Improving SAT-Based Weighted MaxSAT Solvers*

Carlos Ansótegui[1], Maria Luisa Bonet[2], Joel Gabàs[1], and Jordi Levy[3]

[1] DIEI, Univ. de Lleida
{carlos,joel.gabas}@diei.udl.cat
[2] LSI, UPC
bonet@lsi.upc.edu
[3] IIIA-CSIC
levy@iiia.csic.es

Abstract. In the last few years, there has been a significant effort in designing and developing efficient Weighted MaxSAT solvers. We study in detail the WPM1 algorithm identifying some weaknesses and proposing solutions to mitigate them. Basically, WPM1 is based on iteratively calling a SAT solver and adding blocking variables and cardinality constraints to relax the unsatisfiable cores returned by the SAT solver. We firstly identify and study how to break the symmetries introduced by the blocking variables and cardinality constraints. Secondly, we study how to prioritize the discovery of higher quality cores. We present an extensive experimental investigation comparing the new algorithm with state-of-the-art solvers showing that our approach makes WPM1 much more competitive.

1 Introduction

Many combinatorial optimization problems can be modelled as Weighted Partial MaxSAT formulas. Therefore, Weighted Partial MaxSAT solvers can be used in several domains as: combinatorial auctions, scheduling and timetabling problems, FPGA routing, software package installation, etc.

The Maximum Satisfiability (MaxSAT) problem is the optimization version of the satisfiability (SAT) problem. The goal is to maximize the number of satisfied clauses in a SAT formula, in other words, to minimize the number of falsified clauses. The clauses can be divided into hard and soft clauses, depending on whether they must be satisfied (hard) or they may or may not be satisfied (soft). If our formula only contains soft clauses it is a MaxSAT formula, and if it contains both, hard and soft clauses, it is a Partial MaxSAT formula. The Partial MaxSAT problem can be further generalized to the Weighted Partial MaxSAT problem. The idea is that not all soft clauses are equally important.

* This research has been partially founded by the CICYT research projects TASSAT (TIN2010-20967-C04-01/03/04) and ARINF (TIN2009-14704-C03-01).

M. Milano (Ed.): CP 2012, LNCS 7514, pp. 86–101, 2012.

The addition of weights to soft clauses makes the formula Weighted, and lets us introduce preferences between them. The weights indicate the penalty for falsifying a clause. Given a Weighted Partial MaxSAT problem, our goal is to find an assignment that satisfies all the hard clauses, and the sum of the weights of the falsified clauses is minimal. Such an assignment will be optimal in this context.

SAT technology has evolved to a mature state in the last decade. SAT solvers are really successful at solving industrial decision problems. The next challenge is to use this technology to solve more efficiently industrial optimization problems. Although there has been important work in this direction, we have not reached the success of SAT solvers yet. The present work is one more step in MaxSAT technology to achieve full industrial applicability.

Originally, MaxSAT solvers such as WMaxSatz [12], MiniMaxSat [10], IncWMaxSatz [13] and akmaxsat where depth-first branch and bound based. Recently, there has been a development of SAT based approaches which essentially iteratively call a SAT solver: SAT4J [5], WBO and MSUNCORE [14], WPM1 [1], WPM2 [2], *BINC* and *BINCD* [11] and maxHS [8]. While branch and bound based solvers are competitive for random and crafted instances, SAT based solvers are better for industrial instances.

The WPM1, WBO and MSUNCORE solvers implement weighted versions of the Fu and Malik's algorithm [9]. Essentially, they perform a sequence of calls to a SAT solver, and if the SAT solver returns an unsatisfiable core, they reformulate the problem by introducing new auxiliary variables and cardinality constraints which relax the clauses in the core. Further details are given in section 3 and 5. In this work, we analyze in more detail the WPM1 algorithm to identify and mitigate some weaknesses. The first weakness we have observed is that the addition of the auxiliary variables naturally introduce symmetries which should be broken to achieve better performance. The second weakness has to do with the quality of the cores returned by the SAT solver. Since the SAT solver is used as a black box, we need to come up with new strategies to lead the solver to find better quality cores.

We have conducted an extensive experimental investigation with the best solvers at the last MaxSAT evaluation and other solvers that did not take part in the evaluation, but have been reported to show very good performance. We can see that our current approach can boost radically the performance of the WPM1 becoming the most robust approach.

This paper proceeds as follows: Section 2 introduces some preliminary concepts; Section 3 presents the Fu and Malik's algorithm; Section 4 describes the problem of symmetries and shows how to break them; Section 5 presents the WPM1 algorithm and describes the problem of the quality of the cores; Section 6 introduces an stratified approach to come up with higher quality cores; Section 7 presents some previous concepts needed to describe a general stratified approach discussed in Section 8 and finally Section 9 presents the experimental evaluation.

2 Preliminaries

We consider an infinite countable set of *boolean variables* $\mathcal{X}$. A *literal* l is either a variable $x_i \in \mathcal{X}$ or its negation $\overline{x_i}$. A *clause* C is a finite set of literals, denoted as $C = l_1 \vee \cdots \vee l_r$, or as $\square$ for the empty clause. A *SAT formula* φ is a finite set of clauses, denoted as $\varphi = C_1 \wedge \cdots \wedge C_m$.

A *weighted clause* is a pair (C, w), where C is a clause and w is a natural number or infinity, indicating the penalty for falsifying C. A clause is called *hard* if the corresponding weight is infinity, otherwise the clause is called *soft*.

A *(Weighted Partial) MaxSAT formula* is a multiset of weighted clauses

$$\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$$

where the first m clauses are soft and the last m' clauses are hard. The set of variables occurring in a formula φ is noted as $\mathrm{var}(\varphi)$.

