \in V_D\} \cup \{\mathrm{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
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         $\varphi_W \leftarrow \varphi_W \cup \{\mathrm{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
18     else
19         if $\varphi_C \cap \{(\bar{r}) \mid r \in V_D\} = \emptyset$ then
20             return model
21         else
22             $V_A \leftarrow V_A \cup \{r \in V_D \mid (\bar{r}) \in \varphi_C\}$     // update active variables
23             $V_D \leftarrow V_D \setminus V_A$                  // update disabled variables
24             $\lambda \leftarrow \lambda + 1$
25             $\varphi_W \leftarrow \varphi_{init} \cup \{(\bar{r}) \mid r \in V_D\} \cup \{\mathrm{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$
26     if $\lambda = \mu$ then
27         return model
```

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 presented (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 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 the initial model are put in the set of disabled variables.

Example 3.12: *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 the algorithm is using a partitioning strategy based on finding an initial model. The SAT solver is called on φ_W and returns the following assignment $\nu : \langle x_2 = x_3 = x_4 = r_1 = r_4 = 0,\ x_1 = r_2 = r_3 = r_5 = 1\rangle$.

The algorithm updates the set of active relaxation variables to $V_A = \{r_2, r_3, r_5\}$ and the set of disabled relaxation variables to $V_D = \{r_1, r_4\}$. The current model is saved and μ is set to 3. The working formula is now as follows:

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

Next, the algorithm makes another call to the SAT solver that returns unsatisfiable and φ_C contains $(\bar{r}_1)$. The algorithm updates 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 $V_D = \{r_4\}$. 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 and returns the 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 $V_D = \emptyset$, respectively. The working formula is rebuilt again.

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 Algorithm 3

Lemma 3.13: *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 3.14: *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. □

At each stage of 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 3.15: *Suppose that at stage s, 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 until stage s of the algorithm. □

Lemma 3.16: *Suppose that a model ν using μ relaxation variables assigned to 1 is the best solution found until stage s of the algorithm. 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 3.17: *At stage s of the algorithm, 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 must be unsatisfied in order to improve on the best solution found so far.*

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 $\varphi_R \cup \{\texttt{CNF}(\sum_{r \in V_A} r \leq \mu - 1)\}$. As a result, μ must correspond to the value of an optimal solution that was found in previous stages (lemma 3.16).

Otherwise, at least one of the variables in V'_D must be assigned value 1 for φ_W to be satisfiable. Hence, at least one of the corresponding soft clauses will be unsatisfied in any solution that improves on the best found so far. □

Lemma 3.18: *Let λ denote a lower bound on the number of unsatisfied clauses in an optimal solution. If at stage s of the algorithm, φ_W is unsatisfiable, then λ can be increased by 1.*

Proof. If φ_W is unsatisfiable at stage s of the algorithm and 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 3.16).

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, if φ_W is unsatisfiable at stage s, the subset of unit clauses on disabled relaxation variables is disjoint from the ones found in previous stages. Therefore, at each stage, if an optimal solution was not found, a disjoint subset of soft clauses is identified such that at least one will be unsatisfied in an improved solution. As a result, the lower bound λ can be safely increased by 1 when φ_W is unsatisfiable. □

Theorem 3.19: *Given a partial MaxSAT formula φ, Algorithm 3 terminates and returns a 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 3.5). 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. At each stage of the algorithm, 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 3.15).

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 3.16).

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 for φ_W to become satisfiable. 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 3.18). 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 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, the number of relaxation variables that become active are reduced 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

Algorithm 4: Hybrid model-based algorithm for partial MaxSAT

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

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 3.20: *Consider again the same partial MaxSAT formula φ as defined in Equation (1). The initial working formula φ_W is as follows:*

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

Suppose we are using a partitioning strategy based on finding an initial model. The SAT solver is called on φ_W and returns the following assignment ν : $\langle x_2 = x_3 = x_4 = r_1 = r_4 = 0,\ x_1 = r_2 = r_3 = r_5 = 1\rangle$. The set of active relaxation variables is set 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. As a result, the working formula is now as follows:

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

After updating the working formula, the algorithm makes another call to the SAT solver that returns unsatisfiable and φ_C contains $(\bar{r}_1)$. The algorithm updates the set of inactive relaxation variables to $V_I = \{r_1\}$ and the set of disabled relaxation variables to $V_D = \{r_4\}$. As a result, the working formula is now as follows:

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

Next, the SAT solver is called and returns the assignment ν: $\langle x_1 = x_2 = x_3 = x_4 = r_2 = r_4 = 0,\ r_1 = r_3 = r_5 = 1\rangle$. As a result, r_1 is added to V_A, μ' is 3 and so μ is not updated, but the cardinality constraint of the working formula is updated to $CNF(r_1 + r_2 + r_3 + r_5 \leq 2)$.

