On Solving MaxSAT Through SAT

Carlos Ansótegui a,1 María Luisa Bonet b Jordi Levy c
a Universitat de Lleida (DIEI, UdL)
b Universitat Politècnica de Catalunya (LSI, UPC)
c Artificial Intelligence Research Institute (IIIA, CSIC)

Abstract. In this paper we present a Partial MaxSAT solver based on successive calls to a SAT solver. At the kth iteration the SAT solver tries to certify that there exist an assignment that satisfies all but k clauses. The key idea is to add an additional variable to each soft clause and to introduce, at each iteration, at-least and at-most cardinality constraints restricting the possible values of these variables. We prove the correctness of our algorithm and compare it with other (Partial) MaxSAT solvers.

Keywords. Satisfiability, MaxSAT, optimization

Introduction

SAT solvers only produce solutions (assignments) for satisfiable problems. Partial MaxSAT solvers produce solutions, also for unsatisfiable problems, that satisfy the maximum number of soft clauses and all hard clauses, or in other words, that violate the minimum number of soft clauses and none of the hard clauses.

A well known strategy to solve these problems is the usage of MaxSAT solvers that exploit the efficiency accomplished by modern SAT solvers by solving MaxSAT through successive calls to a SAT solver. This can be done by extending every soft clause $C_i$ with a fresh auxiliary variable: $C_i \lor b_i$, and adding an at-most cardinality constraint $\sum b_i \leq k$ on these variables that states that not more than $k$ of them can be true. If the resulting SAT formula is satisfiable for $k = m$, but not for $k = m - 1$, then $m$ is the optimum, i.e., the minimum number of soft clauses that can be violated.

We can search for the value $m$ in several ways. An obvious way is a binary search between zero and the number of soft clauses, where the SAT solver is called at each step with the corresponding cardinality constraint. SAT4J [3] starts with $k$ equal to the number of soft clauses. If the SAT solver reports unsatisfiable, $k$ is the optimum. Otherwise, it calls again the SAT solver with $k$ equal to the number of auxiliary variables set to true in the previous satisfying assignment minus one.

Alternatively, we can start with $k = 0$ and increase this value until the SAT solver reports satisfiable. If the SAT solver is able to return an unsatisfiable core when it reports unsatisfiable, then, we can use this information for establishing additional cardinality constraints on the auxiliary variables of the clauses belonging to the core.

1Corresponding Author: Departament d’Informàtica i Enginyeria Industrial, Jaume II 69, 25001 Lleida, Spain; E-mail: carlos@diei.udl.cat
input: $\varphi = \{C_1, \ldots, C_m\}$

\[ \text{cost} := 0 \]

while true do

\[(st, \varphi_c) := \text{SAT}(\varphi)\]

if $st = \text{SAT}$ then return cost

$BV := \emptyset$

for each $C \in \varphi_c$ do

if $C$ is soft then

$b := $ new blocking variable

$\varphi := \varphi \setminus \{C\} \cup \{C \lor b\}$

$BV := BV \cup \{b\}$

if $BV = \emptyset$ then return UNSAT

$\varphi := \varphi \cup \text{CNF}(\sum_{b \in BV} b = 1)$

$\text{cost} := \text{cost} + 1$

Optimal

Call to the SAT solver

Set of blocking variables

Add blocking variable

There are no soft clauses in the core

Add cardinality constraint as hard clauses

Figure 1. The pseudo-code of the FU&MALIK algorithm.

Fu and Malik [5,6] (see Figure 1), and its implementation msu1.2 [7,8], describe in essence an algorithm based on this approach. On each iteration, they add an auxiliary variable to each clause involved in the new obtained unsatisfiable core and a cardinality constraint stating that the sum of these new variables is equal to one. Therefore, in this algorithm, a clause can be extended with more than one auxiliary variable. This can hamper the efficiency of the SAT solver. In [1], we show how we can replace, in the Fu&Malik algorithm, the sequence of auxiliary variables of each clause with just one. In [9], it is described an algorithm that alternates phases in which the SAT solvers reports satisfiable with other in which it reports unsatisfiable. It also uses a unique auxiliary variable per clause, and exploits the information of the unsatisfiable core.

