Solving (Weighted) Partial MaxSAT through Satisfiability Testing*

Carlos Ansótegui[1], María Luisa Bonet[2], and Jordi Levy[3]

[1] Universitat de Lleida (DIEI, UdL)
[2] Universitat Politècnica de Catalunya (LSI, UPC)
[3] Artificial Intelligence Research Institute (IIIA, CSIC)

Abstract. Recently, Fu and Malik described an unweighted 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. Later Marques-Silva and Planes implemented and extended these ideas. In this paper we present and implement two Partial MaxSAT solvers and the weighted variant of one of them. Both are based on Fu and Malik ideas. We prove the correctness of our algorithm and compare our solver with other (Weighted) MaxSAT and (Weighted) Partial MaxSAT solvers.

1 Introduction

In real-life, some solutions to a problem are acceptable even when some constraints are violated. In fact, in many situations it is impossible to satisfy all constraints. For instance, in the context of planning, scheduling, packing, etc., a solution satisfying all the constraints may be impossible to obtain. However we are still interested on which is the maximum number of constraints that can be satisfied with a minimal penalty.

We can solve these problems through the use of MaxSAT formalisms, such as (Weighted) MaxSAT and (Weighted) Partial MaxSAT. Recently, there has been an increasing interest in the development of solvers for these formalisms. Since 2006, every year takes place the MaxSAT evaluation [2]. Most of the solvers submitted to the last MaxSAT08 evaluation are implementations of branch&bound algorithms (MaxSatz [11], IncWMaxSatz, W-MaxSatz, WMaxsatz_icss [6], MiniMaxSat [10], Lb-Sat and Lb-PSat [12,13], PMS [3], ToolBar3 [9]). There are other approaches like the solver Clone [17], that makes use of a tractable language known as d-DNNF, and those which are based on the use of Satisfiability testing, SAT4J [4], msu1.2 [14,15] and msu4.0 [16].

None of these solvers is a clear winner, specially for industrial and crafted instances. In particular, for the industrial category the solvers based on Satisfiability testing seem to perform very well for many benchmarks. Since the ultimate goal is to solve real world (industrial) instances it makes sense to study in detail

* Research partially supported by the projects TIN2007-68005-C04-{01,03,04} and TIN2006-15662-C02-02 funded by the MEC.

O. Kullmann (Eds.): SAT 2009, LNCS 5584, pp. 427–440, 2009.

this approach. Why these solvers work better for industrial instances may be a phenomena not only related to the hardness of the unsatisfiability cores included in the formulas (that can be efficiently detected by a SAT solver) but also to how these cores are connected.

The base of the study of this paper is the work of Fu and Malik [7,8], where two Partial MaxSAT algorithms based on calls to a SAT solver are proposed, and the work of Marques-Silva and Planes [15,16] and Marques-Silva and Manquinho [14] which extend that previous work.

The contributions of our work are (i) a more optimized implementation of the original Fu and Malik algorithm; (ii) a weighted version of the original Fu and Malik algorithm together which the proof of its correctness; and (iii) another Partial MaxSAT solver variant of the Fu and Malik algorithm, and the proof of its correctness.

For the purpose of the evaluation of these algorithms, there is only one solver, SAT4J [4], that is adapted to deal with weights. In this paper, we provide a weighted version of the Fu and Malik algorithm [8]. Our experimental investigation confirms what we already knew from previous MaxSAT evaluations. There is no unique best algorithm for solving MaxSAT or the other variants. Nevertheless, our implementation has a better performance than other solvers based on Satisfiability testing. In the case of the Partial Weighted MaxSAT, our solver is the first implementation of the original Fu and Malik ideas extended to the weighted problem. Therefore, we can only compare with SAT4J.

2 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 $MaxSAT(F)$.

In Weighted Partial MaxSAT, we use multisets of *weighted clauses*. A weighted clause is a pair (C, w), where C is a clause and w is a natural number meaning the penalty for falsifying the clause C. The pair (C, w) is clearly equivalent to having w copies of clause C in our multiset (in case C is soft). If a clause is hard, the corresponding weight is infinity.

Given a truth assignment I and a multiset of weighted clauses $\mathcal{C}$, the *cost* of assignment I on $\mathcal{C}$ is the sum of the weights of the clauses falsified by I.