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 $V_I = \{r_4\}$, respectively. The working formula is rebuilt as follows:

In the next call to the SAT solver a new assignment ν is found: $\langle x_1 = r_2 = r_3 = r_5 = 0,\ x_2 = x_3 = x_4 = r_1 = r_4 = 1\rangle$. As a result, r_4 is added to V_A, μ is updated to 2 and the current model is saved. The cardinality constraint of the working formula is now $CNF(r_1 + r_2 + r_3 + r_4 + r_5 \leq 1)$ and another call to the SAT solver is made. Finally, the SAT solver returns unsatisfiable 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.

Proof Algorithm 4

Lemma 3.21: *In Algorithm 4, when a relaxation variable is moved from disabled to inactive it never becomes disabled again in all subsequent stages s' until the algorithm ends.*

Proof. If the initial working formula φ_W is satisfiable, V_D is initialized with all relaxation variables assigned value 0 in the first model found (line 7). In the next stages of the algorithm, when φ_W is unsatisfiable, disabled variables that belong to the unsatisfiable core φ_C are added to the set of inactive variables (line 25) and removed from V_D (line 26). The set V_D is not modified elsewhere. □

Lemma 3.22: *In Algorithm 4, when a relaxation variable is moved from inactive to active it remains active in all subsequent stages s' until the algorithm ends.*

Proof. Whenever a new model for is found at stage s of the algorithm, inactive relaxation variables may become active (line 15). The set V_A is not modified elsewhere. □

These two lemmas show that relaxation variables initially classified as disabled are moved to inactive and then to active variables. Note also that lemmas 3.8 and 3.9 also apply to Algorithm 4.

Theorem 3.23: *Given a partial MaxSAT formula φ, Algorithm 4 terminates and returns a correct solution.*

Proof. The initial part of Algorithm 4 is very similar to Algorithm 3. It 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 3.5). Otherwise, relaxation variables are partitioned into active (V_A) and disabled (V_D). Notice that initially the set of inactive variables (V_I) is empty.

Next, the algorithm iterates through the models of working formula φ_W in order to find an optimal solution. At each stage, if a new solution 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. The algorithm only saves the new model if it improves on the best model found in previous stages (lines 17-19). The constraint in the working formula φ_W only excludes models with greater or equal number of active relaxation variables (line 20). However, this suffices since the set of active relaxation variables only increases (lemmas 3.21 and 3.22) and its size is always larger or equal to the value of the best solution found μ.

As in Algorithm 3, one assumes that disabled relaxation variables can be assigned value 0 in working formula φ_W, 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 3.16).

Otherwise, the assumption that 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 for φ_W to become satisfiable. Therefore, in Algorithm 4, these variables are moved from V_D to V_I and the working formula is rebuilt accordingly (lines 25-28). Moreover, the lower bound λ is also updated (lemma 3.18).

Eventually, the set of disabled relaxation variables may become empty 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 29-30). □

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 and consider whether learned clauses should be kept for the next iteration. Recall that a new clause is learned by a SAT solver whenever a conflict is found with the goal of preventing the same conflict from occurring in the subsequent search (Marques-Silva and Sakallah, 1996; Zhang, Madigan, Moskewicz, and Malik, 2001). Discarding all learned clauses between calls to the SAT solver is called *non-incremental MaxSAT solving*. On the other hand, keeping (some of the) learned clauses between calls to the SAT solver is called *incremental MaxSAT solving*. Note that only the learned clauses that guarantee correctness can be kept. Incremental MaxSAT solving is known to be essential for boosting the performance of MaxSAT solvers.

Incrementality in MaxSAT is tightly related to the result of each call of the SAT solver and also 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 rebuild 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 4.1 (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 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 classic linear search algorithm presented in Algorithm 1.