A *total truth assignment* for a formula φ is a function $I : \mathrm{var}(\varphi) \to \{0, 1\}$, that can be extended to literals, clauses, SAT formulas and MaxSAT formulas, the following way:

$$I(\overline{x}_i) = 1 - I(x_i)$$
$$I(l_1 \vee \ldots \vee l_r) = \max\{I(l_1), \ldots, I(l_r)\}$$
$$I(\{C_1, \ldots, C_m\}) = \min\{I(C_1), \ldots, I(C_m)\}$$
$$I(\{(C_1, w_1), \ldots, (C_m, w_m)\}) = w_1 \cdot (1 - I(C_1)) + \ldots + w_m \cdot (1 - I(C_m))$$

We define the *optimal cost* of a MaxSAT formula as

$$\mathrm{cost}(\varphi) = \min\{I(\varphi) \mid I : \mathrm{var}(\varphi) \to \{0, 1\}\}$$

and an *optimal assignment* as an assignment I such that $I(\varphi) = \mathrm{cost}(\varphi)$.

We also define *partial truth assignments* for φ as a partial function $I : \mathrm{var}(\varphi) \to \{0, 1\}$ where instantiated falsified literals are removed and the formula is simplified accordingly.

Example 1. Given $\varphi = \{(\overline{y}, 6), (x \vee y, 2), (x \vee z, 3), (y \vee z, 2)\}$ and $I : \{y, z\} \to \{0, 1\}$ such that $I(y) = 0$ and $I(z) = 0$, we have $I(\varphi) = \{(x, 5), (\square, 2)\}$. We also have $\mathrm{cost}(I(\varphi)) = 2$ and $\mathrm{cost}(\varphi) = 0$.

Notice that, for any MaxSAT formula φ and partial truth assignment I, we have $\mathrm{cost}(\varphi) \leq \mathrm{cost}(I(\varphi))$. Notice also that when w is finite, the pair (C, w) is equivalent to having w copies of the clause $(C, 1)$ in our multiset.

We say that a truth assignment I *satisfies* a literal, clause or a SAT formula if it assigns 1 to it, and *falsifies* it if it assigns 0. A SAT formula is *satisfiable* if there exists a truth assignment that satisfies it. Otherwise, it is *unsatisfiable*. Given an unsatisfiable SAT formula φ, an *unsatisfiable core* φ_c is a subset of clauses $\varphi_c \subseteq \varphi$ that is also unsatisfiable. A *minimal unsatisfiable core* is an unsatisfiable core such that any proper subset of it is satisfiable.

The *Weighted Partial MaxSAT problem* for a weighted partial MaxSAT formula φ is the problem of finding an *optimal assignment*. If the optimal cost

is infinity, then the subset of hard clauses of the formula is unsatisfiable, and we say that the formula is *unsatisfiable*. The *Weighted MaxSAT problem* is the Weighted Partial MaxSAT problem when there are no hard clauses. The *Partial MaxSAT problem* is the Weighted Partial MaxSAT problem when the weights of soft clauses are all equal. The *MaxSAT problem* is the Partial MaxSAT problem when there are no hard clauses. Notice that the *SAT problem* is equivalent to the Partial MaxSAT problem when there are no soft clauses.

3 The Fu and Malik's Algorithm

The first SAT-based algorithm for Partial MaxSAT algorithm was the Fu and Malik's algorithm described in [9]. It was implemented in the MaxSAT solver msu1.2 [17,18], and its correctness was proved in [1].

The algorithm consists in iteratively calling a SAT solver on a working formula φ. This corresponds to the line $(st, \varphi_c) := SAT(\{C \mid (C_i, w_i) \in \varphi\})$. The SAT solver will say whether the formula is satisfiable or not (variable st), and in case the formula is unsatisfiable, it will give an unsatisfiable core (φ_c). At this point the algorithm will produce new variables, blocking variables (BV in the code), one for each soft clause in the core. The new working formula φ will consist in adding the new variables to the soft clauses of the core, adding a cardinality constraint saying that exactly one of the new variables should be true ($CNF(\sum_{b \in BV} b = 1)$ in the code), and adding one to the counter of falsified clauses. This procedure is applied until the SAT solver returns SAT.

For completeness, we reproduce the code of the Fu and Malik's algorithm in Algorithm 1.

Next we present an example of execution that will be used in the next section.

Example 2. Consider the pigeon-hole formula PHP_1^5 with 5 pigeons and one hole where the clauses saying that no two pigeons can go to the same hole are hard, while the clauses saying that each pigeon goes to a hole are soft:

$$\varphi = \{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1), (x_5, 1), (\overline{x}_1 \vee \overline{x}_2, \infty), \ldots, (\overline{x}_4 \vee \overline{x}_5, \infty)\}$$

In what follows, the new b variables will have a super-index indicating the number of the unsatisfiable core, and a subindex indicating the index of the original soft clause.

Suppose that applying the FuMalik algorithm, the SAT solver computes the (minimal) unsatisfiable core $C_1 = \{1, 2\}$. Here we represent the core by the set of indexes of the soft clauses contained in the core. The new formula will be as shown on the right. At this point, the variable *cost* takes value 1.

$$\begin{aligned}
\varphi_1 = \{\ & (x_1 \vee b_1^1, 1), \\
& (x_2 \vee b_2^1, 1), \\
& (x_3, 1), \\
& (x_4, 1), \\
& (x_5, 1)\ \} \cup \\
& \{(\overline{x}_i \vee \overline{x}_j, \infty) \mid i \neq j\} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty)
\end{aligned}$$

Algorithm 1. The pseudo-code of the FuMalik algorithm (with a minor correction).