In this paper we extend the work of [1] showing how to exploit better the information given by the unsatisfiable cores to add richer cardinality constraints on the auxiliary variables.

1. Preliminaries

In the Partial MaxSAT context we work with two sets of clauses, hard and soft. The Partial MaxSAT problem for a multiset of clauses is the problem of finding an optimal assignment to the variables that satisfies all the hard clauses, and the maximum number of soft clauses. The number of soft clause falsified by an assignment is the cost of this assignment. The cost of the optimal assignment of a formula $F$ is called the cost of the formula, and is denoted by $MaxSAT(F)$. Our approach is based on successive calls to a SAT solver. The SAT solver may return a set of clauses that is unsatisfiable. We call this set unsatisfiable core.

To prove the correctness of our algorithm we will prove that it can simulate the Fu&Malik algorithm. The Fu and Malik [6] algorithm for Partial MaxSAT consists in iteratively calling a SAT solver on a working formula $\varphi$. This corresponds to the line $(st, \varphi_c) := \text{SAT}(\varphi_w)$ in the code of Figure 1. 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 ($\varphi_c$). At this point the algorithm will produce new variables, blocking variables ($BV$ in the code), one for each clause. The new working formula $\varphi$ will consist in adding the new variables to the formulas of the core, adding a cardinality constraint saying that exactly one of the new variables should be true ($\text{CNF}(\sum_{b \in BV} b = 1)$ in the code), and adding one to the counter of falsified clauses. This procedure is applied until the SAT solver returns satisfiable.

2. A Partial MaxSAT Algorithm

The next algorithm, that we call PM2, is a variant of the algorithm described in [1]. It is based on the use of cores and covers. Basically, every core will result into an at-least cardinality constraint, and every cover into an at-most cardinality constraint.

At the beginning of the algorithm, every soft clause $C_i$ is extended with a fresh auxiliary blocking variable $b_i$. This variable is different for every soft clause, therefore its index $i$ can be used as an identifier of the clause (we assume that the first $i = 1, \ldots, m$ clauses are soft). The presence of hard clauses in an unsatisfiable core is irrelevant. In fact, they are removed from the core by the algorithm. Therefore, when we say a core we mean a subset of indexes of blocking variables, those that protect the soft clauses contained in the core (see how $B_1$ is calculated in Figure 2). Covers are also subsets of indexes of blocking variables, defined as follows.

**Definition 1.** Given a set of cores $L$, where every core $B_1 \in L$ is a set of indexes, we say that the set of indexes $B_2$ is a cover of $L$, if it is a minimal set such that, for every $B_1 \in L$, if $B_1 \cap B_2 \neq \emptyset$, then $B_1 \subseteq B_2$.

Given a set of cores $L$, we denote the set of covers of $L$ as $\text{coversof}(L)$.

**Lemma 2.** $\text{coversof}(L)$ is a partition of the set of variables of $BV$ that belong to some core of $L$, therefore covers do not intersect.

Moreover, every core of $L$ is included into just one cover of $\text{coversof}(L)$.

PM2 works as follows (see Figure 2): every clause $C_i$ is extended with $b_i$ and we add a clause $-b_i$ saying that all these additional variables have to be false when they do not occur in any core. Also before the first iteration of the algorithm the counter of falsified clauses, cost, is set to zero. We have a list of cores $L_1$ that is increased with a new core in each iteration, and a list of covers $L_2$ that is re-calculated in each iteration from the list of cores.\(^2\) At every iteration of the algorithm a SAT solver is called. As before, if the solver returns unsatisfiable, it also gives an unsatisfiable core. If the core only contains hard clauses, then the algorithm returns unsatisfiable. Otherwise, we put the blocking variables of the soft clauses of the core in a set $B_1$. Since we have found a new unsatisfiable core, variable cost gets increased by one. Also we look for other cores such that the soft clauses are included in the new core. If no such core exists, we add an at-least cardinality constraint saying that the sum of the blocking variables of $B_1$ is larger than or equal to one. If some cores are included, we add the cardinality constraint saying that the number of variables in $B_1$ that need to be one is at least the number of

\(^2\)In the implementation of the algorithm, this re-calculation is done incrementally from the set used in the previous iteration and the new core, but for the sake of simplicity, we assume that it is re-calculated from scratch.
Figure 2. The pseudo-code of the PM2 algorithm.

cores included in the last core found (counting the last). Then, we calculate the set of covers $L_2$ that cover the set of cores $L_1$, as we will see later. Finally, we add, for every cover $B_2$, an at-most cardinality constraint saying that the sum of all blocking variables of the cover is at most equal to the number of cores contained in the cover. If the solver says that the formula is satisfiable, the algorithm returns $cost$ as the minimal number of falsified clauses.