Table 1. Incrementality in linear search algorithms

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

- `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. All experiments were run on two AMD Opteron 6276 processors (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, respectively,

Table 2. Average number of soft clauses per instance

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

`[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'. The close solution problem is encoded in MaxSAT by taking the original solution and encoding each literal as a unit soft clauses. New clauses are added as hard clauses. Close solution instances may have a large number of soft clauses since the number of soft clauses is the same as the number of variables. The underlying SAT instances from the close solution benchmark set come from timetabling and from satisfiable instances from the industrial category of the SAT Competition 2011.

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, 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

[2]The AUSTIN framework is 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.

Table 3. Number of instances solved by each solver

	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

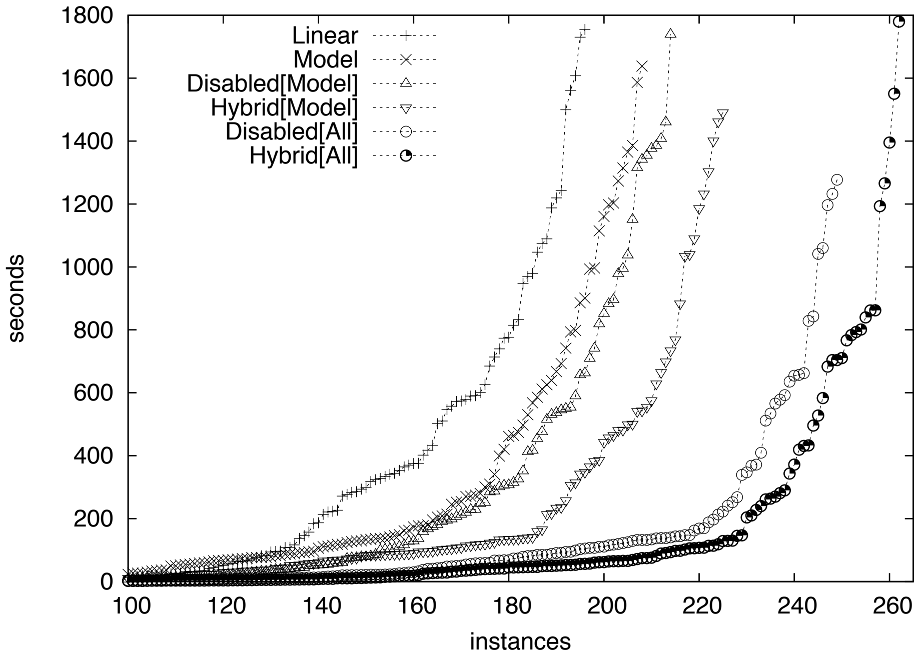


Figure 1. Running times of the different linear search algorithms for the close solution problem

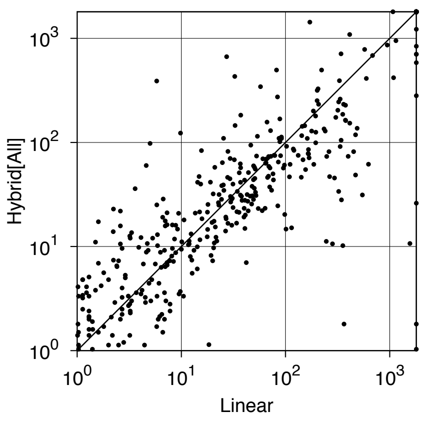
but it is much larger than in the partial industrial benchmarks. On average, each close solution problem instance has over 160,000 soft clauses.

5.2. *Model-based Algorithms*

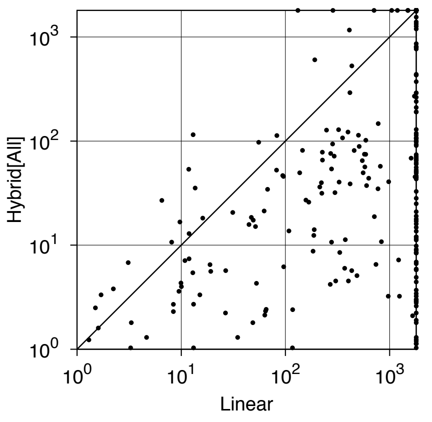
Table 3 shows the number of instances solved by each solver for the different benchmarks.

For the partial industrial MaxSAT benchmarks, model-based algorithms do not perform as well as the classic 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 classic 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



(a) Instances with 10,000 or less soft clauses (1,113 instances)



(b) Instances with more than 10,000 soft clauses (466 instances)

Figure 2. Impact of the number of soft clauses

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 classic 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 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 classic 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

Table 4. Number of instances solved with and without incrementality

	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

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 classic linear search algorithms.

