Artificial Intelligence 196 (2013) 77–105

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

SAT-based MaxSAT algorithms ☆

Carlos Ansótegui [a], Maria Luisa Bonet [b], Jordi Levy [c,*]

[a] Universitat de Lleida (DIEI), Jaume II 69, Lleida, Spain
[b] Universitat Politècnica de Catalunya (LSI), J. Girona 1-3, Barcelona, Spain
[c] Artificial Intelligence Research Institute (IIIA, CSIC), Campus UAB, Bellaterra, Spain

ARTICLE INFO

Article history:
Received 19 June 2012
Received in revised form 3 January 2013
Accepted 13 January 2013
Available online 21 January 2013

Keywords:
MaxSAT
SAT
Boolean optimization

ABSTRACT

Many industrial optimization problems can be translated to MaxSAT. Although the general problem is NP hard, like SAT, many practical problems may be solved using modern MaxSAT solvers. In this paper we present several algorithms specially designed to deal with industrial or real problems. All of them are based on the idea of solving MaxSAT through successive calls to a SAT solver. We show that this SAT-based technique is efficient in solving industrial problems. In fact, all state-of-the-art MaxSAT solvers that perform well in industrial instances are based on this technique. In particular, our solvers won the 2009 partial MaxSAT and the 2011 weighted partial MaxSAT industrial categories of the MaxSAT evaluation. We prove the correctness of all our algorithms. We also present a complete experimental study comparing the performance of our algorithms with latest MaxSAT solvers.

1. Introduction

The MaxSAT problem is a generalization of the satisfiability problem. The idea is that sometimes not all restrictions of a problem can be satisfied, and we try to satisfy the maximum number of them. The MaxSAT problem can be further generalized to the Weighted Partial MaxSAT problem. In this case, we can divide the restrictions in two groups: the clauses that must be satisfied (*hard*), and the ones that may or may not be satisfied (*soft*). In the last group, we may put different weights to the clauses, where the weight is the penalty to falsify the clause. The idea is that not all restrictions are equally important. The addition of weights to clauses makes the instance *weighted*, and the separation into hard and soft clauses makes the instance *partial*. Given a weighted partial MaxSAT instance, we want to find the assignment that satisfies the hard clauses, and the sum of the weights of the falsified clauses is minimal. Such an assignment will be optimal in this context.

The weighted partial MaxSAT problem is a natural combinatorial problem, and it can be used in several domains, such as: scheduling [40] and timetabling [9] problems, FPGA routing [41], software package installation [7], design debugging [36], bioinformatics [39], probabilistic reasoning [34], etc. However, state-of-the-art solvers have not yet experienced the same success as SAT solvers for the Satisfiability problem in the industrial field. Weighted Partial MaxSAT is an NP-hard problem, so our aim is to produce efficient solvers to handle real/industrial problems that although generally big in size, are not as hard as the worst case instances.

☆ Research partially supported by the Ministerio de Economía y Competividad research projects ARINF TIN2009-14704-C03-01, TASSAT TIN2010-20967-C04-01/03/04.

* Corresponding author.
E-mail addresses: carlos@diei.udl.cat (C. Ansótegui), bonet@lsi.upc.edu (M.L. Bonet), levy@iiia.csic.es (J. Levy).
URLs: http://www.lsi.upc.edu/~bonet (M.L. Bonet), http://www.iiia.csic.es/~levy (J. Levy).

0004-3702/$ – see front matter
http://dx.doi.org/10.1016/j.artint.2013.01.002

Originally, MaxSAT solvers (*wmaxsatz* [23], *minimaxsat* [18], *incwmaxsatz* [24]) were based on a depth-first branch and bound solving schema. Recently, the use of SAT solvers for solving MaxSAT problems has emerged as a new paradigm. The core idea is to reduce an optimization problem to a sequence of decision problems which are solved with the appropriated solver. This general approach comes from the observation that many important optimization problems fall into the class of FP^{NP}, which is the class of functions that can be computed polynomially with a SAT oracle [33]. This idea has been applied in [15] to solve scheduling optimization problems through a sequence of Constraint Satisfaction Problems and in [20] to solve planning problems through a sequence of SAT instances.

Fu and Malik [17,16] describe a MaxSAT algorithm in this direction. This algorithm can be considered as our initial source of inspiration. Since then, there has been a development of solvers based on satisfiability testing: *sat4j* [22], *wbo* and *msuncore* [25], *wpm1* [4], *wpm2* [5], *pwbo* [32], *bincd* [19] and *maxhs* [13]. Analyzing the results of the latest MaxSAT evaluations [8] we can conclude that, in general, the branch and bound solvers are more competitive on random problems, while solvers based on calls to a SAT solver are better for industrial or real problems.

The purpose of this paper is to summarize all our contributions to the development of SAT-based MaxSAT solvers [2–5], as well as to present some new results. They include detailed proofs of the correctness of all the algorithms, the introduction of the notion of MaxSAT reducibility and some of its properties, and a complete experimental comparison of our solvers with the present best performing solvers.

We present our algorithm WPM1 [4], which is the weighted version of the Fu and Malik algorithm [17,16], designed originally for Partial MaxSAT. Recently, in [2] we show how we can improve its performance by breaking the symmetries introduced by the reformulation of the MaxSAT problem into a sequence of SAT problems. We also show how we can force the SAT solver to prioritize the discovery of higher quality unsatisfiable cores to boost the overall search process.

We also present our algorithm PM2 [4,3] for Partial MaxSAT, which essentially follows the same search strategy as the Fu and Malik algorithm, but using just one blocking variable per clause. Finally, we present our WPM2 [5] algorithm which is almost a generalization of the PM2 algorithm for Weighted MaxSAT. The WPM2 algorithm was the first algorithm of its class. Although we provide an implementation of WPM2, we mostly focus on the description of a general architecture that can be parametrized to obtain more competitive solvers.

Our solvers based on PM2 and WPM1 solvers won the 2009 Partial MaxSAT category and 2011 Weighted Partial MaxSAT industrial categories of the MaxSAT evaluation, respectively. In this paper we present a complete experimental study with the best performing solvers of the MaxSAT 2011 evaluation and other solvers which have been recently published and did not take part.

This paper proceeds as follows. Section 2 presents some preliminary concepts. Section 3 briefly introduces the main basic ideas behind SAT-based MaxSAT solvers. Section 4 describes the concepts of cost-preservation, MaxSAT equivalence and MaxSAT reducibility. These notions play the same roles for MaxSAT formulas as equisatisfiability and logical equivalence for SAT formulas. Section 5 presents the Fu and Malik's algorithm [17,16] and proves its correctness.

Section 6 describes the problem of symmetries and shows how to break them. Section 7 presents the WPM1 algorithm, proves its correctness and describes the problem of the quality of the cores. Section 8 introduces a stratified approach to come up with higher quality cores in WPM1. This stratified approach is generalized, for any MaxSAT solver preserving MaxSAT reducibility, in Section 9. Sections 10 and 11 present the PM2 and WPM2 algorithms, respectively, and prove their correctness. Finally, Section 12 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 $\neg x_i$. 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 the 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. Given a weighted partial MaxSAT formula φ, we define the SAT formulas $\varphi_{soft} = \{C_1, \ldots, C_m\}$, $\varphi_{hard} = \{C_{m+1}, \ldots, C_{m+m'}\}$ and $\varphi_{plain} = \varphi_{soft} \cup \varphi_{hard}$.

The set of variables occurring in a formula φ is noted as $\mathrm{var}(\varphi)$.

Definition 1 *(Partial and total truth assignment).* A (*partial*) *truth assignment* is a function $I : X \to \{0, 1\}$, where $X \subset \mathcal{X}$. This function can be extended to variables from $\mathcal{X} \setminus X$, literals, clauses, SAT formulas and MaxSAT formulas, resulting into a function from formulas to formulas, as follows.

For variables $x_i \notin X$, $I(x_i) = x_i$.

For literals, $I(\neg x_i) = 1 - I(x_i)$, if $x_i \in X$; otherwise, $I(l) = l$.

For clauses, $I(l_1 \vee \cdots \vee l_r) = I(l_1) \vee \cdots \vee I(l_r)$ and $I(\square) = 0$, simplified considering $1 \vee C = 1$ and $0 \vee C = C$.

For SAT formulas $I(C_1 \wedge \cdots \wedge C_r) = I(C_1) \wedge \cdots \wedge I(C_r)$ and $I(\emptyset) = 1$, simplified considering $1 \wedge \varphi = \varphi$ and $0 \wedge \varphi = 0$.

For MaxSAT formulas, $I(\{(C_1, w_1), \ldots, (C_m, w_m)\}) = \{(I(C_1), w_1), \ldots, (I(C_m), w_m)\}$ and $I(\emptyset) = 1$, considering $\{(1, w)\} \cup \varphi = \varphi$ and $\{(0, w)\} \cup \varphi = \{(\square, w)\} \cup \varphi$ and simplifying $\{(C, w_1), (C, w_2)\} \cup \varphi = \{(C, w_1 + w_2)\} \cup \varphi$.

Given a SAT formula φ, a *total truth assignment* is a truth assignment I with domain $\text{var}(\varphi)$.