The *Weighted Partial MaxSAT problem* for a multiset of weighted clauses $\mathcal{C}$ is the problem of finding an *optimal assignment* to the variables of $\mathcal{C}$ that minimizes the cost of the assignment on $\mathcal{C}$. If the cost is infinity, it means that we have falsified a hard clause, and we say that the multiset is *unsatisfiable*.

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*.

3 A Weighted Partial MaxSAT Algorithm

Before giving the full version of our algorithm, we will present the original Fu and Malik [8] algorithm for Partial MaxSAT, and show the correction of the algorithm. The reason for doing this is that we will need parts of the argument to show the correctness of our algorithm for solving Weighted Partial MaxSAT.

The algorithm consists in iteratively calling a SAT solver on a working formula φ. This corresponds to the line $(st, \varphi_c) := SAT(\varphi_w)$. 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 clause. The new working formula φ 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 ($CNF(\sum_{b \in BV} b = 1)$ in the code), and adding one to the counter of falsified clauses. This procedure is applied until the SAT solver returns satisfiable.

```
input: φ = {C_1, ..., C_m}
cost := 0                                   Optimal
while true do
    (st, φ_c) := SAT(φ)                     Call to the SAT solver
    if st = SAT then return cost
    BV := ∅                                 Set of blocking variables
    for each C ∈ φ_c do
        if C is soft then
            b := new blocking variable
            φ := φ \ {C} ∪ {C ∨ b}          Add blocking variable
            BV := BV ∪ {b}
    if BV = ∅ then return UNSAT             There are no soft clauses in the core
    φ := φ ∪ CNF(Σ_{b∈BV} b = 1)            Add cardinality constraint as hard clauses
    cost := cost + 1
```

Fig. 1. The pseudo-code of the Fu&Malik algorithm

The following lemma and definition are part of the correctness of the algorithm for both the weighted and unweighted versions.

Definition 1. *We say that two (Weighted) (Partial) MaxSAT formulas φ and φ' are MaxSAT equivalent if the cost of the optimal assignment of φ is equal to the cost of the optimal assignment of φ'.*

Lemma 1. *Let φ be an unsatisfiable CNF formula, and $\varphi_c = \{C_1, \ldots, C_s\}$ be an unsatisfiable core in φ. Define*

$$\varphi' = (\varphi \setminus \varphi_c) \ \cup \ \{C_1 \vee b_1, \ldots, C_s \vee b_s\} \cup \ CNF(\sum_{i=1}^{s} b_i = 1) \ \cup \ \{\square\}$$

where $b_1, \ldots, b_s$ are new variables.

Then, the minimum number of falsified clauses of φ is the same as the minimum number of falsified clauses of φ', i.e. φ and φ' are MaxSAT equivalent.

PROOF: Let I be a truth assignment for the variables of φ that satisfies all the hard clauses of φ. Since φ_c is an unsatisfiable core, I falsifies some clause in φ_c. Let C_i be one such clause. Now define I' the following way: for all $x \in \varphi$, $I'(x) = I(x)$; $I'(b_i) = 1$ and $I'(b_j) = 0$ for all $j \neq i$, $1 \leq j \leq s$. Now I' satisfies $CNF(\sum_{i=1}^{s} b_i = 1)$. For every clause C in $\varphi \setminus \varphi_c$, $I'(C) = I(C)$, and the same is true for all the clauses $C_j \vee b_j$ for $j \neq i$. Now, I falsifies C_i but I' satisfies $C_i \vee b_i$ and falsifies $\square$. As a consequence the number of falsified clauses of φ' by I' is the same as the number of falsified clauses of φ by I.

Now consider an optimal assignment I' for φ'. By the optimality of I', we know that I' satisfies $CNF(\sum_{i=1}^{s} b_i = 1)$ and if $I'(C_i) = 1$ then $I'(b_i) = 0$. Now we define an assignment I for φ the following way: $I(x) = I'(x)$ for all $x \in \varphi$. We will see that the number of falsified clauses of I is the same as the number of falsified clauses in I'. The number of falsified clauses in $\varphi \setminus \varphi_c$ is clearly the same. Let b_i be the variable assigned true by I'. Then $I'(C_j \vee b_j) = I(C_j)$ for $j \neq i$. On the other hand, $I(C_i) = 0$ and $I'(C_i \vee b_i) = 1$ but I' falsifies $\square$. ■

Theorem 1. FU&MALIK *is a correct algorithm for Partial MaxSAT.*

PROOF: In each iteration of the while loop, if the SAT solver returns unsatisfiable and the unsatisfiable core has soft clauses, we substitute a formula φ by another φ' plus the addition of one to the variable *cost*. Adding 1 to *cost* is equivalent to considering that φ' has also the empty clause. Lemma 1 shows that both formulas are equivalent in terms of the minimum number of unsatisfiable clauses. ■

The following algorithm is the weighted version of the previous one. Now we iteratively call the SAT solver with the working formula without 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. Now 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*.