Input: $\varphi = \{(C_1, 1), \ldots, (C_m, 1), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \_)$ **then return** $(\infty, \emptyset)$ ▷Hard clauses are unsatisfiable
$cost := 0$ ▷Optimal
while *true* **do**
  $(st, \varphi_c) := SAT(\{C_i \mid (C_i, w_i) \in \varphi\})$ ▷Call to the SAT solver without weights
  if $st = \text{SAT}$ **then return** $(cost, \varphi)$
  $BV := \emptyset$ ▷Set of blocking variables
  foreach $C_i \in \varphi_c$ **do**
    if $w_i \neq \infty$ **then** ▷If the clause is soft
      $b :=$ new_variable( )
      $\varphi := \varphi \setminus \{(C_i, 1)\} \cup \{(C_i \vee b, 1)\}$ ▷Add blocking variable
      $BV := BV \cup \{b\}$
  $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{b \in BV} b = 1)\}$ ▷Add cardinality constraint as hard clauses
  $cost := cost + 1$

If the next unsatisfiable cores found by the SAT solver are $C_2 = \{3, 4\}$ and $C_3 = \{1, 2, 3, 4\}$, then the new formula will be:

$$\begin{aligned}
\varphi_2 = \{\ & (x_1 \vee b_1^1\ \ \ \ , 1),\\
& (x_2 \vee b_2^1\ \ \ \ , 1),\\
& (x_3 \vee\ \ b_3^2\ , 1),\\
& (x_4 \vee\ \ b_4^2\ , 1),\\
& (x_5\ \ \ \ \ \ , 1)\ \} \cup\\
& \{(\overline{x}_i \vee \overline{x}_j, \infty) \mid i \neq j\} \cup\\
& CNF(b_1^1 + b_2^1 = 1, \infty) \cup\\
& CNF(b_3^2 + b_4^2 = 1, \infty)
\end{aligned}$$

$$\begin{aligned}
\varphi_3 = \{\ & (x_1 \vee b_1^1 \vee\ \ \ \ b_1^3, 1),\\
& (x_2 \vee b_2^1 \vee\ \ \ \ b_2^3, 1),\\
& (x_3 \vee\ \ \ b_3^2 \vee b_3^3, 1),\\
& (x_4 \vee\ \ \ b_4^2 \vee b_4^3, 1),\\
& (x_5\ \ \ \ \ \ \ \ \ \ , 1)\ \} \cup\\
& \{(\overline{x}_i \vee \overline{x}_j, \infty) \mid i \neq j\} \cup\\
& CNF(b_1^1 + b_2^1 = 1, \infty) \cup\\
& CNF(b_3^2 + b_4^2 = 1, \infty) \cup\\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)
\end{aligned}$$

After the third iteration, the variable *cost* has value 3. Finally, after finding the core $C_4 = \{1, 2, 3, 4, 5\}$ we get the following satisfiable MaxSAT formula:

$$\begin{aligned}
\varphi_4 = \{\ & (x_1 \vee b_1^1 \vee\ \ \ \ b_1^3 \vee b_1^4\ , 1),\\
& (x_2 \vee b_2^1 \vee\ \ \ \ b_2^3 \vee b_2^4\ , 1),\\
& (x_3 \vee\ \ \ b_3^2 \vee b_3^3 \vee b_3^4\ , 1),\\
& (x_4 \vee\ \ \ b_4^2 \vee b_4^3 \vee b_4^4\ , 1),\\
& (x_5 \vee\ \ \ \ \ \ \ \ b_5^4\ , 1)\ \} \cup\\
& \{(\overline{x}_i \vee \overline{x}_j, \infty) \mid i \neq j\} \cup\\
& CNF(b_1^1 + b_2^1 = 1, \infty) \cup\\
& CNF(b_3^2 + b_4^2 = 1, \infty) \cup\\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty) \cup\\
& CNF(b_1^4 + b_2^4 + b_3^4 + b_4^4 + b_5^4 = 1, \infty)
\end{aligned}$$

At this point *cost* is 4. The algorithm will now call the SAT solver on φ_4, and the solver will return the answer "satisfiable". The algorithm returns $cost = 4$.

4 Breaking Symmetries

It is well known that formulas that contain a great deal of symmetries cause SAT solvers to explore many redundant truth assignments. Adding symmetry breaking clauses to a formula has the effect of removing the symmetries, while keeping satisfiability the same. Therefore it is a way to speed up solvers by pruning the search space.

In the case of executions of the FuMalik algorithm, symmetries can appear in two ways. On one hand, there are formulas that naturally contain many symmetries. For instance, in the case of the pigeon-hole principle we can permute the pigeons or the holes, leaving the formula intact. On the other hand, in each iteration of the FuMalik algorithm, we modify the formula adding new variables and hard constraints. In this process we can also introduce symmetries. In the present paper, we are no concerned with eliminating natural symmetries of a MaxSAT formula as in [16], since that might be costly, and it is not the aim of the present work. Instead we will eliminate the symmetries that appear in the process of performing the algorithm. In this case, it is very efficient to extract the symmetries given our implementation of the algorithm.

Before we formally describe the process of eliminating the symmetries, we will see an example.

Example 3. Consider again the pigeon-hole formula PHP_1^5 of Example 2. The working formula φ_3 from the previous section is still unsatisfiable, this is the reason to find a fourth core C_4. However, if we do not consider the clause x_5 the formula is satisfiable, and has 8 distinct models (two for each variable among $\{x_1, \ldots, x_4\}$ set to true). Here, we show 2 of the models, marking the literals set to true (we do not include the clauses $\overline{x}_i \vee \overline{x}_j$, for $i \neq j$ and put the true literals in boxes):

$$
\begin{array}{llll}
x_1 \vee & \boxed{b_1^1} \vee & & b_1^3 \\
x_2 \vee & b_2^1 \vee & & \boxed{b_2^3} \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3 \\
\multicolumn{4}{l}{\boxed{b_1^1} + b_2^1 = 1} \\
\multicolumn{4}{l}{\boxed{b_3^2} + b_4^2 = 1} \\
\multicolumn{4}{l}{b_1^3 + \boxed{b_2^3} + b_3^3 + b_4^3 = 1}
\end{array}
\qquad
\begin{array}{llll}
x_1 \vee & b_1^1 \vee & & \boxed{b_1^3} \\
x_2 \vee & \boxed{b_2^1} \vee & & b_2^3 \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3 \\
\multicolumn{4}{l}{b_1^1 + \boxed{b_2^1} = 1} \\
\multicolumn{4}{l}{\boxed{b_3^2} + b_4^2 = 1} \\
\multicolumn{4}{l}{\boxed{b_1^3} + b_2^3 + b_3^3 + b_4^3 = 1}
\end{array}
$$

The previous two models are related by the permutation $b_1^1 \leftrightarrow b_2^1, b_1^3 \leftrightarrow b_2^3$. The two ways of assigning values to the b variables are equivalent. The existence of so many *partial* models makes the task of showing unsatisfiability of the formula (including x_5) much harder.