Notice that for any SAT formula, $I(\varphi)$ may be 0, 1, or a partial instantiation of φ, whereas for total truth assignments, $I(\varphi)$ is either 0 or 1.

Example 2. Given $\varphi = \{(\neg y, 6), (x \vee y, 2), (x \vee z, 3), (y \vee z, 2)\}$ 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 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. Given a MaxSAT formula an unsatisfiable core is a subset of clauses, without weights, that is unsatisfiable.

In our algorithms the unsatisfiable cores are described by the indexes of the soft clauses belonging to the core. In this context, given a MaxSAT formula $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$, a *core* is a set $A = \{i_1, \ldots, i_s\} \subseteq \{1, \ldots, m\}$ of indexes of soft clauses such that $\varphi_{hard} \cup \{C_{i_1}, \ldots, C_{i_s}\}$ is unsatisfiable.

Definition 3 *(Optimal cost and assignment).* Given a weighted partial MaxSAT formula φ and a truth assignment $I : \text{var}(\varphi) \to \{0, 1\}$, the *optimal cost* of a formula is the minimal cost of all its total truth assignments:

$$\text{cost}(\varphi) = \min\left\{ \sum_{\substack{(C_i, w_i) \in \varphi \\ I(C_i) = 0}} w_i \;\middle|\; I : \text{var}(\varphi) \to \{0, 1\} \right\}$$

An *optimal assignment* is an assignment with optimal cost.

A MaxSAT formula is said to be satisfiable if it has cost zero.

Example 4. Given $\varphi = \{(\neg y, 6), (x \vee y, 2), (x \vee z, 3), (y \vee z, 2)\}$ and $I : \{y, z\} \to \{0, 1\}$ such that $I(y) = 0$ and $I(z) = 0$, we have $\text{cost}(\varphi) = 0$ and $\text{cost}(I(\varphi)) = 2$.

Notice that for any weighted partial MaxSAT formula φ and total truth assignment $I : \text{var}(\varphi) \to \{0, 1\}$ we have $I(\varphi) = \{(\square, \text{cost}(I(\varphi)))\}$. For any MaxSAT formula φ and truth assignment I we have $\text{cost}(\varphi) \leqslant \text{cost}(I(\varphi))$. 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.

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 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.

A *linear pseudo-boolean constraint* is an inequality of the form $w_1 x_1 + \cdots + w_n x_n \; op \; k$, where $op \in \{\leqslant, \geqslant, =, >, <\}$, $k \in \mathbb{N}$, $w_i \in \mathbb{N}$, and x_i are boolean variables. A *cardinality constraint* is a linear pseudo-boolean constraint where the coefficients w_i are equal to 1.

3. SAT-based MaxSAT

In this section we describe a simple SAT-based approach for solving MaxSAT. The detailed descriptions of our algorithms appear in the following sections.

A Weighted Partial MaxSAT problem $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ can be solved through the resolution of a sequence of SAT instances as follows. Let φ_k be a SAT formula that is satisfiable if, and only if, φ has an assignment with cost smaller or equal to k. One way to encode φ_k is to extend every soft clause C_i with a fresh auxiliary variable b_i, and add the conversion to CNF of the *linear pseudo-boolean constraint* $\sum_{i=1}^{m} w_i b_i \leqslant k$. Then

$$\varphi_k = \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\} \cup \mathit{CNF}\left(\sum_{i=1}^{m} w_i b_i \leqslant k \right)$$

If the cost of the optimal assignment to φ is k_{opt}, then the SAT problems φ_k, for $k \geqslant k_{opt}$, are satisfiable, while for $k < k_{opt}$ are unsatisfiable. Therefore, the search of the cost of the optimal assignment to φ corresponds to the precise location of this transition between satisfiable and unsatisfiable SAT formulas.

Notice that k may range from 0 to $\sum_{i=1}^{m} w_i$ (the sum of the weights of the soft clauses). This encoding of φ_k ensures that the set of all satisfying assignments of $\varphi_{k_{opt}}$ (with variables restricted to the variables of φ) is the set of optimal assignments of φ.

The search for the value k_{opt} can be done following different strategies; searching from $k = 0$ to k_{opt} (increasing k while φ_k is unsatisfiable); from $k = \sum_{i=1}^{m} w_i$ to some value smaller than k_{opt} (decreasing k while φ_k is satisfiable); or alternating unsatisfiable and satisfiable φ_k until the algorithm converges to k_{opt} (for instance, using a binary search scheme). Solvers using the first approach are sometimes called unsatisfiability-based solvers, while solvers using the second are called satisfiability-based solvers.

The key point to boost the efficiency of these approaches is to know whether we can exploit any additional information from the execution of the SAT solver for the next runs.

The approach used by the solver *sat4j* [22] or *qms* (QMaxSAT) [21] is to exploit the information of the satisfiable formulas φ_k, encoded using auxiliary variables as above. The same idea was initially used by the solver miniSAT$^+$ [14] for solving pseudo-boolean optimization problems. It starts with $k = \sum_{i=1}^{m} w_i$. Whenever the underlying SAT solver returns satisfiable for φ_k, it checks the satisfying assignment I and sets the next k equal to the sum of the weights of the soft clauses with auxiliary variable set to true, minus one, i.e. $k = \sum_{I(b_i)=1} w_i - 1$. If the SAT solver returns unsatisfiable, then the algorithm stops and the optimal cost is set to $k + 1$. If the reported optimal cost is $\sum_{i=1}^{m} w_i + 1$ (which may be considered as an infinite cost) the MaxSAT formula is unsatisfiable.

The Fu and Malik algorithm [17,16] – described originally only for Partial MaxSAT – exploits the information of the unsatisfiable formulas φ_k. It starts with $k = 0$ and increases this value until φ_k is satisfiable. Whenever φ_k is unsatisfiable, the SAT solver also returns an unsatisfiable core, that is not necessarily minimal. The next φ_k is constructed adding auxiliary variables to the soft clauses belonging to the core, and adding *cardinality constraints* on these variables, stating that exactly one of them has to be true. This prevents the solver from finding the same unsatisfiable core in the next iterations. Therefore, contrarily to the encoding of φ_k given above, in this encoding we may have more than one auxiliary variable in a clause, or none. This approach is quite effective since it allows to solve more efficiently the unsatisfiable φ_k instances, due to the addition of cardinality constraints at each iteration. However, this approach has also a weakness. A soft clause can be extended with more than one auxiliary variable (if it belongs to more than one core). This can hamper the efficiency of the SAT solver. In Section 10, we describe the algorithm PM2 that uses only one auxiliary variable for each clause. This allows us to encode more efficiently the information provided by the unsatisfiable cores.

The solvers *wpm1* [4], *wbo* and *msuncore* [25,26] are based on the extension of the Fu and Malik algorithm to the Weighted Partial MaxSAT problem. This weighted extension of the Fu and Malik algorithm is described in Section 7. PM2 only works for Partial MaxSAT. In Section 11, we show how the same idea is extended to Weighted Partial MaxSAT by the algorithm WPM2.

The solver *maxhs* [13] also only increases the lower bound to reach the optimum. It also solves a sequence of SAT formulas, but which are simplifications of the initial MaxSAT formula. In this case, the arithmetic reasoning is handled in a different hitting set problem.

Finally, the approach followed by the solvers msu4.0 [31], *pwbo1.1* [32] and *bincd* [19], alternate phases in which the SAT solver reports satisfiable (giving an upper bound of k_{opt}) with other in which it reports unsatisfiable (giving lower bounds for k_{opt}). They also use a unique auxiliary variable for each clause.

The efficiency of these solvers depends critically on which SAT solver we use, and how we encode the cardinality and pseudo-boolean constraints.

4. MaxSAT reducibility

Our algorithms solve a MaxSAT formula by successively transforming it until we get a trivially solvable 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 5 *(Cost-preservation, MaxSAT equivalence and MaxSAT reducibility).* 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 $I : \text{var}(\varphi_1) \cup \text{var}(\varphi_2) \to \{0, 1\}$, 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 $I : \text{var}(\varphi_1) \to \{0, 1\}$, 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 $\text{var}(\varphi_1)$.

In the following we show some examples of the notions of Definition 5.

Example 6. The following example shows a formula transformation that preserves the cost, but not MaxSAT reducibility. Consider $\varphi_1 = \{(x, 2), (\neg x, 1)\}$ 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 $\varphi_1 = \{(x, 2), (\neg x, 1)\}$ and $\varphi_3 = \{(\square, 1), (x, 1), (x \vee y, 1), (\neg x \vee z, 1), (\neg y \vee \neg z, \infty)\}$. We have that φ_1 is MaxSAT reducible to φ_3.[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 cost-preserving transformation is the weakest of all three notions, and suffices to prove the soundness of some 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 do 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.

The notion of MaxSAT equivalence was implicitly defined in [12]. In this paper a MaxSAT resolution rule that preserves MaxSAT equivalence is defined, and proved complete for MaxSAT.

Eventually, the transformations we apply to a formula reach a state of saturation where the cost of the formula is obtained. Moreover, the resulting formula also describes the set of optimal assignments.