Lemma 2. *Let φ be a weighted partial formula. Let $exp(\varphi)$ be the natural expansion of φ into an unweighted formula by substituting every clause (C, w) of φ into w copies of C.*

The minimum weight of φ is the same as the minimum number of falsified clauses of $exp(\varphi)$.

PROOF: This is straightforward. ■

The next lemma shows that if we have several identical unsatisfiable cores, we don't need to add different blocking variables to each core. Instead all cores can have the same set of blocking variables.

input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m)\}$
$cost := 0$ — 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 $st = SAT$ **then return** *cost*
  $BV := \emptyset$ — Blocking variables of the core
  $w_{min} := \min\{w_i \mid C_i \in \varphi_c \text{ and } C_i \text{ is soft}\}$
  for each $C_i \in \varphi_c$ **do**
    if C_i is soft **then**
      $b_i :=$ new blocking variable
      $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min})\} \cup \{(C_i \vee b_i, w_{min})\}$
      Duplicate soft clauses of the core
      $BV := BV \cup \{b_i\}$
  if $BV = \emptyset$ **then return** UNSAT — There are no soft clauses in the core
  else $\varphi := \varphi \cup CNF(\sum_{b \in BV} b = 1)$ — Add cardinality constraint as hard clauses
  $cost := cost + w_{min}$

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

Lemma 3. *Let φ be an unsatisfiable partial formula and let $\varphi_c = \{C_1, \ldots, C_s\}$ be an unsatisfiable core in φ that appears l times. Consider the following formulas:*

$$\varphi_1 = \varphi \setminus \varphi_c \cup \{\underbrace{C_i \vee b_i, \ldots, C_i \vee b_i}_{l \text{ times}} \mid C_i \in \varphi_c\} \cup CNF(\sum_{i=1}^{s} b_i = 1)$$

and

$$\begin{aligned}\varphi_2 = &\varphi \setminus \varphi_c \cup \{C_i \vee b_i^1, \ldots, C_i \vee b_i^l \mid C_i \in \varphi_c\} \\ &\cup CNF(\textstyle\sum_{i=1}^{s} b_i^1 = 1) \cup \ldots \cup CNF(\sum_{i=1}^{s} b_i^l = 1)\end{aligned}$$

Then, the minimum number of unsatisfiable clauses of φ_1 and φ_2 are the same, i.e. φ_1 and φ_2 are MaxSAT equivalent.

Proof: Let I be an optimal interpretation for φ_1. Then, if $I(C_i) = 1$, for some $i = 1, \ldots, s$, then $I(b_i) = 0$. This is true because φ_c is unsatisfiable and I is optimal. Now we will modify I into an assignment I' the following way:

$$\begin{aligned}I'(x) &= I(x) \text{ for all } x \in \varphi \\ I'(b_i^j) &= I(b_i) \text{ for } i = 1, \ldots, s \text{ and } j = 1, \ldots, l\end{aligned}$$

It is clear that for all $C \in \varphi - \varphi_c$, $I'(C) = I(C)$. Also, $I'(C_i \vee b_i^j) = I(C_i \vee b_i)$.

Let now I' be an optimal assignment for φ_2. Then as before, if $I(C_i) = 1$, for some $i = 1, \ldots, s$, then $I(b_i^j) = 0$ for every $j = 1, \ldots, l$. Now we will modify I' into an assignment I the following way:

$$I(x) = I'(x) \text{ for all } x \in \varphi$$
$$I(b_i) = I(b_i^1) \text{ for all } i = 1, \ldots, s$$

It is clear that for all $C \in \varphi - \varphi_c$, $I'(C) = I(C)$. We will show that the number of unsatisfied clauses of $\{C_1 \vee b_1^j, \ldots, C_s \vee b_s^j\}$ by I' for every $j = 1, \ldots, l$ is the same as the number of unsatisfied clauses of $\{C_i \vee b_1, \ldots, C_s \vee b_s\}$ by I. Now suppose that the only b variable that I assigns true is b_i^j and the only b variable that I assigns true is b_k. By assumption, $I(C_i) = I(C_k) = 0$. Then $I'(C_i \vee b_i^j) = 1$ and $I'(C_k \vee b_k^j) = 0$, but $I(C_i \vee b_i) = 0$ and $I(C_k \vee b_k) = 1$. ■

The next lemma shows the correctness of one iteration of our Weighted Partial MaxSAT algorithm WPM1.