The mechanism to eliminate the symmetries caused by the extra variables is as follows: suppose we are in the s iteration of the FuMalik algorithm, and we have obtained the set of cores $\{\varphi_1, \ldots, \varphi_s\}$. We assume that the clauses in the cores follow a total order. For clarity we will name the new variables of core φ_l for l such that $1 \leq l \leq s$ as b_i^l, where i is an index in φ_l. Now, we add the clauses:

$$b_i^s \rightarrow \overline{b}_j^l \qquad \text{for } l = 1, \ldots, s-1 \text{ and } i, j \in \varphi_l \cap \varphi_s \text{ and } j > i$$

This clauses implies that in Example 3 we choose the model on the left rather than the one on the right.

Example 4. For the Example 3, after finding the third unsatisfiable core C_3, we would add the following clauses to break symmetries (written in form of implications):

$$b_1^3 \rightarrow \overline{b}_2^1$$
$$b_3^3 \rightarrow \overline{b}_4^2$$

Adding these clauses, instead of the 8 partial models, we only have 4, one for each possible assignment of x_i to true.

After finding the fourth core C_4, we also add (written in compact form):

$$b_1^4 \rightarrow (\overline{b}_2^1 \wedge \overline{b}_2^3 \wedge \overline{b}_3^3 \wedge \overline{b}_4^3)$$
$$b_2^4 \rightarrow (\overline{b}_3^3 \wedge \overline{b}_4^3)$$
$$b_3^4 \rightarrow (\overline{b}_4^2 \wedge \overline{b}_4^3)$$

5 The WPM1 Algorithm

Algorithm 2 is the weighted version of the FuMalik algorithm described in section 3 [1,14] In this algorithm, we iteratively call a SAT solver with a weighted working formula, but excluding the weights. When the SAT solver returns an unsatisfiable core, we calculate the minimum weight of the clauses of the core (w_{min} in the algorithm.). Then, we transform the working formula in the following way: we duplicate the core having on one of the copies, the clauses with weight the original minus the minimum weight, and on the other copy we put the blocking variables and we give it the minimum weight. Finally we add the cardinality constraint on the blocking variables, and we add w_{min} to the *cost*.

The process of doubling the clauses might imply to end up converting clauses with weight say w into w copies of the clause of weight 1. When this happens, the process becomes very inefficient. In the following we show a (tiny) example that reflects this situation.

Example 5. Consider the formula $\varphi = \{(x_1, 1), (x_2, m), (\overline{x}_2, \infty)\}$.

Assume that the SAT solver always includes the first soft clause in the returned unsatisfiable core, even if this makes the core not minimal. After one iteration, the new formula would be:

$$\varphi_1 = \{(x_1 \vee b_1^1, 1), (x_2 \vee b_2^1, 1), (x_2, m-1), (\overline{x}_2, \infty), (b_1^1 + b_2^1 = 1, \infty)\}$$

Algorithm 2. The pseudo-code of the WPM1 algorithm.

Input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
1: **if** $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \_)$ **then return** $(\infty, \emptyset)$ ▷Hard clauses are unsatisfiable
2: $cost := 0$ ▷Optimal
3: **while** *true* **do**
4: $(st, \varphi_c) := SAT(\{C_i \mid (C_i, w_i) \in \varphi\})$ ▷Call to the SAT solver without weights
5: **if** $st = \text{SAT}$ **then return** $(cost, \varphi)$
6: $BV := \emptyset$ ▷Blocking variables of the core
7: $w_{min} := \min\{w_i \mid C_i \in \varphi_c \wedge w_i \neq \infty\}$ ▷Minimum weight
8: **foreach** $C_i \in \varphi_c$ **do**
9: **if** $w_i \neq \infty$ **then**
10: $b :=$ new_variable()
11: $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min}),\ (C_i \vee b, w_{min})\}$ ▷Duplicate soft clauses of the core
12: $BV := BV \cup \{b\}$
13: $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{b \in BV} b = 1)\}$ ▷Add cardinality constraint as hard clauses
14: $cost := cost + w_{min}$

If from now on, at each iteration i, the SAT solver includes the first clause along with $\{(x_2, m - i + 1), (\overline{x}_2, \infty)\}$ in the unsatisfiable core, then at iteration i, the formula would be:

$$\varphi_i = \{\, (x_1 \vee b_1^1 \vee \cdots \vee b_1^i, 1), (x_2 \vee b_2^1, 1), \ldots, (x_2 \vee b_2^i, 1), (x_2, m - i), (\overline{x}_2, \infty), \\ (b_1^1 + b_2^1 = 1, \infty), \ldots, (b_1^i + b_2^i = 1, \infty)\}$$

The WPM1 algorithm would need m iterations to solve the problem.

Obviously, a reasonable good SAT solver would return a better quality core than in previous example. However, unless it can guarantee that it is minimal, a similar example (but more complicated) could be constructed.

6 A Stratified Approach for WPM1

In Algorithm 3 we present a modification of the WPM1 algorithm that tries to prevent the situation described in Example 5 by carrying out a stratified approach. The main idea is to restrict the set of clauses sent to the SAT solver to force it to concentrate on those with higher weights. As a result, the SAT solver returns unsatisfiable cores with clauses with higher weights. These are better quality cores and contribute to increase the cost faster. When the SAT solver returns SAT, then we allow it to use clauses with lower weights.

In Algorithm 3 we use a variable w_{max}, and we only send to the SAT solver the clauses with weight greater or equal than it. As in Algorithm 2, we start by checking that hard clauses are satisfiable. Then, we initialize w_{max} to the

Algorithm 3. The pseudo-code of the stratified approach for WPM1 algorithm.