Definition 7. We say that $\{(\square, \omega)\} \cup \varphi'$ is a *saturation* of φ, if φ is MaxSAT reducible to $\{(\square, \omega)\} \cup \varphi'$ and φ' is satisfiable. The same notion can be defined for cost-preserving and MaxSAT equivalent transformations.

Next we prove some basic facts about MaxSAT reducibility that will be needed to prove the correctness of our algorithms.

Lemma 8.

(1) *If φ_1 is MaxSAT-reducible to φ_2 and* $\text{var}(\varphi_2) \cap \text{var}(\varphi_3) \subseteq \text{var}(\varphi_1)$, *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* $\text{var}(\varphi_1) \cap \text{var}(\varphi_3) \subseteq \text{var}(\varphi_2)$, *then φ_1 is MaxSAT-reducible to φ_3.*
(3) *If $\{(\square, w)\} \cup \varphi'$ is a saturation of φ, then*
   (a) $w = \text{cost}(\varphi)$ *and*
   (b) $I : \text{var}(\varphi) \to \{0, 1\}$ *is an optimal assignment of φ iff $I(\varphi')$ is satisfiable.*

Proof. (1) Assume that φ_1 is MaxSAT-reducible to φ_2 and that φ_3 is a set of weighted clauses satisfying $\text{var}(\varphi_3) \cap \text{var}(\varphi_2) \setminus \text{var}(\varphi_1) = \emptyset$. Suppose now that $I : \text{var}(\varphi_1 \cup \varphi_3) \to \{0, 1\}$. Then, since I assigns values to all the variables of $\varphi_1 \cup \varphi_3$,

$$\text{cost}\big(I(\varphi_1 \cup \varphi_3)\big) = \sum_{\substack{(C_i, w_i) \in \varphi_1 \cup \varphi_3 \\ I(C_i) = 0}} w_i = \sum_{\substack{(C_i, w_i) \in \varphi_1 \\ I(C_i) = 0}} w_i + \sum_{\substack{(C_i, w_i) \in \varphi_3 \\ I(C_i) = 0}} w_i = \text{cost}\big(I(\varphi_1)\big) + \text{cost}\big(I(\varphi_3)\big)$$

Similarly, since I assigns all variables from φ_3, we have $I(\varphi_3) = \{(\square, \text{cost}(I(\varphi_3)))\}$, hence

$$\text{cost}\big(I(\varphi_2 \cup \varphi_3)\big) = \text{cost}\big(I(\varphi_2) \cup \big\{\big(\square, \text{cost}\big(I(\varphi_3)\big)\big)\big\}\big) = \text{cost}\big(I(\varphi_2)\big) + \text{cost}\big(I(\varphi_3)\big)$$

Since φ_1 is MaxSAT-reducible to φ_2, if we restrict I to the variables of φ_1 obtaining say I', then $\text{cost}(I'(\varphi_1)) = \text{cost}(I'(\varphi_2))$. On the other hand, clearly $I(\varphi_1) = I'(\varphi_1)$, and since $\text{var}(\varphi_3) \cap \text{var}(\varphi_2) \setminus \text{var}(\varphi_1) = \emptyset$, $I(\varphi_2) = I'(\varphi_2)$. Also, by the initial assumption, $I'(\varphi_1) = I'(\varphi_2)$. Therefore, $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$, and (1) is proved.

(2) To prove this result we will need two statements that can be summarized as: in the definition of MaxSAT reducibility we can restrict the domain of interpretations for free, and we can enlarge it with variables not occurring in the formulas.

Statement 1. For any pair of MaxSAT formulas φ_1 and φ_2, and sets of variables V_1 and V_2, if $V_2 \subseteq V_1$ and $\forall I : V_1 \to \{0, 1\}$ $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$, then $\forall I : V_2 \to \{0, 1\}$ $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$. This results proves, for instance that MaxSAT equivalence is stronger than MaxSAT reducibility because $\text{var}(\varphi_1) \subseteq \text{var}(\varphi_1) \cup \text{var}(\varphi_2)$.

Statement 2. For any pair of MaxSAT formulas φ_1 and φ_2, and sets of variables V_1 and V_2, if $V_2 \cap \text{var}(\varphi_1 \cup \varphi_2) = \emptyset$ and $\forall I : V_1 \to \{0, 1\}$ $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$ then $\forall I : V_1 \cup V_2 \to \{0, 1\}$ $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$.

[1] To prove that φ_1 is MaxSAT reducible to φ_3, 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 $I_1(\varphi_3) = \{(\square, 2), (y, 1), (\neg y \vee \neg z, \infty)\}$ that have the same cost 2. In the second case, we obtain $I_2(\varphi_1) = \{(\square, 1)\}$ and $I_2(\varphi_3) = \{(\square, 1), (z, 1), (\neg y \vee \neg z, \infty)\}$ that have also the same cost 1.

Now, since φ_1 is MaxSAT reducible to φ_2, we have $\forall I : \mathrm{var}(\varphi_1) \to \{0,1\}\ \mathrm{cost}(I(\varphi_1)) = \mathrm{cost}(I(\varphi_2))$.

Since φ_2 is MaxSAT reducible to φ_3, we have $\forall I : \mathrm{var}(\varphi_2) \to \{0,1\}\ \mathrm{cost}(I(\varphi_2)) = \mathrm{cost}(I(\varphi_3))$. By statement 1, we can restrict the domain getting $\forall I : \mathrm{var}(\varphi_1) \cap \mathrm{var}(\varphi_2) \to \{0,1\}\ \mathrm{cost}(I(\varphi_2)) = \mathrm{cost}(I(\varphi_3))$. By statement 2, since $\mathrm{var}(\varphi_1) \cap \mathrm{var}(\varphi_3) \subseteq \mathrm{var}(\varphi_2)$, we have $\forall I : \mathrm{var}(\varphi_1) \cap \mathrm{var}(\varphi_2) \cup (\mathrm{var}(\varphi_1) \setminus \mathrm{var}(\varphi_2)) \to \{0,1\}\ \mathrm{cost}(I(\varphi_2)) = \mathrm{cost}(I(\varphi_3))$. Hence, $\forall I : \mathrm{var}(\varphi_1) \to \{0,1\}$ $\mathrm{cost}(I(\varphi_2)) = \mathrm{cost}(I(\varphi_3))$.

Therefore, $\forall I : \mathrm{var}(\varphi_1) \to \{0,1\}\ \mathrm{cost}(I(\varphi_1)) = \mathrm{cost}(I(\varphi_3))$.

(3) Let $\{(\square, w)\} \cup \varphi'$ be the saturation of φ. Then φ' is satisfiable, and φ is MaxSAT-reducible to $\{(\square, w)\} \cup \varphi'$. So for every assignment I of the variables of φ,

$$\begin{aligned}\mathrm{cost}\big(I(\varphi)\big) &= \mathrm{cost}\big(I\big(\{(\square, w)\} \cup \varphi'\big)\big)\\ &= \mathrm{cost}\big(I\big(\{(\square, w)\}\big)\big) + \mathrm{cost}\big(I\big(\varphi'\big)\big)\\ &= w + \mathrm{cost}\big(I\big(\varphi'\big)\big)\end{aligned}$$

From the previous equations we conclude that for any assignment I to the variables of φ, $\mathrm{cost}(I(\varphi)) \geqslant w$, and that any assignment such that $I(\varphi')$ is satisfiable must be optimal for φ.

Since φ' is satisfiable, let I' be an assignment that satisfies φ'. Say that I is the restriction of I' to the variables of φ. For such an assignment, $I(\varphi')$ is satisfiable, and $\mathrm{cost}(I(\varphi')) = 0$. At this point we can conclude that $w = \mathrm{cost}(\varphi)$.

It is clear that, for any assignment $I : \mathrm{var}(\varphi) \to \{0,1\}$, $I(\varphi')$ is satisfiable iff $\mathrm{cost}(I(\varphi)) = w$, and $\mathrm{cost}(I(\varphi)) = w$ iff I is an optimal assignment for φ. □

Example 9. Notice that the side condition of Lemma 8(1) is necessary. For instance, if we take $\varphi_1 = \{(\square, 1)\}$, $\varphi_2 = \{(x, 1), (\neg x, \infty)\}$ and $\varphi_3 = \{(x, 1)\}$, where the side condition $\mathrm{var}(\varphi_2) \cap \mathrm{var}(\varphi_3) = \{x\} \not\subseteq \emptyset = \mathrm{var}(\varphi_1)$ 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 8(2) is also necessary. For instance, if we take $\varphi_1 = \{(x, 1), (\neg x, 1)\}$, $\varphi_2 = \{(\square, 1)\}$ and $\varphi_3 = \{(x, 1), (\neg x, \infty)\}$, where the side condition $\mathrm{var}(\varphi_1)) \cap \mathrm{var}(\varphi_3) = \{x\} \not\subseteq \emptyset = \mathrm{var}(\varphi_2)$ 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 8(1) and (2) (see Example 9) 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.