Lemma 4. *Let φ be an unsatisfiable weighted partial formula, let $\varphi_c = \{C_1, \ldots, C_s\}$ be an unsatisfiable core in the set of clauses from φ, and let $\varphi_c^w = \{(C_1, w_1), \ldots, (C_s, w_s)\}$ the subset of weighted clauses of φ that corresponds to the core. Let $w_{min} = \min(w_1, \ldots, w_s)$, and let*

$$\begin{aligned}\varphi' =& (\varphi \setminus \varphi_c^w) \cup \{(C_i, w_i - w_{min}) \mid C_i \in \varphi_c\} \\ & \cup \{(C_i \vee b_i, w_{min}) \mid C_i \in \varphi_c\} \\ & \cup \mathit{CNF}(\textstyle\sum_{i=1}^{s} b_i = 1) \cup \{(\square, w_{min})\}\end{aligned}$$

where $\{b_1, \ldots, b_s\}$ is a set of new variables.

Then, φ and φ' are MaxSAT equivalent.

Proof: Let $exp(\varphi)$ be the unweighted expansion of φ. Lemma 2 shows that the minimum weight of φ is the same as the number of falsified clauses of $exp(\varphi)$. Now $\varphi_c = \{C_1, \ldots, C_s\}$ is an unsatisfiable core of $exp(\varphi)$, and since $w_{min} = \min(w_1, \ldots, w_s)$, φ_c appears w_{min} times in $exp(\varphi)$. Now we can apply the transformation of lemma 1 w_{min} times to obtain a formula

$$\begin{aligned}\varphi_2 =& \varphi \setminus \varphi_c \cup \{C_i \vee b_i^1 \ldots C_i \vee b_i^{w_{min}} \mid C_i \in \varphi_c\} \\ & \cup \mathit{CNF}(\textstyle\sum_{i=1}^{s} b_i^1 = 1) \cup \ldots \cup \mathit{CNF}(\sum_{i=1}^{s} b_i^l = 1) \\ & \{ \underbrace{C_i, \ldots, C_i}_{w_i - w_{min} \text{ copies}} \mid C_i \in \varphi_c\} \cup \{\underbrace{\square, \ldots, \square}_{w_{min}}\}\end{aligned}$$

MaxSAT equivalent to φ_{exp}. By Lemma 3, φ' is MaxSAT equivalent to the formula

$$\begin{aligned}\varphi_1 =& \varphi \setminus \varphi_c \cup \{\underbrace{C_i \vee b_i, \ldots, C_i \vee b_i}_{w_{min} \text{ copies}} \mid C_i \in \varphi_c\} \\ & \cup \{ \underbrace{C_i, \ldots, C_i}_{w_i - w_{min} \text{ copies}} \mid C_i \in \varphi_c\} \\ & \cup \mathit{CNF}(\textstyle\sum_{i=1}^{s} b_i = 1) \cup \{\underbrace{\square, \ldots, \square}_{w_{min}}\}\end{aligned}$$

Now using again Lemma 2, φ_1 is MaxSAT equivalent to φ' as in the statement of the lemma. ■

Theorem 2. *WPM1 is a correct algorithm for Weighted Partial MaxSAT.*

PROOF: The theorem is proved iterating Lemma 4 for every execution of the loop of the algorithm. ■

4 Another Partial MaxSAT Algorithm

The next algorithm, that we call PM2, is also a variant of the Fu and Malik algorithm that avoids the use of more than one blocking variable in a clause. A single blocking variable is added to each soft clause, like in other solvers like SAT4J [4], msu3 [15] and msu4.0 [16].

PM2 works as follows: every clause gets an additional variable and the cardinality constraint says that all these additional variables have to be false. Also before the first iteration of the algorithm the counter of falsified clauses, *cost*, is set to zero. 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. 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 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 that need to be one is at least the number of cores included in the last core found (counting the last). In every call to the SAT solver we also add an *at most* cardinality constraint saying that the sum of all blocking variables is at most *cost*. 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 [4], msu3 [15] and msu4.0 [16]. 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 [15], 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 [16], 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, our *at most* constraint has a bound *cost* that is successively increased like in msu3, instead of decreased like in SAT4J. Second, our *at least* constraints may impose a bound strictly greater than one, in contrast with the msu4.0 algorithm. This would have to result in a more restrictive constraint, thus in fewer assignments to check by the SAT solver.

