On Solving MaxSAT Through SAT[*]

Carlos Ansótegui
Universitat de Lleida
(DIEI, UdL)
carlos@diei.udl.cat
, María Luisa Bonet
Universitat Politècnica de Catalunya
(LSI, UPC)
bonet@lsi.upc.edu
and Jordi Levy
Artificial Intelligence Research Institute
(IIIA, CSIC)
levy@iiia.csic.es

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.

1 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: $C_i \vee b_i$, 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 must 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. We can also, as SAT4J [Ber], start 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.

Fu and Malik [Fu07, FM06] (see Figure 1), and its implementation msu1.2 [MSM08, MSP07], 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

[*]Research partially supported by the projects MULOG2 (TIN2007-68005) and ARINF (TIN2009-14704) funded by the CICyT.

```
input: $\varphi = \{C_1, \ldots, C_m\}$
$cost := 0$                                         Optimal
while $true$ do
    $(st, \varphi_c) := SAT(\varphi)$               Call to the SAT solver
    if $st = SAT$ then return $cost$
    $BV := \emptyset$                               Set of blocking variables
    for each $C \in \varphi_c$ do
        if $C$ is soft then
            $b :=$ new blocking variable
            $\varphi := \varphi \setminus \{C\} \cup \{C \vee b\}$     Add blocking variable
            $BV := BV \cup \{b\}$
    if $BV = \emptyset$ then return UNSAT           There are no soft clauses in the core
    $\varphi := \varphi \cup \mathrm{CNF}(\sum_{b \in BV} b = 1)$     Add cardinality constraint as hard clauses
    $cost := cost + 1$
```

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

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 [ABL09], we show how we can replace, in the Fu&Malik algorithm, the sequence of auxiliary variables of each clause with just one. In [MSP08], an algorithm is described 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 [ABL09] showing how to exploit better the information given by the unsatisfiable cores to add richer cardinality constraints on the auxiliary variables.

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 clauses 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. If the formula passed to the SAT solver is unsatisfiable, then it returns an unsatisfiable set of clauses. We call to this set an *unsatisfiable core*.

To prove the correctness of our algorithm we will prove that it can simulate the Fu&Malik algorithm. The Fu and Malik [FM06] algorithm for Partial MaxSAT consists in iteratively calling a SAT solver on a working formula φ. This corresponds to the line $(st, \varphi_c) := SAT(\varphi_w)$ 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 (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 ($\mathrm{CNF}(\sum_{b \in BV} b = 1)$ in the code), and adding one to the counter of falsified clauses. This procedure is applied until the SAT solver returns satisfiable.

3 A Partial MaxSAT Algorithm

The next algorithm, that we call PM2, is a variant of the algorithm described in [ABL09]. 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 $i = 1, \dots, m$ 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 $A \in L$ is a set of indexes, we say that the set of indexes B is a* cover *of L, if it is a minimal non-empty set such that, for every $A \in L$, if $A \cap B \neq \emptyset$, then $A \subseteq B$.*
Given a set of cores L, we denote the set of covers of L as coversof(L).

Lemma 1. *coversof(L) is a partition of the set of indexes, therefore covers do not intersect. Moreover, every core of L is included into just one cover of coversof(L).*