PM2 simplifies Fu&Malik in the sense that it only adds one blocking variable per clause. Intuitively, this would have to result into a more efficient algorithm because there are less blocking variables, so the SAT solver will have to check less possible assignments. This idea is already used in other MaxSAT solvers, like SAT4J [3], msu3 [8] and msu4-0 [9]. In SAT4J only one at-most cardinality constraint (saying that the sum of blocking variables is smaller than $k$) is used. This bound $k$ is reduced until the SAT solvers says unsatisfiable. In msu3 [8], in a first phase they compute a maximal set of disjoint cores, and in a second phase they do as in SAT4J but increasing the bound $k$ (starting with the number of disjoint cores) until the SAT solver returns sat, and only summing the blocking variables that have appeared in some core. Finally, in the msu4-0 algorithm [9], apart from the at-most constraint, they also use some at-least constraints saying that blocking variables occurring in a core, and not occurring in previous cores, have to sum at least one. The algorithm alternates phases where the SAT solver returns
sat or unsat, refining a lower or upper bound, and only terminates when the upper and lower bound coincide, or when the new core does not contain new blocking variables. Our approach is different from previous ones in two senses. First, we can have several at-most constraints. In particular, if cores are disjoint, we will have an at-least and an at-most constraint for each core. Second, our at-least constraints may impose a bound strictly greater than one, in contrast with the msu4.0 algorithm. These differences would have to result in a more restrictive constraint, thus in fewer assignments to check by the SAT solver.

To prove that the PM2 algorithm is correct, we will prove that the Fu&Malik algorithm can simulate it. We have to be aware they are non-deterministic, since we assume that the SAT solver returns an unsatisfiable core non-deterministically. However, recall that we have proved that Fu&Malik algorithm is correct for every possible run. The proof is by induction on the number of execution steps. From now on, when we say that a set of soft clauses is a core, we mean that this set, together with the hard clauses, is a core. A core \( B = \{ i_1, \ldots, i_m \} \) will be a set of indexes of soft clauses. Suppose that Fu&Malik has simulated PM2 for \( s \) steps. We will prove that 1) if PM2 finds a core \( B \), then this set \( B \) of (soft) clauses is also a core for the Fu&Malik algorithm (see Lemma 4), in particular if the set \( B = \emptyset \) is a core for PM2, and it returns UNSAT, then the same set \( B = \emptyset \) is a core for Fu&Malik, that also returns UNSAT; and 2) if PM2 does not find any cores, and stops, then Fu&Malik does not find any cores either (see Lemma 5), and also stops returning the same MaxSAT value, since both have run the same number of steps.

Since Theorem 6 will be proved by induction, assume by induction hypothesis that

\[
\varphi = \{ C_1 \lor a_1, \ldots, C_m \lor a_m \} \cup \bigcup_{B_r \in \{ B_1, \ldots, B_s \}} \text{CNF} \left( \sum_{i \in B_r} a_i \geq |\{ B_j \mid j \leq r \land B_j \subseteq B_r \}| \right) \\
\cup \bigcup_{B' \in \text{coversoft} \{ B_1, \ldots, B_s \}} \text{CNF} \left( \sum_{i \in B'} a_i \leq |\{ B_j \mid j \leq s \land B_j \subseteq B' \}| \right) \\
\cup \{ -b_i \mid \forall r = 1, \ldots, s \ i \notin B_r \}
\]

is the formula computed by PM2 after \( s \) execution steps, where \( B_1, \ldots, B_s \) is the sequence of cores. Assume also that Fu&Malik, after \( s \) steps simulating PM2, with the same sequence of cores \( B_1, \ldots, B_s \), obtains the formula

\[
\hat{\varphi} = \{ C_1 \lor \bigvee_{1 \in B_j} b^1_j, \ldots, C_n \lor \bigvee_{n \in B_j} b^n_j \} \cup \bigcup_{j=1}^{s} \text{CNF} \left( \sum_{i \in B_j} b^i_j = 1 \right)
\]

Notice that, if \( i \in B_j \), then \( b^i_j \) is the variable added by the Fu&Malik algorithm to clause \( i \), at iteration \( j \).