---

input: $\varphi = \{C_1, \ldots, C_m\}$
$BV := \{b_1, \ldots, b_m\}$ — Set of all blocking variables
$\varphi_w := \{C_1 \vee b_1, \ldots, C_m \vee b_m\}$ — Protect all clauses
$cost := 0$ — Optimal
$L := \emptyset$ — Set of Cores
while $true$ **do**
  $(st, \varphi_c) := SAT(\varphi_w \cup CNF(\sum_{b \in BV} b \leq cost))$ — Call to the SAT solver with at most cardinality constraint
  if $st = SAT$ **then return** $cost$
  remove the hard clauses from φ_c
  if $\varphi_c = \emptyset$ **then return** UNSAT
  $B := \emptyset$ — Blocking variables of the core
  for each $C = C_i \vee b_i \in \varphi_c$ **do**
    $B := B \cup \{b_i\}$
  $L := L \cup \{\varphi_c\}$
  $k := |\{\psi \in L \mid \psi \subseteq \varphi_c\}|$ — Num. of cores contained in φ_c including φ_c
  $\varphi_w := \varphi_w \cup CNF(\sum_{b \in B} b \geq k)$ — Add at least cardinality constraint
  $cost := cost + 1$

Fig. 3. The pseudo-code of the PM2 algorithm

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, 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, and also stops returning the same MaxSAT value, since both have run the same number of steps.

Since Theorem 3 will be proved by induction, assume by induction hypothesis that

$$\varphi = \{C_1 \vee a_1, \ldots, C_m \vee a_m\} \cup \mathit{CNF}(\sum_{i=1}^{m} a_i \leq s) \cup \bigcup_{r=1}^{s} \mathit{CNF}(\sum_{i \in B_r} a_i \geq k_r)$$

is the formula computed by PM2 after s execution steps, where $B_1, \ldots, B_s$ is the sequence of cores, and k_j is the number of cores from $B_1, \ldots, B_j$ included in B_j. Assume also that FU&MALIK, after s steps simulating PM2, with the same sequence of cores $B_1, \ldots, B_s$, obtains the formula

$$\hat{\varphi} = \{C_1 \vee \bigvee_{1 \in B_j} b_1^j, \ldots, C_n \vee \bigvee_{n \in B_j} b_n^j\} \cup \bigcup_{j=1}^{s} \mathit{CNF}(\sum_{i \in B_j} b_i^j = 1)$$

Lemma 5. *Let φ 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 φ given by*

$$\begin{array}{ll} I(x) = \hat{I}(x) & \textit{for any variable } x \in \{C_1, \ldots, C_n\} \\ I(a_i) = \max\{\hat{I}(b_i^j) \mid i \in B_j\} & \textit{for the blocking variables} \end{array}$$

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 φ; and
(3) if $\hat{I}$ satisfies the soft clause $C_i \vee \bigvee_{i \in B_j} b_i^j \in \hat{\varphi}$, 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), if $\hat{I}$ satisfies the cardinality constraints of $\hat{\varphi}$, then $\sum_{i \in B_r} \hat{I}(b_i^r) = 1$, for any core B_r. Hence,

$$\sum_{i \in B_r} \sum_{\substack{i \in B_j \\ j=1,\ldots s}} \hat{I}(b_i^j) \geq \sum_{\substack{B_j \subset B_r \\ j \leq r}} \sum_{i \in B_j} \hat{I}(b_i^j) = |\{B_j \mid B_j \subseteq B_r \wedge j \leq r\}|$$

for any of the cores B_r obtained in the execution. If the interpretation $\hat{I}$ is optimal, it means that it assigns *true* to at most one of the blocking variables of a clause. In other words,

$$\sum_{\substack{i \in B_j \\ j=1,\ldots,s}} \hat{I}(b_i^j) \leq 1 \quad \text{hence} \quad I(a_i) = \max\{\hat{I}(b_i^j) \mid i \in B_j \wedge j = 1, \ldots, s\} = \sum_{\substack{i \in B_j \\ j=1,\ldots,s}} \hat{I}(b_i^j)$$

for any $i = 1, \ldots, n$. From all this, we conclude $\sum_{i \in B_r} I(a_i) \geq |\{B_j \mid B_j \subseteq B_r \wedge j \leq r\}| = k_r$, i.e. I satisfies the cardinality constraints $\mathit{CNF}(\sum_{i \in B_r} a_i \geq k_r)$.