Notice that, if L is empty, then all covers are singletons.

```
input: $\varphi = \{C_1, \dots, C_m, C_{m+1}, \dots, C_{m+m'}\}$      $m$ soft clauses and $m'$ hard clauses
$BV := \{b_1, \dots, b_m\}$                                           Set of all blocking variables
$\varphi_w := \{C_1 \vee b_1, \dots, C_m \vee b_m, C_{m+1}, \dots, C_{m+m'}\}$  Protect all soft clauses
$cost := 0$                                                           Optimal
$L := \emptyset$                                                      Set of Cores
$AL := \emptyset$                                                     Set of at-least constraints
while $true$ do
    $AM := \emptyset$                                                 Set of at-most constraints
    for each $B \in$ coversof($L$) do
        $k := |\{A \in L \mid A \subseteq B\}|$                       Num. of cores contained in the cover $B$
        $AM := AM \cup \mathrm{CNF}(\sum_{i \in B} b_i \leq k)$       Add new at-most cardinality constraint
    $(st, \varphi_c) := SAT(\varphi_w \cup AL \cup AM)$               Call to the SAT solver, return core if unsat
    if $st = SAT$ then return $cost$
    $A = \{i \in \{1, \dots, m\} \mid C_i \vee b_i \in \varphi_c \wedge b_i \in BV\}$  Indexes of soft clauses of the core
    if $A = \emptyset$ then return UNSAT
    $L := L \cup \{A\}$
    $k := |\{A' \in L \mid A' \subseteq A\}|$                         Num. of cores contained in $A$ including $A$
    $AL := AL \cup \mathrm{CNF}(\sum_{i \in A} b_i \geq k)$           Add new at-least cardinality constraint
    $cost := cost + 1$
```

Figure 2: The pseudo-code of the PM2 algorithm.

PM2 works as follows (see Figure 2): every soft clause C_i is extended with a blocking variable b_i. 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 that is increased with a new core in each iteration. The list of *covers* is re-calculated in each iteration from the list of cores.[1] At every iteration of the algorithm, we start by calculating the set of *covers* that cover the set of cores L. We add, for every cover B, 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. Then, a SAT solver is called. If the solver says that the formula is satisfiable, the algorithm returns *cost* as the minimal number of falsified clauses. If the solver returns unsatisfiable, it also gives an unsatisfiable core φ_c. If the core only contains hard clauses, then the algorithm returns unsatisfiable. Otherwise, we put the indexes of the soft clauses of the core in a set A. 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 A is larger than or equal to one. If some cores are included, we add the cardinality constraint saying that the number of variables with indexes in A that need to be one is at least the number of cores included in the last core found (counting the last).

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 [Ber], msu3 [MSP07] and msu4.0 [MSP08]. 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 [MSP07], 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 [MSP08], 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 $A = \{i_1, \dots, i_m\}$ 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 A, then this set A of (soft) clauses is also a core for the Fu&Malik algorithm (see Lemma 3), in particular if the set $A = \emptyset$ is a core for PM2, and it returns UNSAT, then the same set $A = \emptyset$ 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 4), and also stops returning the same MaxSAT value, since both have run

[1]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.

the same number of steps.

Since Theorem 1 will be proved by induction, assume by induction hypothesis that

$$\begin{aligned}\varphi = \quad & \{C_1 \vee a_1, \dots, C_m \vee a_m, C_{m+1}, \dots, C_{m+m'}\} \\ \cup & \bigcup_{A \in \{A_1,\dots,A_s\}} \mathrm{CNF}(\sum_{i\in A} a_i \geq |\{A_j \mid j \leq r \wedge A_j \subseteq A\}|) \\ \cup & \bigcup_{B \in \mathrm{coversof}\{A_1,\dots,A_s\}} \mathrm{CNF}(\sum_{i\in B} a_i \leq |\{A_j \mid j \leq s \wedge A_j \subseteq B\}|)\end{aligned}$$

is the formula computed by PM2 after s execution steps, where $A_1, \dots, A_s$ is the sequence of cores. Assume also that FU&MALIK, after s steps simulating PM2, with the same sequence of cores $A_1, \dots, A_s$, obtains the formula

$$\hat{\varphi} = \{C_1 \vee \bigvee_{1\in A_j} b_1^j, \dots, C_m \vee \bigvee_{m\in A_j} b_m^j, C_{m+1}, \dots, C_{m+m}\} \cup \bigcup_{j=1}^{s} \mathrm{CNF}(\sum_{i\in A_j} b_i^j = 1)$$

Notice that, if $i \in A_j$, then b_i^j is the variable added by the FU&MALIK algorithm to clause i, at iteration j.