Figure 2 shows scatter plots between the `Linear` and `Hybrid[All]` showing the impact of the number of soft clauses in the performance of the solver. For a better understanding of the correlation between the number of soft clauses and the performance of the solver, we have split the soft clauses into two disjoint sets: (a) instances with a small number of soft clauses ($\leq$ 10,000 soft clauses), and (b) instances with a large number of soft clauses ($>$ 10,000 clauses). Each point in the scatter plot corresponds to a problem instance, where the x-axis corresponds to the run time required by the `Linear` algorithm and the y-axis corresponds to the run time required by the `Hybrid[All]`. If a point is above the diagonal, then it means that the `Linear` algorithm outperforms the `Hybrid[All]` algorithm. Otherwise, it means that the `Hybrid[All]` algorithm outperforms the `Linear` algorithm.

Figure 2 (a) shows that there is no clear winner for instances with a small number of soft clauses. On the other hand, Figure 2 (b) shows that `Hybrid[All]` clearly outperforms `Linear` for instances with a large number of soft clauses. This supports our claim that model-based algorithms are best suited for instances with a large number of soft clauses.

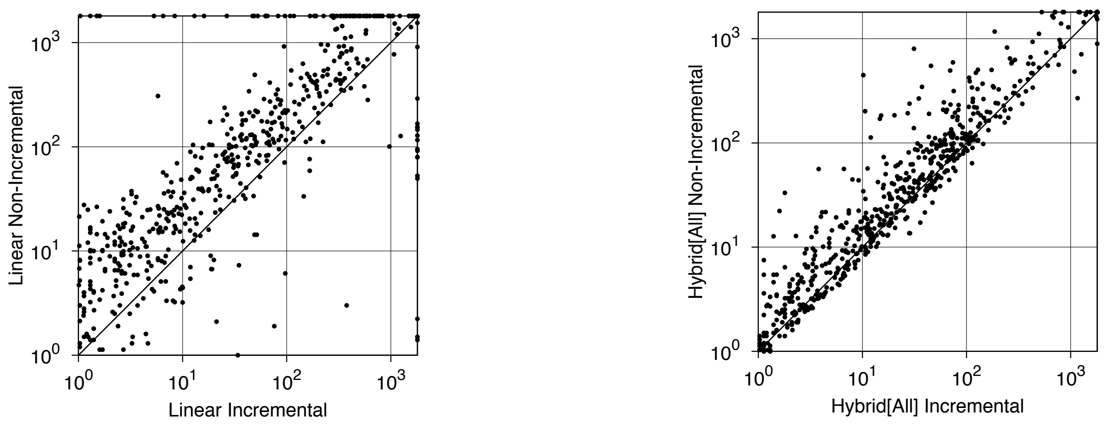
5.3. *Incrementality in MaxSAT*

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 approach, 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 3 shows a scatter plot between the non-incremental and incremental approaches for the `Linear` and `Hybrid[All]` algorithms. Figure 3 (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



(a) Linear search algorithm (b) Hybrid model-based algorithm

Figure 3. Incrementality in MaxSAT

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 3 (b) shows that the incremental approach significantly reduces the solving time of the hybrid model-based algorithm. Note that both incremental and non-incremental approaches of the hybrid model-based algorithm rebuild 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 non-incremental approach is almost always worst than the incremental approach.

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 classic 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. WPM1 is a non-incremental solver based on `PicoSAT` (Biere, 2008) and uses the regular

[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.

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

(a) Close solution benchmarks

Solver	close_solution
AUSTIN	262
WPM1	221
BCD2	197
MSU4	195
QMAXSAT	183
MAXHS	178

(b) MaxSAT evaluation benchmarks

Solver	ms_industrial	pms_industrial	Total
BCD2	95	886	981
AUSTIN	89	873	962
QMAXSAT	61	876	937
MAXHS	44	884	928
MSU4	93	822	915
WPM1	102	722	824

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., 2013a) and is based on `PicoSAT` (Biere, 2008). The cardinality network encoding (Asín et al., 2011) is used to encode cardinality constraints.
- BCD2 (Heras et al., 2011; Morgado, Heras, and Marques-Silva, 2012): uses a binary search algorithm that is guided by unsatisfiable subformulas. BCD2 is a non-incremental solver that uses the `MSUnCore` framework (Morgado et al., 2013a) and is based on `PicoSAT` (Biere, 2008). The cardinality network encoding (Asín et al., 2011) is used to encode cardinality constraints.