Similarly, we can prove that if $\hat{I}$ is optimal

$$\sum_{i=1}^{n} I(a_i) = \sum_{i=1}^{n} \max\{\hat{I}(b_i^j) \mid i \in B_j \wedge j = 1, \ldots, s\}$$
$$= \sum_{i=1}^{n} \sum_{\substack{i \in B_j \\ j=1,\ldots,s}} \hat{I}(b_i^j) = \sum_{j=1}^{s} \sum_{i \in B_j} \hat{I}(b_i^j) = \sum_{j=1}^{s} 1 = s$$

Hence, the other cardinality constraint $CNF(\sum_{i=1}^{m} a_i \leq s)$ is also satisfied.

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

Lemma 6. *Let φ 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 φ, then B is also a core of $\hat{\varphi}$.*

Proof: If B is not a core of $\hat{\varphi}$, then there exists an interpretation I of $\hat{\varphi}$ that satisfies all hard clauses, all cardinality constraints of $\hat{\varphi}$, and all soft clauses $C_i \vee \bigvee_{i \in B_j} b_i^j$ where $i \in B$. By Lemma 5, $\check{I}$ will satisfy all hard clauses, all cardinality constraints of φ, and all soft clauses $C_i \vee a_i$ where $i \in B$. This would contradict that B is a core of φ. ■

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 7. *Let φ and $\hat{\varphi}$ be the formulas obtained by* PM2 *and* Fu&Malik *algorithms, respectively, after s steps of simulation. If φ is satisfiable, then $\hat{\varphi}$ is satisfiable.*

Proof: Let I be an interpretation satisfying φ. In particular, I satisfies $CNF(\sum_{i=1}^{m} a_i \leq s)$, and all hard and soft clauses. Therefore, I satisfies all the *original* soft clauses C_i where $I(a_i) = false$, and there are at least $n - s$ of such clauses. We have $MaxSAT(C_1, \ldots, C_n) \leq s$. 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. ■

Theorem 3. PM2 *is a correct algorithm for Partial MaxSAT.*

Table 1. Time in seconds (solved). Timeout of 1200 seconds. # stands for number of instances of the benchmark.

set	#	best08	WPM1	PM2	msu1.2	msu4.0	SAT4J
Unweighted MaxSAT Category							
Crafted							
Maxcut/dimacs_mod/	62	IncMaxSatz - **81.8(52)**	0.03(4)	175(7)	0.28(4)	1.71(3)	0.93(2)
Maxcut/random/	58	MaxSatz - **4.5(40)**	-(0)	-(0)	- (0)	- (0)	- (0)
Maxcut/Spinglass/	5	MiniMaxSatz - **1.62(3)**	0.85(2)	102.5(2)	0.68 (2)	-(0)	-(0)
Industrial							
SeanSafarpour	112	msu1.2 - 57.5(72)	**66.6(81)**	90.2(75)	57.5(72)	64.4(50)	14.5(10)
Partial MaxSAT Category							
Crafted							
Maxclique/Random/	96	MiniMaxSAT - **2.4(96)**	50.4(1)	-(0)	-(0)	106(61)	114(52)
Maxclique/Structured/	62	MiniMaxSAT - **73(36)**	41.2(11)	32.6(6)	4.9(7)	105.2(13)	50.5(13)
Maxone/3SAT/	80	IncMaxSatz - **0.46(80)**	15.82(46)	105.7(79)	52.7(40)	118.2(35)	96.6(31)
Maxone/Structured/	60	SAT4J - **10.1(60)**	0.69(2)	547.5(13)	122.7(2)	3.34(1)	**10.1(60)**
Industrial							
Bcp-fir/	59	msu1.2 - 49.2(46)	**31.7 (57)**	67.4(56)	49.2(46)	-(0)	13.3(10)
Bcp-hipp-yRa1/	1183	SAT4J - 19.2(1111)	2.9(1122)	**13.5(1162)**	7.2(1105)	0.29(348)	12.20(1109)
Bcp-msp/	148	MiniMaxSAT - **48.9(104)**	15.5(26)	384.2(36)	4.9(25)	22.9(79)	8.8(93)
Bcp-mtg/	215	MiniMaxSAT - 25.7(206)	5.8(170)	**10.5(214)**	17.5(164)	0.43(22)	57(196)
Bcp-syn/	74	lb-psat - **63.4(34)**	14.1(32)	**71.2(34)**	51.1(31)	105.2(11)	67.4(21)
Pbo-mqc-nencdr/	128	msu4.0 - **167.5(115)**	80.4(50)	142(78)	50.3(54)	**167.5(115)**	180.6(102)
Pbo-mqc-nlogencdr/	128	msu4.0 - **111(128)**	67.1(75)	140.3(97)	53(65)	**111(128)**	117.5(126)
Pbo-routing/	15	msu1.2 - 2.9(15)	**0.94(15)**	24.7(15)	2.9(15)	54.9(15)	26.4(9)