Lemma 2. *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*

$$\begin{array}{ll} I(x) = \hat{I}(x) & \textit{for any variable } x \in \{C_1, \dots, C_{m+m'}\} \\ I(a_i) = \max\{\hat{I}(b_i^j) \mid i \in A_j \wedge j = 1,\dots,s\} & \textit{for the blocking variables} \end{array}$$

Then, (1) if $\hat{I}$ satisfies the hard clause $C_{i+m} \in \hat{\varphi}$, for $i = 1, \dots, m'$, then I satisfies the hard clause $C_{i+m} \in \varphi$;
(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 $C_i \vee \bigvee_{i\in A_j} b_i^j \in \hat{\varphi}$, for $i = 1, \dots, m$, 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 *true* to at most one of the blocking variables of a clause. In other words, for any clause $i = 1, \dots, m$, we have $\sum\{\hat{I}(b_i^j) \mid j = 1, \dots, s \wedge i \in A_j\} \leq 1$. Therefore, the following maximal value and the sum coincides and we have

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

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

$$\sum_{i\in A_r} I(a_i) = \sum_{i\in A_r} \sum_{\substack{i\in A_j \\ j=1,\dots s}} \hat{I}(b_i^j) \geq \sum_{\substack{A_j\subset A_r \\ j\leq r}} \sum_{i\in A_j} \hat{I}(b_i^j) = \sum_{\substack{A_j\subset A_r \\ j\leq r}} 1 = |\{A_j \mid j \leq r \wedge A_j \subseteq A_r\}|$$

Hence, for any $r = 1, \dots, s$, I satisfies the at-least cardinality constraints

$$\mathrm{CNF}(\sum_{i \in A_r} a_i \geq |\{A_j \mid j \leq r \wedge A_j \subseteq A_r\}|)$$

Now, for any cover $B \in \mathrm{coversof}\{A_1, \dots, A_s\}$ we have

$$\begin{aligned} \sum_{i\in B} I(a_i) &= \sum_{i\in B} \sum_{\substack{i\in A_j \\ j=1,\dots,s}} \hat{I}(b_i^j) = \sum_{i\in B} \sum_{\substack{A_j\subseteq B \\ j=1,\dots,s}} \hat{I}(b_i^j) = \sum_{\substack{A_j\subseteq B \\ j=1,\dots,s}} \sum_{i\in A_j} \hat{I}(b_i^j) = \sum_{\substack{A_j\subseteq B \\ j=1,\dots,s}} 1 \\ &= |\{A_j \mid j = 1, \dots, s \wedge A_j \subset B\}| \end{aligned}$$

because, by definition of cover, $i \in B$ and $i \in A_j$ implies $A_j \subseteq B$.

Hence, for any cover $B \in \mathrm{coversof}\{A_1, \dots, A_s\}$, I satisfies the at-most cardinality constraints

$$\mathrm{CNF}(\sum_{i \in B} a_i \geq |\{A_j \mid j \leq s \wedge A_j \subseteq B\}|)$$

For (3), if $\hat{I}$ satisfies $C_i \vee \bigvee_{i\in A_j} b_i^j$, 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_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 3. *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 $\hat{I}$ of $\hat{\varphi}$ that satisfies all hard clauses, all cardinality constraints of $\hat{\varphi}$, and all soft clauses $C_i \vee \bigvee_{i\in A_j} b_i^j$ where $i \in B$. By Lemma 2, 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 4. *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 $\mathrm{CNF}(\sum_{i=1}^m a_i \leq s)$, 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 $m-s$ of such clauses. We have $MaxSAT(C_1, \dots, C_m) \leq s$. 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 1. PM2 *is a correct algorithm for Partial MaxSAT.*