Lemma 8(1) holds for MaxSAT equivalence, even if φ_3 does not satisfy the restriction $\mathrm{var}(\varphi_3) \cap \mathrm{var}(\varphi_2) \subseteq \mathrm{var}(\varphi_1)$. However, it does not hold for cost-preserving transformations: $\varphi_1 = \{(x, 1), (\neg x, 2)\}$ and $\varphi_2 = \{(\square, 1)\}$ have the same cost 1, however $\varphi_1 \cup \{(x, 1)\}$ and $\varphi_2 \cup \{(x, 1)\}$ have distinct costs. Lemma 8(2) holds for cost-preserving and MaxSAT equivalence transformations. Moreover, these two relations are also symmetric. Finally, Lemma 8(3) also holds for cost-preserving and MaxSAT equivalence transformations.

5. The Fu and Malik's algorithm

Before giving the full version of our algorithms, we will present the original Fu and Malik's algorithm [17,16] for Partial MaxSAT, and show the correction of the algorithm. We will use parts of the proof of correctness of this algorithm to show the correctness of our WPM1 and PM2 algorithms.

The algorithm consists in iteratively calling a SAT solver on a working formula φ. This corresponds to the line $(st, \varphi_c) := SAT(\{C \mid (C, w) \in \varphi\})$. 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 (b_i^s in the code, with superindex s indicating the number of the core and subindex i indicating the index of the soft clause), 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 ($CNF(\sum_{i \in A_s} b_i^s = 1)$ 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.

Before we prove the correctness of the algorithm, we present an example of the execution of the algorithm.

Example 10. 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. The variable x_i means that pigeon i goes to the only hole.

Algorithm 1: The pseudo-code of the Fu and Malik algorithm (with a minor correction).

```
Input: φ = {(C_1, 1), ..., (C_m, 1), (C_{m+1}, ∞), ..., (C_{m+m'}, ∞)}
1:  if SAT({C_i | w_i = ∞}) = (UNSAT, _) then return (∞, ∅)        ▷Hard clauses are unsatisfiable
2:  cost := 0                                                       ▷Optimal
3:  s := 0                                                          ▷Counter of cores
4:  while true do
5:      (st, φ_c) := SAT({C_i | (C_i, 1) ∈ φ})                      ▷Call the SAT solver without weights
6:      if st = SAT then return (cost, φ)
7:      s := s + 1
8:      A_s := ∅                                                    ▷Indexes of the core
9:      foreach C_i ∈ φ_c do
10:         if w_i ≠ ∞ then                                         ▷If the clause is soft
11:             b_i^s := new_variable( )
12:             φ := φ \ {(C_i, 1)} ∪ {(C_i ∨ b_i^s, 1)}            ▷Add blocking variable
13:             A_s := A_s ∪ {i}
14:     φ := φ ∪ {(C, ∞) | C ∈ CNF(Σ_{i∈A_s} b_i^s = 1)}             ▷Add cardinality constraint as hard clauses
15:     cost := cost + 1
```

$$\varphi = \{(x_1,1),(x_2,1),(x_3,1),(x_4,1),(x_5,1),(\neg x_1 \vee \neg x_2,\infty),\ldots,(\neg x_4 \vee \neg x_5,\infty)\}$$

$$\begin{aligned}\varphi_1 = \{\ & (x_1 \vee b_1^1,\ 1),\\ & (x_2 \vee b_2^1,\ 1),\\ & (x_3,\ 1),\\ & (x_4,\ 1),\\ & (x_5,\ 1)\ \} \cup\\ & \{(\neg x_i \vee \neg x_j,\infty) \mid i<j\} \cup\\ & CNF(b_1^1 + b_2^1 = 1,\infty)\end{aligned}$$

Suppose that applying the Fu and Malik algorithm, the SAT solver computes the (minimal) unsatisfiable core $A_1 = \{1,2\}$. The new formula will be as shown on the left. At this point, the variable cost takes value 1.

If the next unsatisfiable cores found by the SAT solver are $A_2 = \{3,4\}$ and $A_3 = \{1,2,3,4\}$, then the new formula will be, respectively:

$$\begin{aligned}\varphi_2 = \{\ & (x_1 \vee b_1^1,\ 1),\\ & (x_2 \vee b_2^1,\ 1),\\ & (x_3 \vee b_3^2,\ 1),\\ & (x_4 \vee b_4^2,\ 1),\\ & (x_5,\ 1)\ \} \cup\\ & \{(\neg x_i \vee \neg x_j,\infty) \mid i<j\} \cup\\ & CNF(b_1^1 + b_2^1 = 1,\infty) \cup\\ & CNF(b_3^2 + b_4^2 = 1,\infty)\end{aligned}$$

$$\begin{aligned}\varphi_3 = \{\ & (x_1 \vee b_1^1 \vee b_1^3,\ 1),\\ & (x_2 \vee b_2^1 \vee b_2^3,\ 1),\\ & (x_3 \vee b_3^2 \vee b_3^3,\ 1),\\ & (x_4 \vee b_4^2 \vee b_4^3,\ 1),\\ & (x_5,\ 1)\ \} \cup\\ & \{(\neg x_i \vee \neg x_j,\infty) \mid i<j\} \cup\\ & CNF(b_1^1 + b_2^1 = 1,\infty) \cup\\ & CNF(b_3^2 + b_4^2 = 1,\infty) \cup\\ & CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1,\infty)\end{aligned}$$

After the third iteration, the variable cost has value 3. Finally, after finding the core $A_4 = \{1,2,3,4,5\}$ we get the following satisfiable MaxSAT formula:

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

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.

The following lemma is part of the correctness of the Fu and Malik algorithm.

Lemma 11. *Let* $\varphi = \{(C_1, 1), \ldots, (C_m, 1), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ *be a partial MaxSAT formula,* $A \subseteq \{1, \ldots, m\}$ *be a core, and*

$$\varphi' = \varphi \setminus \{(C_i, 1) \mid i \in A\} \cup \{(C_i \vee b_i, 1) \mid i \in A\} \cup \left\{(C, \infty) \;\middle|\; C \in CNF\left(\sum_{i \in A} b_i = 1\right)\right\} \cup \{(\square, 1)\}$$

where b_i*, for* $i \in A$*, are new variables. Then,* φ *is MaxSAT reducible to* φ'*.*

Proof. Let $I : \mathrm{var}(\varphi) \to \{0, 1\}$ be a truth assignment. If $I(\varphi_{hard}) = 0$ then $\mathrm{cost}(I(\varphi)) = \mathrm{cost}(I(\varphi')) = \infty$ and the lemma holds.

Otherwise, we can assume that $I(\varphi_{hard}) = 1$, and, since $\{C_i \mid i \in A\} \cup \varphi_{hard}$ is an unsatisfiable core, I falsifies some soft clause C_i where $i \in A$.

Now, consider an optimal assignment $I' : \mathrm{var}(I(\varphi')) \to \{0, 1\}$ for $I(\varphi')$, i.e. $\mathrm{cost}(I(\varphi')) = \mathrm{cost}(I'(I(\varphi')))$. For every clause C_i with $i \notin A$, we have $I'(I(C_i)) = I(C_i)$. Since I' is optimal, it satisfies $CNF(\sum_{i \in A} I'(b_i) = 1)$. Let $j \in A$ be the unique index satisfying $I'(b_j) = 1$. For the rest of indexes $k \in A$ and $k \neq j$, $I'(b_k) = 0$, hence $I' \circ I(C_k \vee b_k) = I(C_k)$. On the other hand, since I' is optimal and I falsifies some of the soft clauses in the core, the clause C_j must be one of such falsified clauses: $I(C_j) = 0$. We have $I' \circ I(C_j \vee b_j) = 1$, but I' falsifies $\square$. Hence, $\mathrm{cost}(I(\{C_i \mid i \in A\})) = \mathrm{cost}(I' \circ I(\{(C_i \vee b_i, 1) \mid i \in A\} \cup \{(\square, 1)\}))$. We conclude $\mathrm{cost}(I(\varphi')) = \mathrm{cost}(I'(I(\varphi'))) = \mathrm{cost}(I(\varphi))$. □

Theorem 12. *Fu and Malik (see Algorithm 1) is a correct algorithm for Partial MaxSAT.*

Moreover, when for a formula φ*, the algorithm returns* $(\mathrm{cost}, \varphi')$*, then* $\{(\square, \mathrm{cost})\} \cup \varphi'$ *is a saturation of* φ*.*

Proof. In each iteration of the while loop, if the SAT solver returns UNSAT 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 11 shows that φ is MaxSAT reducible to $\{(\square, 1)\} \cup \varphi'$, hence cost-preserving $\mathrm{cost}(\varphi) = \mathrm{cost}(\{(\square, 1)\} \cup \varphi')$.

When hard clauses are satisfiable, Lemma 11 also shows that when variable cost is equal to the optimal cost of the original φ, then the actual formula is satisfiable. This ensures that, either hard clauses are unsatisfiable, and the algorithm terminates in the second line, or they are satisfiable, and the algorithm terminates in k_{opt} iterations of the main loop.

Notice that, to prove the first statement of the theorem, we only need to prove that the Fu and Malik transformation is cost-preserving. However, Lemma 11 proves that the transformation is a MaxSAT reduction. This fact, together with the transitivity of MaxSAT reductions (stated in Lemma 8), proves the second statement of the theorem, i.e. that the resulting formula is a saturation of the original one. □

The original version of the Fu and Malik algorithm, published in [17], was slightly different. After line 13 there was the following line:

13bis: **if** $BV = \emptyset$ **then return** UNSAT,

that we have replaced by line 1 of Algorithm 1.