- MAXHS (Davies and Bacchus, 2011, 2013a,b): uses a hybrid approach between a Mixed-Integer Linear Programming (MILP) solver and a SAT solver. MAXHS uses `MiniSAT 2.0` (Eén and Sörensson, 2003) and `CPLEX 12.6` [7] as the SAT and MILP solvers respectively. Cardinality constraints are not encoded to CNF and are handle directly by the MILP solver.

Table 5 compares the number of instances solved by state-of-the-art MaxSAT solvers. Table 5 (a) shows the number of instances solved by MaxSAT solvers for the close solution benchmarks, whereas Table 5 (b) shows the number of instances solved by MaxSAT solvers for MaxSAT evaluation benchmarks (ms_industrial and pms_industrial benchmarks).

For the close_solution benchmarks, all MaxSAT solvers have difficulties handling the large number of soft clauses. Since AUSTIN is best suited for instances with a large number of soft clauses, it outperformed the remaining MaxSAT solvers.

For the MaxSAT evaluation benchmarks, BCD2 is the most robust solver. Unsatisfiability-based solvers (BCD2, MSU4 and WPM1) are the best performing solvers for the ms_industrial instances, since the number of unsatisfiable cores to be found to prove optimality is usually small. Linear search algorithm QMAXSAT performs poorly since it needs to encode a cardinality constraint on a huge number of relaxation variables. This is avoided in AUSTIN due to the techniques proposed in the paper. As a result, AUSTIN performs much better than QMAXSAT. On the pms_industrial benchmarks, the performance of BCD2, MAXHS, QMAXSAT and AUSTIN is similar. Nevertheless, BCD2 solves a few more instances than the other solvers within the time limit.

Overall, results show that model-based AUSTIN MaxSAT solver is competitive with state-of-the-art MaxSAT solvers. Even though, it is not the best for the ms_industrial or the pms_industrial benchmarks it only solved 19 less instances than BCD2 on the MaxSAT

[7] Available at `http://www-03.ibm.com/ibm/university/academic/pub/page/ban_ilog_programming` under the academic initiative program.

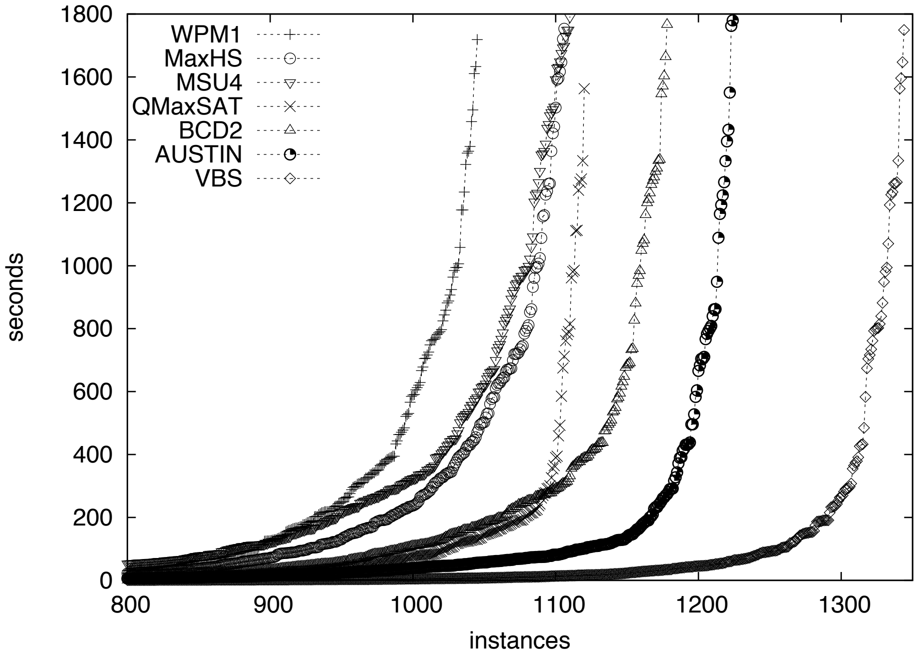


Figure 4. Running times of state-of-the-art MaxSAT solvers on all instances

evaluation benchmarks. However, AUSTIN shows to be able to solve more instances from the close solutions benchmark set. In this set, linear search algorithms have a problem in dealing with a large number of soft clauses and unsatisfiability-based solvers need to find a large number of unsatisfiable cores in order to find an optimal solution.