Input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \_)$ **then return** $(\infty, \emptyset)$
$cost := 0$ ▷Optimal
$w_{max} := \max\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}$
while *true* **do**
  $(st, \varphi_c) := SAT(\{C_i \mid (C_i, w_i) \in \varphi \wedge w_i \geq w_{max}\})$ ▷Call without weights
  if $st = \text{SAT}$ **and** $w_{max} = 0$ **then return** $(cost, \varphi)$
  else
    if $st = \text{SAT}$ **then** $w_{max} := \max\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}$
    else
      $BV := \emptyset$ ▷Blocking variables of the core
      $w_{min} := \min\{w_i \mid C_i \in \varphi_c \wedge w_i \neq \infty\}$ ▷Minimum weight
      foreach $C_i \in \varphi_c$ **do**
        if $w_i \neq \infty$ **then**
          $b :=$ new_variable()
          $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min}),\ (C_i \vee b, w_{min})\}$
          ▷Duplicate soft clauses of the core
          $BV := BV \cup \{b\}$
      $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{b \in BV} b = 1)\}$
        ▷Add cardinality constraint as hard clauses
      $cost := cost + w_{min}$

highest weight smaller than infinite. If the SAT solver returns SAT, there are two possibilities. Either w_{max} is zero (it means that we have already sent all clauses to the SAT solver) and we finish; or it is not yet zero, and we decrease w_{max} to the highest weight smaller than w_{max}, allowing the SAT solver to use clauses with smaller weights. If the SAT solver returns UNSAT, we proceed like in Algorithm 2. This algorithm was submitted to the MaxSAT evaluation 2011 as WPM1 (version 2011). It was the best performing solver for the weighted partial industrial category. The description of the solver was never published in a paper before.

We can use better strategies to decrease the value of w_{max}. Notice that, in the worst case, we could need more executions of the SAT solver than Algorithm 2, because the calls that return SAT but $w_{max} > 0$ do not contribute to increase the computed cost. Therefore, we need to find a balance between the number of those unproductive SAT calls, and the minimum weight of the cores. For example, one of the possible strategies is to decrease w_{max} until the following condition is satisfied

$$\frac{|C_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}|}{|\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}|} > \alpha$$

or $w_{max} = 0$. This strategy tends to send more new clauses to the SAT solver when they have bigger diversity of weights. In our implementation of WPM1 submitted to the MaxSAT evaluation 2012, we use this strategy, called *diversity heuristic*, with $\alpha = 1.25$.

The proof of the correctness of this algorithm is like the proof for WPM1. The only additional point is that the new algorithm is forcing the SAT solver to find some cores before others. In the proof of correctness of WPM1 there is no assumption on what cores the SAT solver finds first.

7 MaxSAT Reducibility

Our algorithms solve a MaxSAT formula by successively transforming it until we get a satisfiable formula. To prove the soundness of the algorithms it suffices to prove that these transformations preserve the cost of the formula. However, apart from this notion of *cost-preserving transformation*, we can define other (stronger) notions of formula transformation, like *MaxSAT equivalence* and *MaxSAT reducibility*.

Definition 1.
We say that φ_1 *and* φ_2 *are* **cost-equivalent** *if* $\text{cost}(\varphi_1) = \text{cost}(\varphi_2)$.
We say that φ_1 *and* φ_2 *are* **MaxSAT equivalent** *if, for any assignment* $I : var(\varphi_1) \cup var(\varphi_2) \to \{0, 1\}$, *we have* $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$.
We say that φ_1 *is* **MaxSAT reducible** *to* φ_2 *if, for any assignment* $I : var(\varphi_1) \to \{0, 1\}$, *we have* $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$.

Notice that the distinction between MaxSAT equivalence and MaxSAT reduction is the domain on the partial assignment. In one case it is $\text{var}(\varphi_1) \cup \text{var}(\varphi_2)$, and in the other $\text{var}(\varphi_1)$.

The notion of cost-preserving transformation is the weakest of all three notions, and suffices to prove the soundness of the algorithms. However, it does not allow us to replace *sub*-formulas by cost-equivalent *sub*-formulas, in other words $\text{cost}(\varphi_1) = \text{cost}(\varphi_2)$ does not imply $\text{cost}(\varphi_1 \cup \varphi_3) = \text{cost}(\varphi_2 \cup \varphi_3)$. On the other hand, the notion of MaxSAT equivalence is the strongest of all three notions, but too strong for our purposes, because the formula transformations we use does not satisfy this notion. When φ_2 has variables not occurring in φ_1, it is convenient to use the notion of MaxSAT reducibility, that, in these cases, is weaker than the notion of MaxSAT equivalence.

In the following we show some examples of the notions of Definition 1.

Example 6. The following example shows a formula transformation that preserves the cost, but not MaxSAT reducibility. Consider $\varphi_1 = \{(x, 2), (\overline{x}, 1)\}$ and $\varphi_2 = \{(\square, 1)\}$. We have $\text{cost}(\varphi_1) = \text{cost}(\varphi_2) = 1$, hence the transformation of φ_1 into φ_2 is cost-preserving. However, φ_1 is not MaxSAT reducible to φ_2, because the assignment $I : \{x\} \to \{0, 1\}$ with $I(x) = 0$, makes $\text{cost}(I(\varphi_1)) = 2 \neq 1 = \text{cost}(I(\varphi_2))$.

On the contrary, φ_2 is MaxSAT reducible to φ_1, because there is a unique assignment $I : \emptyset \to \{0, 1\}$, and it satisfies $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$. Hence, MaxSAT reducibility is not a symmetric relation.

The following example shows that MaxSAT reducibility does not imply MaxSAT equivalence. Consider $\varphi_1 = \{(x, 2), (\overline{x}, 1)\}$ and $\varphi_3 = \{(\square, 1), (x, 1), (x \vee y, 1), (\overline{x} \vee z, 1), (\overline{y} \vee \overline{z}, \infty)\}$. We have that φ_1 is MaxSAT reducible to φ_3. To prove this, we must consider two interpretations I_1 and I_2, defined by $I_1(x) = 0$ and $I_2(x) = 1$. In the first case, we obtain $I_1(\varphi_1) = \{(\square, 2)\}$ and $I_1(\varphi_3) = \{(\square, 2), (y, 1), (\overline{y} \vee \overline{z}, \infty)\}$ that have the same cost 2. In the second case, we obtain $I_2(\varphi_1) = \{(\square, 1)\}$ and $I_2(\varphi_3) = \{(\square, 1), (z, 1), (\overline{y} \vee \overline{z}, \infty)\}$ that have also the same cost 1. However, φ_1 and φ_3 are not MaxSAT equivalent because for $I : \{x, y, z\} \to \{0, 1\}$ defined by $I(x) = I(y) = I(z) = 1$ we have $\text{cost}(I(\varphi_1)) = 1 \neq \infty = \text{cost}(I(\varphi_3))$.