This change ensures that the algorithm terminates even when the set of hard clauses is unsatisfiable. Recall that the SAT solver does not guarantee to return minimal unsatisfiable cores, and, if it always includes an irrelevant soft clause in the core, then the original algorithm does not terminate.

6. 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 Fu and Malik 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 Fu and Malik 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 not concerned with eliminating natural symmetries of a MaxSAT formula as in [28], 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 13. Consider again the pigeon-hole formula PHP_1^5 of Example 10. The working formula φ_3 from the previous section is still unsatisfiable, this is the reason to find a fourth core A_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 $\neg x_i \vee \neg x_j$, 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 Fu and Malik algorithm, and we have obtained the set of cores $\{A_1, \ldots, A_s\}$. Now, we add the hard clauses:

$$b_i^s \rightarrow \neg b_j^l \quad \text{for } l = 1, \ldots, s-1 \text{ and } i, j \in A_l \cap A_s \text{ and } j > i$$

This can be done adding the following line to the Fu and Malik algorithm, just after line 14.

14bis: **foreach** $l = 1, \ldots, s-1$ **do**
  foreach $i, j \in A_l \cap A_s$ **and** $j > i$ **do**
    $\varphi := \varphi \cup \{(\neg b_i^s \vee \neg b_j^l, \infty)\}$

These clauses imply that in Example 13 we choose the model on the left rather than the one on the right.

Example 14. For Example 13, after finding the third unsatisfiable core A_3, we would add the following clauses to break symmetries (written in form of implications):

$$b_1^3 \rightarrow \neg b_2^1$$
$$b_3^3 \rightarrow \neg b_4^2$$

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 forth core A_4, the set of clause for breaking symmetries are (written in compact form):

$$b_1^3 \rightarrow \neg b_2^1$$
$$b_3^3 \rightarrow \neg b_4^2$$
$$b_1^4 \rightarrow \left(\neg b_2^1 \wedge \neg b_2^3 \wedge \neg b_3^3 \wedge \neg b_4^3\right)$$
$$b_2^4 \rightarrow \left(\neg b_3^3 \wedge \neg b_4^3\right)$$
$$b_3^4 \rightarrow \left(\neg b_4^2 \wedge \neg b_4^3\right)$$

The following lemma is used to prove the correctness of the Fu and Malik algorithm with symmetry breaking. Notice that φ is the formula passed by the Fu and Malik algorithm to the SAT solver, whereas the formula φ' is the one passed by the Fu and Malik algorithm with symmetry breaking.

Lemma 15. *For any sequence of cores $A_1, \ldots, A_s$ the following two formulas are equi-satisfiable.*

$$\varphi = \varphi_{hard} \cup \left\{ C_i \vee \bigvee_{i \in A_j} b_i^j \;\middle|\; i = 1, \ldots, m \right\} \cup \bigcup_{j=1}^{s} CNF\left(\sum_{i \in A_j} b_i^j = 1 \right)$$

$$\varphi' = \varphi \cup \bigcup_{1 \leqslant k < l \leqslant s} \; \bigcup_{\substack{i, j \in A_k \cap A_l \\ i < j}} \left\{ \neg b_i^k \vee \neg b_j^l \right\}$$

Proof. Obviously, when φ' is satisfiable, φ is also satisfiable.

Now, assume that I satisfies φ, but falsifies φ'. Then, there exist four values i, j, k, l such that $1 \leqslant k < l \leqslant s$ and $i, j \in A_k \cap A_l$ and $i < j$ and $I(b_i^k) = I(b_j^l) = 1$. Since for these k and l, the assignment I evaluates $\sum_{i \in A_k} I(b_i^k) = 1$ and $\sum_{i \in A_l} I(b_i^l) = 1$, and $i \in A_l$ and $j \in A_k$, we have $I(b_i^l) = I(b_j^k) = 0$. We can construct a new assignment I' that assigns the same values as I, except for the following four variables:

$$I\big(b_i^k\big) = I\big(b_j^l\big) = 1 \qquad I'\big(b_i^k\big) = I'\big(b_j^l\big) = 0$$
$$I\big(b_i^l\big) = I\big(b_j^k\big) = 0 \qquad I'\big(b_i^l\big) = I'\big(b_j^k\big) = 1$$

It is not difficult to see that I' also satisfies φ.

This way, we can define a *rewriting rule* on assignments. Concatenating these transformations we can eventually construct a sequence of assignment transformations $I \longrightarrow I' \longrightarrow \cdots \longrightarrow I^n$ such that I^n satisfies φ'. We will prove that this *finite* sequence always exists.

Notice that, for any assignment satisfying φ, either we can transform it, or it also satisfies φ'. Therefore, it suffices to prove that the rewriting rule is *terminating*, i.e. there does not exist infinite sequences of transformations $I \longrightarrow I' \longrightarrow \cdots$. We define a *weight* for each of these assignments as follows:

$$\mathit{weight}(I) = \sum_{I(b_i^j)=1} i \cdot j$$

When we transform I into I' the weight is transformed as follows.

$$\mathit{weight}\big(I'\big) = \mathit{weight}(I) - i \cdot l - j \cdot k + i \cdot k + j \cdot l = \mathit{weight}(I) + (j - i) \cdot (l - k)$$

Since $i < j$ and $k < l$, the weight of an assignment strictly increases in each transformation $\mathit{weight}(I') > \mathit{weight}(I)$. On the other hand, the weight of an assignment I satisfying $\sum_{i \in A_k} I(b_i^k) = 1$ for $k = 1, \ldots, s$ is bounded by $m \cdot s^2$, hence this bound the length of any sequence of assignment transformations. □

Theorem 16. *The Fu and Malik algorithm with symmetry breaking is a correct algorithm for MaxSAT.*

Proof. The technical details of the proof are in Lemma 15. The formulas φ and φ' stated in the lemma are the formulas send by the classical Fu and Malik algorithm and the symmetry breaking Fu and Malik algorithm to the SAT solver, respectively. They are equi-satisfiable, therefore we will obtain the same optimal (number of calls to the SAT solver). □

The following example shows that although the proposed method for breaking symmetries is correct, it can still be improved.

Example 17. Consider the following set of cores $A_1 = \{2, 3\}$, $A_2 = \{1, 3\}$, $A_3 = \{1, 2\}$. Assume that after finding these cores, the following formula sent to the SAT solver is satisfiable.

$$\varphi = \varphi_{\mathit{hard}} \cup \big\{C_1 \vee b_1^2 \vee b_1^3, C_2 \vee b_2^1 \vee b_2^3, C_3 \vee b_3^1 \vee b_3^2, \ldots\big\}$$
$$\cup \big\{\mathit{CNF}\big(b_2^1 + b_3^1 = 1\big),\ \mathit{CNF}\big(b_1^2 + b_3^2 = 1\big),\ \mathit{CNF}\big(b_1^3 + b_2^3 = 1\big)\big\}$$

The symmetry breaking procedure would not add any additional clause (because the intersection of any pair of cores is just a singleton). However, the formula has (at least) two models produced by symmetries in the b's:

$$I\big(b_1^2\big) = I\big(b_2^3\big) = I\big(b_3^1\big) = 1 \qquad I'\big(b_1^2\big) = I'\big(b_2^3\big) = I'\big(b_3^1\big) = 0$$
$$\text{and}$$
$$I\big(b_1^3\big) = I\big(b_2^1\big) = I\big(b_3^2\big) = 0 \qquad I'\big(b_1^3\big) = I'\big(b_2^1\big) = I'\big(b_3^2\big) = 1$$

7. The WPM1 algorithm

Algorithm 2 is the weighted version of the Fu and Malik algorithm described in Section 5. In this algorithm, we iteratively call a SAT solver with a weighted working formula, but excluding the weights. In particular, the difference with respect to Algorithm 1 is that, 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 duplicating the clauses of the core. In one of the copies the clauses have the original weight minus the minimum weight, and in the other copy we put the blocking variables and we give them the minimum weight. Finally we add the cardinality constraint on the blocking variables as in Algorithm 1, and we add w_{min} to the cost.

The following lemmas are used to prove the correctness of the algorithm.