Lemma 3. Let \( \varphi \) and \( \hat{\varphi} \) be the formulas obtained by PM2 and Fu&Malik algorithms, respectively, after \( s \) steps of simulation. For any optimal interpretation \( \hat{I} \) of the variables of \( \hat{\varphi} \), let \( I \) be the interpretation of the variables of \( \varphi \) given by

\[
I(x) = \hat{I}(x) \quad \text{for any variable } x \in \{ C_1, \ldots, C_n \}
I(a_i) = \max \{ \hat{I}(b^i_j) \mid i \in B_j \land j = 1, \ldots, s \} \quad \text{for the blocking variables}
\]
Then, (1) if \( \hat{I} \) satisfies the hard clause \( C \in \hat{\varphi} \), then \( I \) satisfies the hard clause \( C \in \varphi \); (2) if \( \hat{I} \) satisfies the cardinality constraints of \( \hat{\varphi} \), then \( I \) satisfies the cardinality constraints of \( \varphi \); and (3) if \( \hat{I} \) satisfies the soft clause \( C_i \vee \bigvee_{i \in B_j} b^j_i \in \hat{\varphi} \), then \( I \) satisfies the soft clause \( C_i \vee a_i \in \varphi \).

**Proof.** Statement (1) is trivial, since \( I \) and \( \hat{I} \) assign the same values to the original variables.

For (2), since the interpretation \( \hat{I} \) is optimal, it means that it assigns \textit{true} to at most one of the blocking variables of a clause. In other words, for any clause \( i = 1, \ldots, m \), we have \( \sum \{ \hat{I}(b^j_i) \mid j = 1, \ldots, s \wedge i \in B_j \} \leq 1 \). Therefore, the following maximal value and the sum coincides and we have

\[
I(a_i) = \max \{ \hat{I}(b^j_i) \mid j = 1, \ldots, s \wedge i \in B_j \} = \sum_{j \in B_j} \hat{I}(b^j_i)
\]

Now, if \( \hat{I} \) satisfies the cardinality constraints of \( \hat{\varphi} \), then \( \sum_{i \in B_r} \hat{I}(b^r_i) = 1 \), for any core \( B_r \). Hence, for any core \( B_r \), we have

\[
\sum_{i \in B_r} I(a_i) = \sum_{i \in B_r} \sum_{j \in B_j} \hat{I}(b^j_i) \geq \sum_{B_j \subseteq B_r} \sum_{i \in B_j} \hat{I}(b^j_i) = \sum_{B_j \subseteq B_r} 1 = |\{ B_j \mid j \leq r \wedge B_j \subseteq B_r \}|
\]

Hence, for any \( r = 1, \ldots, s \), \( I \) satisfies the at-least cardinality constraints

\[
\text{CNF} \left( \sum_{i \in B_r} a_i \geq |\{ B_j \mid j \leq r \wedge B_j \subseteq B_r \}| \right)
\]

Now, for any cover \( B' \in \text{coversof} \{ B_1, \ldots, B_s \} \) we have

\[
\sum_{i \in B'} I(a_i) = \sum_{i \in B'} \sum_{j \in B_j} \hat{I}(b^j_i) = \sum_{i \in B'} \sum_{B_j \subseteq B'} \hat{I}(b^j_i) = \sum_{B_j \subseteq B'} \sum_{i \in B_j} \hat{I}(b^j_i) = \sum_{B_j \subseteq B'} 1 = |\{ B_j \mid j = 1, \ldots, s \wedge B_j \subseteq B' \}|
\]

because, by definition of cover, \( i \in B' \) and \( i \in B_j \) implies \( B_j \subseteq B' \).