Figure 4 shows a cactus plot with the running times of state-of-the-art MaxSAT solvers on all instances. In addition to the solvers previously mentioned, we have also included the Virtual Best Solver (VBS) between all solvers. The VBS shows the best running time for each instance, i.e. for each instance it considers the minimum running time between AUSTIN, BCD2, QMAXSAT, MSU4, MAXHS, and WPM1.

AUSTIN solves more instances when considering all sets of instances. This is mostly due to the good performance of AUSTIN on the close_solution benchmarks while being competitive with state-of-the-art MaxSAT solvers on the remaining benchmarks. However, the VBS solves significantly more instances than AUSTIN. This shows that different solvers are able to solve different sets of instances, and that there is no clear winner for all benchmarks. For example, MAXHS has a poor performance when considering all instances, but it has an important contribution to the VBS.

6. Conclusions and Future Work

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 classic 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 classic linear search algorithm. In this paper we show that model-based algorithms are effective, in particular for problem instances where the number of soft clauses is large. Moreover, disabling a subset of relaxation variables allows model-based algorithms to further improve their performance.

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 linear search algorithms for problem instances where the number of soft clauses is large, while maintaining the same performance as linear search algorithms for problem instances with a small number of soft clauses. Moreover, our new hybrid model-based algorithm is competitive with state-of-the art MaxSAT solvers.

Recently and independently, model-guided approaches have also been applied to binary search algorithms (Morgado, Heras, and Marques-Siva, 2013b). However, the authors only use a basic model-guided approach, which can be further extended by the hybrid algorithms presented in this paper. As future work, we propose to study the effectiveness of our new hybrid model-based algorithm when applied to binary search algorithms.

Due to the success of model-based algorithm in partial MaxSAT problems, as future work, we also 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 M. Milano, editor, *Principles and Practice of Constraint Programming*, volume 7514 of *LNCS*, pages 70–85. Springer, 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 K. A. Sakallah and L. Simon, editors, *International Conference on Theory and Applications of Satisfiability Testing*, volume 6695 of *LNCS*, pages 273–286. Springer, 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. IEEE Computer Society Press, 2005.

C. Ansótegui, M. L. Bonet, and J. Levy. Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In O. Kullmann, editor, *International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *LNCS*, pages 427–440. Springer, 2009.

C. Ansótegui, M. L. Bonet, and J. Levy. A New Algorithm for Weighted Partial MaxSAT. In M. Fox and D. Poole, editors, *AAAI Conference on Artificial Intelligence*, pages 3–8. AAAI Press, 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*, 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. IJCAI/AAAI Press, 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*, volume 7962 of *LNCS*, pages 309–317. Springer, 2013.

O. Bailleux and Y. Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. In F. Rossi, editor, *Principles and Practice of Constraint Programming*, volume 2833 of *LNCS*, pages 108–122. Springer, 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.

J. Davies and F. Bacchus. Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. In J. H.-M. Lee, editor, *Principles and Practice of Constraint Programming*, volume 6876 of *LNCS*, pages 225–239. Springer, 2011.

J. Davies and F. Bacchus. Postponing Optimization to Speed Up MAXSAT Solving. In C. Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *LNCS*, pages 247–262. Springer, 2013a.

J. Davies and F. Bacchus. Exploiting the Power of mip Solvers in maxsat. In M. Järvisalo and A. V. Gelder, editors, *International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *LNCS*, pages 166–181. Springer, 2013b.

N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

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 W. Burgard and D. Roth, editors, *AAAI Conference on Artificial Intelligence*, pages 36–41. AAAI Press, 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. IEEE Computer Society Press, 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 M. W. Hall and D. A. Padua, editors, *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 M. M. Veloso, editor, *International Joint Conferences on Artificial Intelligence*, pages 2334–2339. IJCAI/AAAI Press, 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 O. Kullmann, editor, *International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *LNCS*, pages 495–508. Springer, 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. IEEE Computer Society Press, 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. IEEE Computer Society, 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*, volume 7317 of *LNCS*, pages 284–297. Springer, 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, 2013a.

A. Morgado, F. Heras, and J. Marques-Siva. Model-Guided Approaches for MaxSAT. In *International Conference on Tools with Artificial Intelligence*. IEEE Computer Society Press, 2013b.

C. Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In P. van Beek, editor, *Principles and Practice of Constraint Programming*, volume 3709 of *LNCS*, pages 827–831. Springer, 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. IEEE Computer Society Press, 2001.