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

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

if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \_)$ **then return** $(\infty, \emptyset)$ ▷Hard clauses are unsatisfiable

$\text{cost} := 0$ ▷Optimal

$s := 0$ ▷Counter of cores

while *true* **do**

  $(st, \varphi_c) := SAT(\{C_i \mid (C_i, w_i) \in \varphi\})$ ▷Call the SAT solver without weights

  if $st = \text{SAT}$ **then return** (cost, φ)

  $s := s + 1$

  $A_s := \emptyset$ ▷Indexes 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**

      $b_i^s := \text{new\_variable}(\,)$

      $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min}),\ (C_i \vee b_i^s, w_{min})\}$ ▷Duplicate soft clauses of the core

      $A_s := A_s \cup \{i\}$

  $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{i \in A_s} b_i^s = 1)\}$ ▷Add cardinality constraint as hard clauses

  $\text{cost} := \text{cost} + w_{min}$

Lemma 18. *Let φ be a weighted partial Max-SAT formula, and let $\exp(\varphi)$ be the natural expansion of φ into an unweighted formula by substituting every soft clause (C, w) of φ by w copies*[2] *of $(C, 1)$. Then, $\text{cost}(I(\varphi)) = \text{cost}(I(\exp(\varphi)))$, for any assignment I.*

Proof. Straightforward. □

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

Lemma 19. *Let $\varphi = \varphi_{soft} \cup \varphi_{hard}$ be a partial MaxSAT formula, $\{C_1, \ldots, C_s\} \cup \varphi_{hard}$ be an unsatisfiable SAT formula,*[3] *and $V = \text{var}(\varphi) \cup \text{var}(\{C_1, \ldots, C_s\})$. Let*

$$\varphi_1 = \varphi \cup \Big\{\underbrace{(C_i \vee b_i, 1), \ldots, (C_i \vee b_i, 1)}_{r \text{ times}} \;\Big|\; i = 1, \ldots, s\Big\} \cup \left\{(C, \infty) \;\middle|\; C \in CNF\left(\sum_{i=1}^{s} b_i = 1\right)\right\}$$

and

$$\varphi_2 = \varphi \cup \bigcup_{j=1}^{r}\Big\{\big(C_i \vee d_i^j, 1\big) \;\Big|\; i = 1, \ldots, s\Big\} \cup \bigcup_{j=1}^{r}\left\{(C, \infty) \;\middle|\; C \in CNF\left(\sum_{i=1}^{s} d_i^j = 1\right)\right\}$$

where $b_i, d_i^j \notin V$ are fresh variables. Then, $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$, for any assignment $I : V \to \{0, 1\}$.

Proof. For any assignment $I : V \to \{0, 1\}$, if $I(\varphi_{hard}) = 0$ then $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2)) = \infty$ and the lemma holds.

Otherwise, $I(\varphi_{hard}) = 1$. Let I_1 be an optimal assignment for $I(\varphi_1)$. We modify I_1 into an assignment I_2 the following way:

$$I_2\big(d_i^j\big) = I_1(b_i) \quad \text{for all } i = 1, \ldots, s \text{ and } j = 1, \ldots, r$$

It is clear that $I_2 \circ I(C) = I_1 \circ I(C)$, for all $C \in \varphi$. Also, $I_2 \circ I(C_i \vee d_i^j) = I_1 \circ I(C_i \vee b_i)$. Therefore, the set of soft clauses in φ_1 falsified by $I_1 \circ I$ is equal to the number of soft clauses in φ_2 falsified by $I_2 \circ I$.

Let now I_2 be an optimal assignment for $I(\varphi_2)$. We modify I_2 into an assignment I_1, the following way:

$$I(b_i) = I'\big(d_i^1\big) \quad \text{for all } i = 1, \ldots, s$$

It is clear that $I_1 \circ I(C) = I_2 \circ I(C)$, for all $C \in \varphi$.

By optimality of I_2, if $I_2(d_i^j) = 1$ for some j, then $I(C_i) = 0$.

For $j = 1, \ldots, r$, let i_j be the index of the d variable set to 1 by I_2, i.e. $I_2(d_{i_j}^j) = 1$ and $I_2(d_i^j) = 0$, for $i \neq i_j$. Notice that $I_1(b_{i_1}) = 1$ and $I_1(b_i) = 0$, for $i \neq i_1$.

[2] Recall that a MaxSAT formula is a multiset of clauses, so we may have several copies of the same clause.

[3] Notice that $C_1, \ldots, C_s$ are not necessarily clauses of φ.

For any $j = 1, \ldots, l$, if $i_j = i_1$, then $I_1 \circ I(C_i \vee b_i) = I_2 \circ I(C_i \vee d_i^1) = I_2 \circ I(C_i \vee d_i^j)$.
Otherwise, if $i_j \neq i_1$, since $I_2 \circ I(d_{i_1}^1) = I_2 \circ (d_{i_j}^j) = 1$, by optimality of I_2, we have

$$I(C_{i_1}) = I(C_{i_j}) = 0$$

Therefore

$$I_1 \circ I(C_{i_1} \vee b_{i_1}) = 1 \quad \text{whereas } I_2 \circ I\big(C_{i_1} \vee d_{i_1}^j\big) = 0$$
$$I_1 \circ I(C_{i_j} \vee b_{i_j}) = 0 \quad \text{whereas } I_2 \circ I\big(C_{i_1} \vee d_{i_1}^j\big) = 1 \quad \text{and}$$
$$I_1 \circ I(C_i \vee b_i) = I_2 \circ I\big(C_i \vee d_i^j\big), \quad \text{for } i \neq i_1, i_j$$

In any case, the number of soft clauses in φ_1 falsified by $I_1 \circ I$ is equal to the number of soft clauses in φ_2 falsified by $I_2 \circ I$. □

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

Lemma 20. *Let $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ be a weighted partial MaxSAT formula, $A \subseteq \{1, \ldots, m\}$ be a core, and $w_{min} = \min\{w_i \mid i \in A\}$. Let*

$$\varphi' = \varphi \setminus \big\{(C_i \vee b_i, w_i) \;\big|\; i \in A\big\} \cup \big\{(C_i, w_i - w_{min}), (C_i \vee b_i, w_{min}) \;\big|\; i \in A\big\}$$
$$\cup \left\{(C, \infty) \;\middle|\; C \in CNF\left(\sum_{i \in A} b_i = 1\right)\right\} \cup \big\{(\square, w_{min})\big\}$$

where $\{b_i \mid i \in A\}$ are fresh variables not occurring in φ. Then, φ is MaxSAT reducible to φ'.

Proof. Let $I : \mathrm{var}(\varphi) \to \{0, 1\}$ be an assignment of the variables of φ. Let $\exp(\varphi)$ be the unweighted expansion of φ. Lemma 18 shows that $\mathrm{cost}(I(\varphi)) = \mathrm{cost}(I(\exp(\varphi)))$, for any assignment, in particular for I. If $I(\varphi_{hard}) = 0$ the cost of both formulas is infinity and we are done. Otherwise, since $\{C_i \mid i \in A\} \cup \varphi_{hard}$ is an unsatisfiable core of $\exp(\varphi)$, and since $w_{min} = \min(w_i \mid i \in A)$, each one of the clauses C_i appears at least w_{min} times in $\exp(\varphi)$. Now we can apply the transformation of Lemma 11 w_{min} times to obtain a formula

$$\varphi_2 = \exp\big(\varphi \setminus \big\{(C_i \vee b_i, w_i) \;\big|\; i \in A\big\}\big) \cup \big\{\big(C_i \vee d_i^j, 1\big) \;\big|\; i \in A,\ j = 1, \ldots, w_{min}\big\}$$
$$\cup \Big\{\underbrace{(C_i, 1), \ldots, (C_i, 1)}_{w_i - w_{min}\ times} \;\Big|\; i \in A\Big\}$$
$$\cup \bigcup_{j=1}^{w_{min}} \left\{(C, \infty) \;\middle|\; C \in CNF\left(\sum_{i \in A} d_i^j = 1\right)\right\}$$
$$\cup \Big\{\underbrace{(\square, 1), \ldots, (\square, 1)}_{w_{min}\ times}\Big\}$$

that is a MaxSAT reduction of $\exp(\varphi)$. Hence $\mathrm{cost}(I(\exp(\varphi))) = \mathrm{cost}(I(\varphi_2))$ because $\mathrm{var}(\varphi) = \mathrm{var}(\exp(\varphi))$.
Now by Lemma 19, the following formula

$$\varphi_3 = \exp\big(\varphi \setminus \big\{(C_i \vee b_i, w_i) \;\big|\; i \in A\big\}\big) \cup \Big\{\underbrace{(C_i \vee b_i, 1), \ldots, (C_i \vee b_i, 1)}_{w_{min}\ times} \;\Big|\; i \in A\Big\}$$
$$\cup \Big\{\underbrace{(C_i, 1), \ldots, (C_i, 1)}_{w_i - w_{min}\ times} \;\Big|\; i \in A\Big\}$$
$$\cup \left\{(C, \infty) \;\middle|\; C \in CNF\left(\sum_{i \in A} b_i = 1\right)\right\}$$
$$\cup \Big\{\underbrace{(\square, 1), \ldots, (\square, 1)}_{w_{min}\ times}\Big\}$$

satisfies $\mathrm{cost}(I(\varphi_2)) = \mathrm{cost}(I(\varphi_3))$, for the fixed I, because it has domain $\mathrm{var}(\varphi)$. Now using again Lemma 18, φ_3 satisfies $\mathrm{cost}(I(\varphi_3)) = \mathrm{cost}(I(\varphi'))$, for any assignment I. We conclude then that φ' is a MaxSAT reduction of φ. □

Theorem 21. *WPM1 (see Algorithm 2) is a correct algorithm for Weighted Partial MaxSAT.*
Moreover, when for a formula φ, the algorithm returns (cost, φ'), *then* $\{(\square, \text{cost})\} \cup \varphi'$ *is a saturation of* φ.