Hence, for any cover \( B' \in \text{coversof} \{ B_1, \ldots, B_s \} \), \( I \) satisfies the at-most cardinality constraints

\[
\text{CNF} \left( \sum_{i \in B'} a_i \geq |\{ B_j \mid j \leq s \wedge B_j \subseteq B' \}| \right)
\]

For (3), if \( \hat{I} \) satisfies \( C_i \vee \bigvee_{i \in B_j} b^j_i \), then either it satisfies \( C_i \), and so \( I \), because they assign the same values to original variables, or \( \hat{I} \) satisfies some of the variables \( b^j_i \). In this second case, the way we define the value of \( a_i \), ensures that \( I \) satisfies this value, hence the clause \( C_i \vee a_i \in \varphi \). \( \square \)

**Lemma 4.** Let \( \varphi \) and \( \hat{\varphi} \) be the formulas obtained by PM2 and FU&MALIK algorithms, respectively, after \( s \) steps of simulation. If \( B \) is a core of \( \varphi \), then \( B \) is also a core of \( \hat{\varphi} \).
Proof. If \( B \) is not a core of \( \hat{\varphi} \), then there exists an interpretation \( \hat{I} \) of \( \hat{\varphi} \) that satisfies all hard clauses, all cardinality constraints of \( \hat{\varphi} \), and all soft clauses \( C_i \lor \bigvee_{i \in B} b_i^i \) where \( i \in B \). By Lemma 3, \( I \) will satisfy all hard clauses, all cardinality constraints of \( \varphi \), and all soft clauses \( C_i \lor a_i \) where \( i \in B \). This would contradict that \( B \) is a core of \( \varphi \). \( \square \)

The previous lemma ensures that if PM2 finds a core, then FU&MALIK also finds a core. Hence, if PM2 does not stop, then FU&MALIK does not stop, either. Therefore, the values computed by PM2 are smaller than the values calculated by FU&MALIK. However, it is still possible that PM2 computes underestimated values of MaxSAT of a formula. The following lemma shows that this is not the case. Notice that the proof of this lemma relies on the correctness of the FU&MALIK algorithm.

**Lemma 5.** Let \( \varphi \) and \( \hat{\varphi} \) be the formulas obtained by PM2 and FU&MALIK algorithms, respectively, after \( s \) steps of simulation. If \( \varphi \) is satisfiable, then \( \hat{\varphi} \) is satisfiable.

**Proof.** Let \( I \) be an interpretation satisfying \( \varphi \). In particular, \( I \) satisfies \( \text{CNF}(\sum_{i=1}^{m} a_i \leq s) \), and all hard and soft clauses. Therefore, \( I \) satisfies all the original soft clauses \( C_i \) where \( I(a_i) = \text{false} \), and there are at least \( n - s \) of such clauses. We have \( \text{MaxSAT}(C_1, \ldots, C_n) \leq s \). Since, FU&MALIK is a correct algorithm for MaxSAT, it has to stop before \( s \) or less execution steps. And, since it has not finished before, it has to finish after these \( s \) steps, hence \( \hat{\varphi} \) is satisfiable. \( \square \)

**Theorem 6.** PM2 is a correct algorithm for Partial MaxSAT.

3. Experimental Results

In order to conduct our experimental investigation we have selected the benchmarks submitted to the MaxSAT’08 evaluation [2]. We have focus on the crafted and industrial instances for the unweighted MaxSAT and partial MaxSAT categories. The appropriate testing instances for our algorithms would be the industrial instances, where we can expect these approaches to be competitive. However, since there is a lack of industrial instances, we decided also to incorporate the crafted instances.

Our experiments have been run on the same machine specs as the MaxSAT evaluation; AMD Opteron 248 Processor, 2 GHz with 1 GB of RAM.

The solvers we compare are the best solvers for each category and benchmark at the MaxSAT’08 evaluation [2], our solver PM2.0 [1], and the new version introduced in this paper PM2.1.

Our solvers are implemented on top of the SAT solver picosat846 [4], although they can be easily adapted to work with any other solver that provides an interface to access to the unsatisfiable core when the formula is UNSAT. In order to encode the cardinality constraints we use the encoding based on sequential counters presented in [10].