Finally, φ_1 is MaxSAT equivalent to $\varphi_4 = \{(\square, 1), (x, 1)\}$.

The notion of MaxSAT equivalence was implicitly defined in [7]. In this paper a MaxSAT resolution rule that preserves MaxSAT equivalence is defined, and proved complete for MaxSAT.

For lack of space we state without proof:

Lemma 1. (1) *If φ_1 is MaxSAT-reducible to φ_2 and $var(\varphi_2) \cap var(\varphi_3) \subseteq var(\varphi_1)$, then $\varphi_1 \cup \varphi_3$ is MaxSAT-reducible to $\varphi_2 \cup \varphi_3$.*
(2) *MaxSAT-reducibility is transitive: if φ_1 is MaxSAT-reducible to φ_2, φ_2 is MaxSAT-reducible to φ_3, and $var(\varphi_1) \cap var(\varphi_3) \subseteq var(\varphi_2)$, then φ_1 is MaxSAT-reducible to φ_3.*

Example 7. Notice that the side condition of Lemma 1 (1) is necessary. For instance, if we take $\varphi_1 = \{(\square, 1)\}$, $\varphi_2 = \{(x, 1), (\overline{x}, \infty)\}$ and $\varphi_3 = \{(x, 1)\}$, where the side condition $\text{var}(\varphi_2) \cap \text{var}(\varphi_3) = \{x\} \not\subseteq \emptyset = \text{var}(\varphi_1)$ is violated, we have that φ_1 is MaxSAT reducible to φ_2, but $\varphi_1 \cup \varphi_3$ is not MaxSAT reducible to $\varphi_2 \cup \varphi_3$.

Similarly, the side condition in Lemma 1 (2) is also necessary. For instance, if we take $\varphi_1 = \{(x, 1), (\overline{x}, 1)\}$, $\varphi_2 = \{(\square, 1)\}$ and $\varphi_3 = \{(x, 1), (\overline{x}, \infty)\}$, where the side condition $\text{var}(\varphi_1)) \cap \text{var}(\varphi_3) = \{x\} \not\subseteq \emptyset = \text{var}(\varphi_2)$ is also violated, we have that φ_1 is MaxSAT reducible to φ_2 and this to φ_3. However, φ_1 is not MaxSAT reducible to φ_3.

There are two side conditions in Lemma 1 (1) and (2) (see Example 7) that restrict the set of variables that can occur in the MaxSAT problems. However, if we ensure that problem transformations only *introduce fresh* variables, i.e. when φ_1 is MaxSAT reduced to φ_2, all new variables introduced in φ_2 do not occur elsewhere, then these conditions are trivially satisfied. In our algorithms, all formula transformations satisfy this restriction.

8 Generic Stratified Approach

In Algorithm 4 we show how the stratified approach can be applied to any *generic* weighted MaxSAT solver WPM. In the rest of the section we will describe what

Algorithm 4. The pseudo-code of a generic MaxSAT algorithm that follows a stratified approach heuristics.

```
   Input: φ = {(C_1, w_1), ..., (C_m, w_m)}
1: cost := 0
2: w_max = ∞
3: while true do
4:     φ_{w_max} := {(C_i, w_i) ∈ φ | w_i ≥ w_max}
5:     (cost', φ_sat, φ_res) = WPM(φ_{w_max})
6:     cost = cost + cost'
7:     if cost = ∞ or w_max = 0 then return (cost, φ_sat)
8:     W = Σ{w_i | (C_i, w_i) ∈ φ \ φ_{w_max} ∪ φ_res}
9:     φ_sat = {(C_i, harden(w_i, W)) | (C_i, w_i) ∈ φ_sat}
10:    φ = (φ \ φ_{w_max}) ∪ φ_sat ∪ φ_res
11:    w_max = decrease(w_max)
12: return (cost, φ)

13: function harden(w,W)
14: begin
15:    if w > W then return ∞
16:    else return w
```

properties the generic algorithm WPM has to satisfy in order to ensure the correctness of this approach.

We assume that, given a weighted MaxSAT formula φ, $WPM(\varphi)$ returns a triplet $(cost, \varphi_{sat}, \varphi_{res})$ such that φ is MaxSAT reducible to $\{(\square, cost)\} \cup \varphi_{sat} \cup \varphi_{res}$, φ_{sat} is satisfiable (has cost zero), and clauses of φ_{res} have cost strictly smaller than w_{max}. Given φ, WPM1 return a pair $(cost, \varphi')$ where φ is MaxSAT reducible to $\{(\square, cost)\} \cup \varphi'$ and φ is satisfiable, hence satisfies the requirements taking $\varphi_{res} = \emptyset$. Moreover, we can also think of WPM as an algorithm that *partially* solves the formula, and returns a lower bound cost, a satisfiable part of the formula φ_{sat}, and an unsolved residual φ_{res}.

The algorithm uses a variable w_{max} to restrict the clauses sent to the MaxSAT solver. The first time $w_{max} = \infty$, and we run WPM only on the hard clauses. Then, in each iteration we send clauses with weight w_{max} or bigger to WPM. We add the return cost to the current cost, and decrease w_{max}, until w_{max} is zero.

Algorithm 3 is an instance of this generic schema where WPM is a partial execution of WPM1 where clauses generated during duplication with weight smaller than w_{max} are put apart in φ_{res}.

Lines 8 and 9 are optional and can be removed from the algorithm without affecting to its correctness. They are inspired in [15]. The idea is to harden all soft clauses with weight bigger than the sum of the weights of the clauses not sent to the WPM plus the clauses returned in φ_{res}. The proof of the correctness of these lines is based in the following lemma (not proved for lack of space).