Proof. The theorem is proved iterating Lemma 20 for every execution of the loop of the algorithm, and an argument similar to the proof of Theorem 12 for the termination. □

Example 22. In the following we show the execution of WPM1 on the formula $\varphi = \{(x, 1), (y, 2), (z, 3), (\neg x \vee \neg y, \infty), (x \vee \neg z, \infty), (y \vee \neg z, \infty)\}$. As we can observe, the number of iterations depends on which unsatisfiable core we use in each step (even if these cores are minimal).[4]

$$
\begin{array}{r}
(x, 1) \\
(y, 2) \\
(z, 3) \\
(\neg x \vee \neg y, \infty) \\
(x \vee \neg z, \infty) \\
(y \vee \neg z, \infty)
\end{array}
\qquad
\begin{array}{r}
(\square, 3) \\
(x, 1) \\
(y, 2) \\
(z \vee b_3^1, 3) \\
(\neg x \vee \neg y, \infty) \\
(x \vee \neg z, \infty) \\
(y \vee \neg z, \infty) \\
(b_3^1 = 1, \infty)
\end{array}
\qquad
\begin{array}{r}
(\square, 4) \\
(x \vee b_1^2, 1) \\
(y, 1) \\
(y \vee b_2^2, 1) \\
(z \vee b_3^1, 3) \\
(\neg x \vee \neg y, \infty) \\
(x \vee \neg z, \infty) \\
(y \vee \neg z, \infty) \\
(b_3^1 = 1, \infty) \\
(b_1^2 + b_2^2 = 1, \infty)
\end{array}
$$

A longer possible execution, where all clauses get weight one or infinite, would be the following one.

$$
\begin{array}{r}
(x, 1) \\
(y, 2) \\
(z, 3) \\
(\neg x \vee \neg y, \infty) \\
(x \vee \neg z, \infty) \\
(y \vee \neg z, \infty)
\end{array}
\qquad
\begin{array}{r}
(\square, 1) \\
(x \vee b_1^1, 1) \\
(y, 2) \\
(z, 2) \\
(z \vee b_3^1, 1) \\
(\neg x \vee \neg y, \infty) \\
(x \vee \neg z, \infty) \\
(y \vee \neg z, \infty) \\
(b_1^1 + b_3^1 = 1, \infty)
\end{array}
\qquad
\begin{array}{r}
(\square, 3) \\
(x \vee b_1^1, 1) \\
(y \vee b_2^2, 2) \\
(z \vee b_3^2, 2) \\
(z \vee b_3^1, 1) \\
(\neg x \vee \neg y, \infty) \\
(x \vee \neg z, \infty) \\
(y \vee \neg z, \infty) \\
(b_1^1 + b_3^1 = 1, \infty) \\
(b_2^2 + b_3^2 = 1, \infty)
\end{array}
\qquad
\begin{array}{r}
(\square, 4) \\
(x \vee b_1^1 \vee b_1^3, 1) \\
(y \vee b_2^2, 1) \\
(y \vee b_2^2 \vee b_2^3, 1) \\
(z \vee b_3^2, 1) \\
(z \vee b_3^2 \vee b_3^3, 1) \\
(z \vee b_3^1, 1) \\
(\neg x \vee \neg y, \infty) \\
(x \vee \neg z, \infty) \\
(y \vee \neg z, \infty) \\
(b_1^1 + b_3^1 = 1, \infty) \\
(b_2^2 + b_3^2 = 1, \infty) \\
(b_1^3 + b_2^3 + b_3^3 = 1, \infty)
\end{array}
$$

In the next section we describe a heuristic that leads to shorter executions by using preferably clauses with higher weight in the cores.

8. A stratified approach for WPM1

Next we will propose an improvement to the algorithm WPM1. But before doing that, we will analyze some examples that study the limitations of WPM1. The WPM1 algorithm consists of iteratively calling a SAT solver on the instance without the weights, and once an unsatisfiable core is found, we find the minimum weight of the clauses in the core. At this point we double the clauses in the core, once with the minimum weight and an extra variable, and once with the remaining 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 another example that reflects this situation.

Example 23. Consider the formula

$$\varphi = \{(x_1, 1), (x_2, m), (\neg x_2, \infty)\}$$

[4] This example comes from the slides of the conference presentation of [4] where some weaknesses of Algorithm 2 were discussed.

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** $(\infty, \emptyset)$
$\text{cost} := 0$ ▷Optimal
$s := 0$ ▷Counter of cores
$w_{max} := \max\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < \infty\}$
while *true* **do**
  $(st, \varphi_c) := SAT(\{C_i \mid (C_i, w_i) \in \varphi \wedge w_i \geqslant w_{max}\})$ ▷Call without weights
  if $st = \text{SAT}$ **then**
    if $w_{max} = 0$ **then return** (cost, φ)
    else $w_{max} := \max\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}$
  else
    $s := s + 1$
    $A_s := \emptyset$ ▷Indexes 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**
        $b_i^s := \text{new\_variable}()$
        $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min}),\ (C_i \vee b_i^s, w_{min})\}$ ▷Duplicate soft clauses
        $A_s := A_s \cup \{i\}$
    $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{i \in A_s} b_i^s = 1)\}$ ▷Add cardinality constraint as hard clauses
    $\text{cost} := \text{cost} + w_{min}$

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:

$$\varphi_1 = \left\{\left(x_1 \vee b_1^1, 1\right), \left(x_2 \vee b_2^1, 1\right), (x_2, m-1), (\neg x_2, \infty), \left(b_1^1 + b_2^1 = 1, \infty\right)\right\}$$

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

$$\varphi_i = \left\{\left(x_1 \vee b_1^1 \vee \cdots \vee b_1^i, 1\right), \left(x_2 \vee b_2^1, 1\right), \ldots, \left(x_2 \vee b_2^i, 1\right), (x_2, m-i), (\neg x_2, \infty),\right.$$
$$\left.\left(b_1^1 + b_2^1 = 1, \infty\right), \ldots, \left(b_1^i + b_2^i = 1, \infty\right)\right\}$$

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 the previous example. However, unless the SAT solver can guarantee that it is minimal, a similar example (but more complicated) could be constructed. Moreover, Example 22 shows that, even if the SAT solver returns minimal cores, the number of iterations of WPM1 may be different depending on the cores obtained by the SAT solver.

In Algorithm 3 we present a modification of the WPM1 algorithm that tries to prevent the situations described in Examples 22 and 23, 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 than or equal to w_{max}. As in Algorithm 2, we start by checking that hard clauses are satisfiable. Then, we initialize w_{max} to the 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.

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 in Algorithm 3 than in 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. There are several heuristics that can be applied to decrease w_{max} faster than Algorithm 3 when the clauses have a wide variety of distinct weights. One of them is the *diversity heuristic* [2] where w_{max} is decreased until the following condition is satisfied

$$\frac{|\{C_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}|}{|\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}|} > \alpha \tag{1}$$

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

```
   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 = ∑{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
```

or $w_{max} = 0$, for some value of α. 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 with $\alpha = 1.25$. This value gives us good results experimentally.

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 which cores the SAT solver finds first.

9. A 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 which 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 $\varphi_{w_{max}}$ with clauses with weight w_{max} or higher, the generic WPM returns a triplet $(\text{cost}, \varphi_{sat}, \varphi_{res})$ such that φ is MaxSAT reducible to $\{(\square, \text{cost})\} \cup \varphi_{sat} \cup \varphi_{res}$, φ_{sat} is satisfiable (has cost zero), and clauses of φ_{res} have cost strictly smaller than w_{max}. Given φ, WPM1 returns a pair (cost, φ') where φ is MaxSAT reducible to $\{(\square, \text{cost})\} \cup \varphi'$ and φ' is satisfiable. Hence, WPM1 is an instance of the generic WPM where $\varphi_{res} = \emptyset$. 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 (Algorithm 4) where WPM is substituted by a partial execution of WPM1. In addition, 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 its correctness. They are inspired in [27]. The idea is to harden all soft clauses in φ_{sat} whose satisfiability does not need to be reconsidered, because falsifying them cannot be compensated by satisfying the rest of soft clauses. The proof of the correctness of these lines is based on the following lemma.

Lemma 24. *Let $\varphi_1 = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ be a MaxSAT formula with cost zero, let $\varphi_2 = \{(C'_1, w'_1), \ldots, (C'_r, w'_r)\}$ be a MaxSAT formula without hard clauses and $W = \sum_{j=1}^{r} w'_j$. Let*

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

and $\varphi'_1 = \{(C_i, \text{harden}(w_i)) \mid (C_i, w_i) \in \varphi_1\}$. Then, $\text{cost}(\varphi_1 \cup \varphi_2) = \text{cost}(\varphi'_1 \cup \varphi_2)$, and any optimal assignment for $\varphi'_1 \cup \varphi_2$ is an optimal assignment of $\varphi_1 \cup \varphi_2$.