5 Experimental Results

In order to conduct our experimental investigation we have selected the benchmarks submitted to the MaxSAT08 evaluation [2]. We have focus on the crafted and industrial instances for all the four categories: unweighted MaxSAT, partial MaxSAT, weighted MaxSAT and weighted partial MaxSAT. 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, in particular for the weighted and partial weighted categories, we decided also to incorporate the crafted instances.

Our experiments have been run on the same machine specs as the MaxSAT evaluation; Operating System: Rocks Cluster 4.0.0 Linux 2.6.9, Processor: AMD Opteron 248 Processor, 2 GHz and compilers, Memory: 1 GB and Compilers GCC 3.4.3, javac JDK 1.5.0. The solvers we compare are the best solvers for each category and benchmark at the MaxSAT08 evaluation [2], the solvers based on satisfiability testing (msu1.2, msu4.0 [16], and SAT4J [4]) and our implementations of the weighted version of Fu&Malik (WPM1) and Partial MaxSAT 2 (PM2) presented in this paper.

Our solvers are implemented on top of the SAT solver picosat846 [5], although they can be easily adapted to work with any other solver that provides an

Table 2. Time in seconds (solved). Timeout of 1200 seconds. # stands for number of instances of the benchmark.

set	#	best08	WPM1	SAT4J
Weighted MaxSAT Category				
Crafted				
KeXu/	15	IncWMaxsatz - **126.5(15)**	478(1)	7.7(4)
Ramsey/	48	lb-psat - **1.63(37)**	0.05(34)	16(35)
WMaxcut/dimacs_mod/	62	ToolBar3 - **59(56)**	0.12(3)	0.84(2)
WMaxcut/Random/	40	MiniMaxSAT - **5.43(40)**	-(0)	-(0)
WMaxcut/Spinglass/	5	MiniMaxSAT - **27.6(4)**	-(0)	-(0)
Weighted Partial MaxSAT Category				
Crafted				
Auctions/Auc_paths/	88	IncWMaxsatz - **8.4(88)**	-(0)	497(15)
Auctions/Auc_regions/	88	MiniMaxSAT - **1.7(84)**	-(0)	166(76)
Auctions/Auc_Sched/	84	MiniMaxSAT - **46(84)**	-(0)	317(49)
Random-net/	350	Clone - **72(236)**	194(91)	331(13)
Pseudo-factor/	186	IncWMaxsatz - **0.07(186)**	16(124)	3.3(186)
Pseudo- miplib/	16	SAT4J - 13(6)	0.29(3)	13(6)
QCP/	25	SAT4J - 6.14(25)	**0.27(25)**	6.14(25)
WCSP/Planning/	71	SAT4J - **6.55(71)**	0.9(46)	**6.55(71)**
WCSP/Spot5/Dir/	21	Clone - 87.6(6)	2.31(4)	76(3)
WCSP/Spot5/Log/	21	Clone - **15(6)**	0.52(5)	63.8(3)
Industrial				
Protein_ins	12	MiniMaxSAT - **482(8)**	42(1)	6.05(1)

interface to access to the unsatisfiable core when the formula is UNSAT. In order to encode the cardinality constraints, for WPM1, we use the regular encoding presented in [1], and for PM2 we use the encoding based on sequential counters presented in [18].

In SAT4J [4], for each clause c_i in the original problem, a new blocking variable b_i is added. Then a SAT solver is called to solve the new formula, and each time a model is found, a cardinality constraint is added to the formula that states that the sum of blocking variables has to be less than the number of blocking variables satisfied in the previous iteration. Once the SAT solver gives an UNSAT answer, the latest model is an optimal solution.