Lemma 2. *Let* $\varphi_1 = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ *be a satisfiable MaxSAT formula,* $\varphi_2 = \{(C'_1, w'_1), \ldots, (C'_r, w'_r)\}$ *be a MaxSAT formula without hard clauses and* $W = \sum_{j=1}^{r} w'_j$*. Let*

$$harden(w) = \begin{cases} w & \text{if } w \leq W \\ \infty & \text{if } w > W \end{cases}$$

and $\varphi'_1 = \{(C_i, harden(w_i)) \mid (C_i, w_i) \in \varphi_1\}$*. Then* $\text{cost}(\varphi_1 \cup \varphi_2) = \text{cost}(\varphi'_1 \cup \varphi_2)$.

Notice that we check the applicability of this lemma dynamically, recomputing the value W in every iteration in line 8 of Algorithm 4.

Theorem 1. *Assuming that WPM, given a formula* φ*, returns a triplet* $(cost, \varphi_{sat}, \varphi_{res})$ *such that* φ *is MaxSAT reducible to* $\{(\square, cost)\} \cup \varphi_{sat} \cup \varphi_{res}$*,* φ_{sat} *is satisfiable, and* φ_{res} *only contain clauses with weight strictly smaller than* w_{max}*, Algorithm 4 is a correct algorithm for Weighted Partial MaxSAT. Moreover, when for a formula* φ*, the algorithm returns* (c, φ')*, then* $c = \text{cost}(\varphi)$ *and any assignment satisfying* φ' *is an optimal assignment of* φ*.*

9 Experimental Results

We conducted our experimentation on the same environment as the MaxSAT evaluation [4] (processor 2 GHz). We increased the timeout from half hour to two hours, and the memory limit from 0.5G to 1G. The solvers that implement our Weighted Partial MaxSAT algorithms are built on top of the SAT solver picosat (v.924) [6]. The solver *wpm1* implements the original WPM1 algorithm [1]. The cardinality constraints introduced by WPM1 are translated into SAT through the regular encoding [3]. This encoding assures a linear complexity on the size of the cardinality constraint. This is particularly important for the last queries where the size of the cores can be potentially close to the number of soft clauses. We use the subscript b to indicate that we break symmetries as described in section 4, s to indicate we apply the stratified approach and d to indicate that we apply the diversity heuristic to compute the next w_{max}, both described in section 6. *wpm1* was the solver submitted to the 2009 and 2010 MaxSAT evaluations, and $wmp1_s$ the one submitted to the 2011 evaluation. The hardening soft clauses (lines 8 and 9 in Algorithm 4) had not impact in our implementations' performance.

In the following we present results for the benchmarks of the Weighted Partial MaxSAT categories of the MaxSAT 2011 evaluation. We compare our solvers with the best three solvers of the evaluation, and other solvers which did not compete but have been reported to exhibit good performance, such as, *binc* and *bincd* [11], maxhs [8] and the Weighted CSP solver toulbar2 [19].

We present the experimental results following the same classification criteria as in the MaxSAT evaluation. For each solver and set of instances, we present the number of solved instances in parenthesis and the mean time required to solve them. Solvers are ordered from left to right according to the total number

Table 1. Experimental results

set	#	wpm1$_{bsd}$	wpm1$_{bs}$	wpm1$_{s}$	wpm1$_{b}$	wbo1.6	wpm1	wpm2	maxhs	bincd	maxhs	sat4j	binc	toulbar2
haplotyping	100	**423(95)**	425(95)	378(88)	376(79)	93.4(72)	390(65)	350(42)	1043(34)	196(21)	1247(35)	108(20)	408(27)	383(5)
timetabling	26	685.60(9)	**683(9)**	585(8)	992(9)	776(4)	1002(7)	707(9)	1238(4)	766(6)	2388(5)	0(0)	1350(4)	0(0)
upgradeability	100	35.8(100)	37.2(100)	36.5(100)	36.9(100)	63.3(100)	114(100)	311(100)	**29(100)**	637(78)	28.4(50)	844(30)	0(0)	0(0)
Total	226	**204**	204	196	188	176	172	151	138	105	90	50	31	5

(a) Weighted Partial - Industrial

set	#	wpm1$_{bsd}$	incw maxsatz	ak maxsat	toulbar2	wpm1$_{bs}$	wmax satz09z	maxhs	sat4j	wpm1$_{s}$	bincd	binc	wpm1	wbo1.6	wpm1$_{b}$
auc-paths	86	274(53)	7.59(86)	**4.6(86)**	28.8(86)	332(53)	570(80)	72.2(86)	994(44)	22(33)	1828(2)	0(0)	0(0)	0(0)	0(0)
auc-sched	84	**11.9(84)**	220(84)	123(84)	133(84)	12.2(84)	92(84)	1125(69)	716(80)	7.6(80)	130(50)	103(45)	0(0)	0(0)	0(0)
planning	56	27.9(52)	92.2(38)	354(40)	149(41)	**26.3(56)**	220(50)	306(29)	3.27(55)	12.5(54)	59.5(47)	51.1(46)	1.46(28)	1.33(30)	3.63(29)
warehouses	18	44(14)	1184(18)	37(2)	0.03(1)	571(3)	0.32(1)	0.37(1)	1.34(1)	1644(3)	7.03(1)	8.67(1)	**4.23(18)**	0.51(4)	0.88(12)
miplib	12	1187(4)	**1419(5)**	0.47(2)	63.2(3)	1165(4)	266(3)	0.07(1)	693(4)	34(3)	618(3)	699(3)	0.21(1)	0(0)	1507(1)
random-net	74	**241(39)**	1177(1)	1570(2)	0(0)	0(0)	0(0)	2790(6)	0(0)	0(0)	0(0)	0(0)	615(27)	63(37)	439(12)
spot5dir	21	257(10)	1127(5)	1106(5)	217(5)	383(10)	11.5(2)	199(6)	1.95(2)	1.41(5)	**66.7(11)**	51.7(6)	1.03(4)	2.60(5)	12.8(6)
spot5log	21	**532(14)**	0.63(4)	200(5)	170(5)	574(14)	15.6(2)	710(6)	6.04(3)	44.4(6)	124(11)	79.3(7)	21.5(6)	25.5(6)	131(7)
Total	372	**270**	241	226	225	224	222	204	189	184	125	108	84	82	67