set	#	Best in MaxSAT eval.'09	PM2
Unweighted MaxSAT Category			
Crafted			
Maxcut/dimacs_mod/	62	IncMaxSatz - 81.8(52)	186 (10)
Maxcut/Spinglass/	5	WMaxSatz1.6 - 4.88(3)	3.92 (2)
Bipartite/maxcut/-140-630-0.7/	50	IUT_BMB_Maxsatz - 213.06(50)	0 (0)
Bipartite/maxcut/-140-630-0.8/	50	IUT_BMB_Maxsatz - 156.19(50)	0 (0)
Industrial			
CircuitDebuggingProblems	9	MSUnCore - 17.50(8)	20.32(5)
SeanSafarpour	112	MSUnCore - 37.97(84)	63 (72)
Partial MaxSAT Category			
Crafted			
JobShop/	4	**PM2 - 151.78(3)**	151.78(3)
Maxclique/Random/	96	IUT_BMB_Maxsatz - 74(82)	210 (58)
Maxclique/Structured/	62	WMaxSatz-2.5 - 183.87(24)	525.58(17)
Maxone/3SAT/	80	**PM2 - 21.43(80)**	21.43(80)
Maxone/Structured/	60	SAT4J - 5.71(60)	430.7(41)
PSEUDO/miplib/	4	**PM2 - 22.26(4)**	22.26(4)
partial_crafted/frb/	25	**PM2 - 896.50(3)**	896.5(3)
min-enc/kbtree/	54	IUT_BMB_Maxsatz - 225.60(19)	121.95(17)
Industrial			
CircuitTraceCompaction/	4	SAT4J - 5.71(3)	442.99(3)
HaplotypeAssembly/	6	WBO - 9.72(5)	34.52(5)
PROTEIN_INS/	12	**PM2 - 56.83(2)**	56.83(2)
Bcp-fir/	59	**PM2 - 13.6(57)**	13.6(57)
Bcp-hipp-yRa1/SU/	38	**PM2 - 101.65(28)**	101.65(28)
Bcp-hipp-yRa1/Simp/	138	**PM2 - 8.10(136)**	8.10(136)
Bcp-msp/	148	SAT4J - 7.54(95)	91.31(94)
Bcp-mtg/	215	**PM2 - 1.36(215)**	1.36(215)
Bcp-syn/	74	**PM2 - 12.64(38)**	12.64(38)
Pbo-mqc-nencdr/	128	SAT4J - 318.54(114)	233.02(91)
Pbo-mqc-nlogencdr/	128	SAT4J - 176(125)	148(108)
Pbo-routing/	15	MSUnCore - 0.63(15)	1.18(15)

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.

4 Experimental Results

Our solver is implemented on top of the SAT solver picosat846 [Bie08], although it 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 [Sin05].

We include in Table 1 the results of the MaxSAT'09 evaluation, where our solver won the category of industrial partial MaxSAT, and performs well in the crafted partial MaxSAT category. 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 evaluations [ALMP08] 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 families where PM2 wins.

References

[ABL09] Carlos Ansotegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'09)*, 2009.

[ALMP08] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. The first and second Max-SAT evaluations. *Journal on Satisfiability*, 4:251–278, 2008.

[Ber] Daniel Le Berre. Sat4jmaxsat. In *www.sat4j.org*.

[Bie08] Armin Biere. PicoSAT essentials. *Journal on Satisfiability*, 4:75–97, 2008.

[FM06] Zhaohui Fu and Sharad Malik. On solving the partial max-sat problem. In *Proc. of the 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 252–265, 2006.

[Fu07] Zhaohui Fu. *Extending the Power of Boolean Satisfiability: Techniques and Applications.* PhD thesis, Princeton University, 2007.

[MSM08] João Marques-Silva and Vasco M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *Proc. of the 11th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'08)*, pages 225–230, 2008.

[MSP07] João Marques-Silva and Jordi Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, abs/0712.1097, 2007.

[MSP08] João Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Proc. of the Conf. on Design, Automation and Test in Europe (DATE'08)*, pages 408–413, 2008.

[Sin05] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proc. of the 11th Int. Conf. on Principles and Practice of Constraint Programming (CP'05)*, pages 827–831, 2005.