Table 1 show the results of our experimental investigation. We set as timeout 1200 seconds. We report the number of solved instances (within parenthesis), and the mean time of the solved instances for each solver. The rules at the MaxSAT’08 evaluation [2] establish that the winner is the solver which solves more instances and ties are broken by selecting the solver with the minimum mean time. In bold we present the results of the winners.
<table>
<thead>
<tr>
<th>set</th>
<th>#</th>
<th>Best in MaxSAT eval.'08</th>
<th>PM2.0</th>
<th>PM2.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unweighted MaxSAT Category</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crafted</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maxcut/dimacs_mod/</td>
<td>62</td>
<td>IncMaxSatz - 81.8(52)</td>
<td>175(7)</td>
<td>74 (10)</td>
</tr>
<tr>
<td>Maxcut/random/</td>
<td>58</td>
<td>MaxSat - 4.5(40)</td>
<td>-(0)</td>
<td>-(0)</td>
</tr>
<tr>
<td>Maxcut/Spinglass/</td>
<td>5</td>
<td>MiniMaxSat - 1.62(3)</td>
<td>102.5(2)</td>
<td>4 (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Industrial</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SeanSafarpour</td>
<td>112</td>
<td>msu1.2 - 57.5(72)</td>
<td>90.2(75)</td>
<td>60 (78)</td>
</tr>
<tr>
<td>Partial MaxSAT Category</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crafted</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maxclique/Random/</td>
<td>96</td>
<td>MiniMaxSAT - 2.4(96)</td>
<td>-(0)</td>
<td>126 (54)</td>
</tr>
<tr>
<td>Maxclique/Structured/</td>
<td>62</td>
<td>MiniMaxSAT - 73(36)</td>
<td>32.6(6)</td>
<td>62(12)</td>
</tr>
<tr>
<td>Maxone/3SAT/</td>
<td>80</td>
<td>IncMaxSatz - 0.46(80)</td>
<td>105.7(79)</td>
<td>22(80)</td>
</tr>
<tr>
<td>Maxone/Structured/</td>
<td>60</td>
<td>SAT4J - 10.1(60)</td>
<td>547.5(13)</td>
<td>253(34)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Industrial</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bcp-fir/</td>
<td>59</td>
<td>msu1.2 - 49.2(46)</td>
<td>67.4(56)</td>
<td>18(58)</td>
</tr>
<tr>
<td>Bcp-hipp+yral1/</td>
<td>1183</td>
<td>SAT4J - 19.2(1111)</td>
<td>13.5(1162)</td>
<td>6(1163)</td>
</tr>
<tr>
<td>Bcp-msp/</td>
<td>148</td>
<td>MiniMaxSAT - 48.9(104)</td>
<td>384.2(36)</td>
<td>106(94)</td>
</tr>
<tr>
<td>Bcp-mtg/</td>
<td>215</td>
<td>MiniMaxSAT - 25.7(206)</td>
<td>10.5(214)</td>
<td>1.3(215)</td>
</tr>
<tr>
<td>Bcp-syn/</td>
<td>74</td>
<td>lb-psat - 63.4(34)</td>
<td>71.2(34)</td>
<td>14 (38)</td>
</tr>
<tr>
<td>Pbo-mqc-nencdr/</td>
<td>128</td>
<td>msu4.0 - 167.5(115)</td>
<td>142(78)</td>
<td>125(84)</td>
</tr>
<tr>
<td>Pbo-mqc-nolencdr/</td>
<td>128</td>
<td>msu4.0 - 111(128)</td>
<td>140.3(97)</td>
<td>130(106)</td>
</tr>
<tr>
<td>Pbo-rout/</td>
<td>15</td>
<td>msu1.2 - 2.9(15)</td>
<td>24.7(15)</td>
<td>1.35(15)</td>
</tr>
</tbody>
</table>

Table 1. Comparison of performance of Partial MaxSAT solvers. Time in seconds and number of instances solved between parenthesis. The timeout was 1200 seconds and # stands for the number of instances of the benchmark. PM2.0 is the version of PM2 described in [1], and PM2.1 the new version introduced in this paper.

As we can see our new solver PM2.1 always outperforms PM2.0, and it is able to win in six families. As we pointed out, our solver does not perform well on crafted instances. As a whole, we can say that there is not a clear winner in all the categories, and so far, all the approaches can have some potential.

Acknowledgements

Research partially supported by the projects TIN2007-68005-C04-(01,03,04) and TIN2006-15662-C02-02 funded by the MEC.
References