(b) Weighted Partial - Crafted

Instance set	#	maxhs	wbo1.6	wpm1$_{b}$	wpm1$_{bsd}$	wpm1	wpm1$_{bs}$	wpm1$_{s}$	bincd	sat4j	binc	toulbar2
Table4 [8]	13	**4.41(13)**	5.03(13)	5.54(13)	8.84(13)	18.92(13)	534.13(13)	559.23(13)	56.69(11)	3485.52(6)	19.50(1)	0.00(0)
Total	13	**13**	13	13	13	13	13	13	11	6	1	0

(c) Table 4 from [8]. Linux upgradibility family forcing diversity of weights

of instances they solved. We present in bold the results for the best performing solver in each set. '# ' stands for number of instances of the given set.

Table 1(a) presents the results for the industrial instances of the Weighted Partial MaxSAT category. As we can see, our original solver *wpm1* would have ranked as the second best solver after *wbo1.6*. By breaking symmetries ($wpm1_b$) we solve 12 more instances than *wbo1.6*, and 20 more if we apply the stratified approach. Combining both, we solve 28 more instances. The addition of the diversity heuristic to the stratified approach has no impact for the instances of this category. We do not present any result on branch and bound based solvers since they typically do not perform well on industrial instances.

Table 1(b) presents the results for the crafted instances of the Weighted Partial MaxSAT category. The best ranked solvers in this category for the MaxSAT 2011 evaluation were: *incwmaxsatz*, *akmaxsat* and *wmaxsatz09*, in this order. All are branch and bound based solvers, which typically dominate the crafted and random categories. We can see that our solver *wpm1* shows a poor performance in this category. However, by applying the stratified approach ($wpm1_s$) we jump from 84 solved instances to 184. If we also break symmetries ($wpm1_{bs}$) we solve 224 instances, ranking as the third best solver respect to the participants of the MaxSAT 2011 evaluation, very close to *akmaxsat*. If we compare carefully the results of *wpm1* and $wpm1_{bs}$, we notice that there are two sets where *wpm1* behaves much better (warehouses and random-net). This suggests that we must make our stratified approach more flexible, for example, by incorporating the diversity heuristic ($wpm1_{bsd}$). Using $wpm1_{bsd}$ we solve up to 270 instances, outperforming all the branch and bound solvers.

In [8] it is pointed out that instances with a great diversity of weights can be a bottleneck for some Weighted MaxSAT solvers. To test this hypothesis they generate 13 instances from the Linux upgradibility set in the Weighted Partial MaxSAT industrial category preserving the underlying CNF but modifying the weights to force a greater diversity. We have reproduced that experiment in Table 1(c). As we can see, *wpm1* compares well to the best performing solvers, and by breaking symmetries ($wpm1_b$) we reach the performance of *maxhs* and *wbo1.6*. On the other hand, the stratified approach impacts negatively ($wpm1_s$ or $wpm1_{bs}$), but the diversity heuristic fixes this problem.

Taking into consideration the experimental results obtained in the different categories, we can see that our approach $wpm1_{bsd}$ is the most robust solver for Weighted Partial MaxSAT instances. We also checked the effectiveness of breaking symmetries for Unweighted Partial MaxSAT instances. For industrial instances we improve from 181 to 262 solved instances, and for crafted from 55 to 115.

References

1. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 427–440. Springer, Heidelberg (2009)
2. Ansótegui, C., Bonet, M.L., Levy, J.: A new algorithm for weighted partial MaxSAT. In: AAAI 2010 (2010)

3. Ansótegui, C., Manyà, F.: Mapping Problems with Finite-Domain Variables to Problems with Boolean Variables. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 1–15. Springer, Heidelberg (2005)
4. Argelich, J., Li, C.M., Manyà, F., Planes, J.: MaxSAT evaluations (2006, 2007, 2008, 2009, 2010, 2011), http://www.maxsat.udl.cat
5. Berre, D.L.: SAT4J, a satisfiability library for java (2006), http://www.sat4j.org
6. Biere, A.: PicoSAT essentials. Journal on Satisfiability 4, 75–97 (2008)
7. Bonet, M.L., Levy, J., Manyà, F.: Resolution for Max-SAT. Artif. Intell. 171(8-9), 606–618 (2007)
8. Davies, J., Bacchus, F.: Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 225–239. Springer, Heidelberg (2011)
9. Fu, Z., Malik, S.: On Solving the Partial MAX-SAT Problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006)
10. Heras, F., Larrosa, J., Oliveras, A.: MiniMaxSat: A New Weighted Max-SAT Solver. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 41–55. Springer, Heidelberg (2007)
11. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: AAAI 2011 (2011)
12. Li, C.M., Manyà, F., Mohamedou, N., Planes, J.: Exploiting Cycle Structures in Max-SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 467–480. Springer, Heidelberg (2009)
13. Lin, H., Su, K., Li, C.M.: Within-problem learning for efficient lower bound computation in Max-SAT solving. In: AAAI 2008, pp. 351–356 (2008)
14. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for Weighted Boolean Optimization. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 495–508. Springer, Heidelberg (2009)
15. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Ann. Math. Artif. Intell. 62(3-4), 317–343 (2011)
16. Marques-Silva, J., Lynce, I., Manquinho, V.: Symmetry Breaking for Maximum Satisfiability. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 1–15. Springer, Heidelberg (2008)
17. Marques-Silva, J., Manquinho, V.: Towards More Effective Unsatisfiability-Based Maximum Satisfiability Algorithms. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 225–230. Springer, Heidelberg (2008)
18. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. CoRR abs/0712.1097 (2007)
19. Sanchez, M., Bouveret, S., Givry, S.D., Heras, F., Jégou, P., Larrosa, J., Ndiaye, S., Rollon, E., Schiex, T., Terrioux, C., Verfaillie, G., Zytnicki, M.: Max-CSP competition 2008: toulbar2 solver description (2008)