Msu1.2 [14,15] is another implementation of the Fu&Malik algorithm. Msu4.0 [16] is a more sophisticated approach, which alternates iterations to discover new cores with iterations to reduce the number of blocking variables that need to be set to true in each core.

Table 1 and Table 2 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 MaxSAT08 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.

We are interested in answering two questions: how our solvers would have performed at the MaxSAT08 evaluation [2], and how they compare to the current solvers based on satisfiability testing.

As we can see for the unweighted category, the solvers based on satisfiability testing perform well at the industrial category being our solver WPM1 the best performing one. For the crafted instances, the solvers based on satisfiability testing are not competitive, however our solver PM2 is the best among them.

For the partial category and the crafted instances, the solvers based on satisfiability testing are again not competitive, except for SAT4J [4] at one benchmark. However, for the industrial instances, they win 7 out of 9 benchmarks, in particular, WPM1 wins at 2, PM2 at 3 and msu4.0 [16] at 2.

For the weighted category, WPM1 and SAT4J [4] just show a good behavior on one set of instances, the Ramsey set. However, looking more closely to the set, many of the instances are actually satisfiable. Unfortunately, there are not available industrial instances for this category.

For the weighted partial category and crafted instances, WPM1 and SAT4J [4] are just able to win at 2 out 9 benchmarks. Again, unfortunately, there is only one set of industrial instances, and MinimaxSAT is the only solver able to solve 8 instances while the rest of the solvers submitted to the MaxSAT08 evaluation [2] and the one presented in this paper are not able to solve more than 2 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. Respect to the solvers we have presented in this paper, we have shown that our implementations show a good performance on the industrial instances for the unweighted and partial categories. That should be a promising base point for the weighted versions. Although we can not make such a claim yet, since there are not enough industrial instances at these categories in order to test our solvers, we think this research avenue is worth further investigation.

Acknowledgements

We especially thank Armin Biere for the insightful discussions about his SAT solver picosat. We would also like to thank the organizers of the MaxSAT evaluations.

References

1. Ansótegui, C., Manyà, F.: Mapping problems with finite-domain variables to problems with boolean variables. In: Hoos, H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 1–15. Springer, Heidelberg (2005)
2. Argelich, J., Li, C.M., Manyà, F., Planes, J.: The first and second Max-SAT evaluations. Journal on Satisfiability 4, 251–278 (2008)

3. Argelich, J., Manyà, F.: Partial Max-SAT solvers with clause learning. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 28–40. Springer, Heidelberg (2007)
4. Berre, D.L.: Sat4jmaxsat, `http://www.sat4j.org`
5. Biere, A.: PicoSAT essentials. Journal on Satisfiability 4, 75–97 (2008)
6. Darras, S., Dequen, G., Devendeville, L., Li, C.M.: On inconsistent clause-subsets for Max-SAT solving. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 225–240. Springer, Heidelberg (2007)
7. Fu, Z.: Extending the Power of Boolean Satisfiability: Techniques and Applications. PhD thesis, Princeton University, Princeton (2007)
8. 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)
9. Heras, F., Larrosa, J.: New inference rules for efficient Max-SAT solving. In: Proc. the 21th National Conference on Artificial Intelligence (AAAI 2006) (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. Li, C.M., Manyà, F., Planes, J.: New inference rules for Max-SAT. J. Artif. Intell. Res. (JAIR) 30, 321–359 (2007)
12. Lin, H., Su, K.: Exploiting inference rules to compute lower bounds for Max-SAT solving. In: Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007), pp. 2334–2339 (2007)
13. Lin, H., Su, K., Li, C.M.: Within-problem learning for efficient lower bound computation in Max-SAT solving. In: Proc. the 23th National Conference on Artificial Intelligence (AAAI 2008), pp. 351–356 (2008)
14. Marques-Silva, J., Manquinho, V.M.: 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)
15. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. CoRR, abs/0712.1097 (2007)
16. Marques-Silva, J., Planes, J.: Algorithms for maximum satisfiability using unsatisfiable cores. In: Proc. of the Conf. on Design, Automation and Test in Europe (DATE 2008), pp. 408–413 (2008)
17. Pipatsrisawat, K., Darwiche, A.: Clone: Solving weighted Max-SAT in a reduced search space. In: Australian Conference on Artificial Intelligence, pp. 223–233 (2007)
18. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 827–831. Springer, Heidelberg (2005)