Proof. Since $\varphi'_1 \cup \varphi_2$ have the same clauses as $\varphi_1 \cup \varphi_2$ with equal or higher weight, we trivially have $\text{cost}(\varphi_1 \cup \varphi_2) \leqslant \text{cost}(\varphi'_1 \cup \varphi_2)$. For the opposite inequality, let I be an optimal assignment of $\varphi_1 \cup \varphi_2$. Since φ_1 is satisfiable, and $W = \sum\{w \mid (C, w) \in \varphi_2\}$, we have that the optimal cost of $\varphi_1 \cup \varphi_2$ satisfies $\text{cost}(I(\varphi_1 \cup \varphi_2)) \leqslant W$. Therefore, I must satisfy all clauses $(C, w) \in \varphi_1$ where $w > W$, i.e. all hard clauses of φ'_1. Clauses of $(C, w) \in \varphi_1$ where $w \leqslant W$ have the same weight in φ_1 and in φ'_1. Therefore, $\text{cost}(I(\varphi_1 \cup \varphi_2)) = \text{cost}(I(\varphi'_1 \cup \varphi_2))$, and, by optimality of I, $\text{cost}(\varphi_1 \cup \varphi_2) \geqslant \text{cost}(\varphi'_1 \cup \varphi_2)$. □

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

Example 25. Consider $\varphi = \{(x_2, 2), (\neg x_1, 1000), (x_1, 1001), (x_1 \vee \neg x_2, \infty)\}$. This formula does not satisfy the conditions of Lemma 24. In the first iteration, we pass the last clause $\{(x_1 \vee \neg x_2, \infty)\}$ to the generic MaxSAT solver WPM, and it returns cost zero, and the same formula. In the second iteration we pass the last two clauses $\{(x_1, 1001), (x_1 \vee \neg x_2, \infty)\}$ with the same result (cost zero and the same formula).

In the third iteration we pass the three last clauses $\varphi_{w_{max}} = \{(\neg x_1, 1000), (x_1, 1001), (x_1 \vee \neg x_2, \infty)\}$ and, assuming we use WPM1, it returns cost $= 1000$ and

$$\varphi_{sat} = \left\{(x_1, 1), \left(\neg x_1 \vee b_1^1, 1000\right), \left(x_1 \vee b_2^1, 1000\right), (x_1 \vee \neg x_2, \infty), \left(b_1^1 + b_2^1 = 1, \infty\right)\right\}$$

We cannot consider all these clauses as hard clauses, in particular, we cannot force the clause $(x_1, 1)$ to be true. However, the new formula after this third iteration satisfies the conditions of Lemma 24, being $W = 2$, and φ_{sat} can be replaced by:

$$\varphi'_{sat} = \left\{(x_1, 1), \left(\neg x_1 \vee b_1^1, \infty\right), \left(x_1 \vee b_2^1, \infty\right), (x_1 \vee \neg x_2, \infty), \left(b_1^1 + b_2^1 = 1, \infty\right)\right\}$$

preserving the cost.

Notice that, in Algorithm 4, the condition $w > \sum_{i=i}^{k} w_i$ has to be checked *dynamically* (recall that the clause $(x_1, 1)$ in Example 25 was not present in the original formula). Notice also that, in Lemma 24, in general, the formula $\varphi_1 \cup \varphi_2$ is not MaxSAT reducible to $\varphi'_1 \cup \varphi_2$, and the transformation only preserves the cost.

Theorem 26. *Given a formula φ, if WPM returns a triplet $(\text{cost}, \varphi_{sat}, \varphi_{res})$ such that φ is MaxSAT reducible to $\{(\square, \text{cost})\} \cup \varphi_{sat} \cup \varphi_{res}$, φ_{sat} is satisfiable and φ_{res} only contains clauses with weight strictly smaller than w_{max}, then Algorithm 4 is a correct algorithm for Weighted Partial MaxSAT.*

Moreover, when for a formula φ, Algorithm 4 returns (c, φ'), then $c = \text{cost}(\varphi)$ and any assignment satisfying φ' is an optimal assignment of φ.

Proof. Termination of the algorithm is insured by decreasing of w_{max}.

Consider first the algorithm without optional lines 8 and 9. In every iteration, $\varphi_{w_{max}}$ is replaced by $\varphi_{sat} \cup \varphi_{res}$ (line 10) and cost by cost + cost′ (line 6), where $\varphi_{w_{max}}$ is MaxSAT reducible to $\{(\square, \text{cost}')\} \cup \varphi_{sat} \cup \varphi_{res}$. Therefore, by Lemma 8, and assuming that the side conditions of this lemma hold, we have as invariant that the input formula is MaxSAT reducible to $\{(\square, \text{cost})\} \cup \varphi$. Now, the termination condition ensures that, either $\text{cost}' = \infty$ and the cost of the input formula is ∞, or $w_{max} = 0$. In the second case, $\varphi_{w_{max}} = \varphi$ and $\varphi_{res} = \emptyset$. Therefore, the return pair $(\text{cost}, \varphi_{sat})$ satisfies that the input formula is MaxSAT reducible to $\{(\square, \text{cost})\} \cup \varphi_{sat}$, and φ_{sat} satisfiable.

Consider now lines 8 and 9. These lines have as effect hardening some clauses of φ. By Lemma 24, this transformation preserves the cost of φ. This is weaker than MaxSAT reducibility, but since cost-preserving is a transitive property, it is enough to prove the correctness of the algorithm. Notice that with these optional lines, the return pair $\{(\square, \text{cost})\} \cup \varphi_{sat}$ satisfies $\text{cost}(\varphi) = \text{cost}$ and any satisfying assignment for φ_{sat} is an optimal assignment for φ. However, the input formula is not necessarily MaxSAT reducible to $\{(\square, \text{cost})\} \cup \varphi_{sat}$. □

10. The PM2 algorithm

The next algorithm, that we call PM2, is a variant of the algorithm described in [4]. 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 unique fresh auxiliary blocking variable b_i. This variable is different for every soft clause. The presence of hard clauses in an unsatisfiable core is irrelevant. In fact, they are removed from the core by the algorithm. Therefore, like in previous sections, when we say a *core* we mean a subset of indexes of soft clauses. Covers are also subsets of indexes of blocking variables, defined as follows.

Definition 27. Given a set of cores L, 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 $\text{SC}(L)$.

Lemma 28. *$\text{SC}(L)$ 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 $\text{SC}(L)$.

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

Algorithm 5: The pseudo-code of the PM2 algorithm.

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

$BV := \{b_1, \ldots, b_m\}$ ▷Set of all blocking variables

$\varphi_w := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$ ▷Protect all soft clauses

$\text{cost} := 0$ ▷Optimal

$L := \emptyset$ ▷Set of Cores

$AL := \emptyset$ ▷Set of at-least constraints

while *true* **do**

  $AM := \emptyset$ ▷Set of at-most constraints

  foreach $B \in \text{SC}(L)$ **do**

    $k := |\{A \in L \mid A \subseteq B\}|$ ▷Num. of cores contained in the cover B

    $AM := AM \cup \{\sum_{i \in B} b_i \leqslant k\}$ ▷Add new at-most cardinality constraint

  $(st, \varphi_c) := SAT(\varphi_w \cup CNF(AL \cup AM))$ ▷Call the SAT solver, return core if UNSAT

  if $st = \text{SAT}$ **then return** $(\text{cost}, \varphi_w \cup CNF(AL \cup AM))$

  $A = \{i \in \{1, \ldots, m\} \mid C_i \vee b_i \in \varphi_c \wedge b_i \in BV\}$ ▷Indexes of soft clauses of the core

  $L := L \cup \{A\}$

  $k := |\{A' \in L \mid A' \subseteq A\}|$ ▷Number of cores contained in A including A

  $AL := AL \cup \{\sum_{i \in A} b_i \geqslant k\}$ ▷Add new at-least cardinality constraint

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

PM2 works as follows (see Algorithm 5): every soft clause C_i is extended with a blocking variable b_i.[5] 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.[6] 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 UNSAT, it also gives an unsatisfiable core φ_c. Since the hard clauses are satisfiable, the core must contain some soft clause. 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 cores such that the soft clauses are included in the new core A. 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 A (counting itself).

PM2 simplifies Fu and 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 [22], msu1.2 [29], msu3 [30] and msu4.0 [31]. 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 solver says unsatisfiable. In msu3 [30], in a first phase the authors compute a maximal set of disjoint cores, and in a second phase the authors 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 [31], 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 resulted 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 and 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 the Fu and Malik algorithm is correct for every possible run. The proof is by induction on the number of execution steps. Suppose that Fu and 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 and Malik algorithm (see Lemma 30); and

[5] In fact, for efficiency, in our implementation blocking variables b_i's are not introduced in clauses until they appear in some core.

[6] 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.

2. If PM2 does not find any core, and stops, then Fu and Malik does not find any core either (see Lemma 31), and also stops returning the same MaxSAT value, since both have run the same number of steps.

Theorem 32 will be proved by induction. We will assume by induction hypothesis that PM2 and Fu and Malik, after s execution steps, have both found the sequence of cores $A_1, \ldots, A_s$. The next lemma describes the SAT formula passed to the SAT solver in the next iteration.

Lemma 29. *After finding the sequence of cores $A_1, \ldots, A_s$, in the $(s+1)$th iteration, PM2 and Fu and Malik algorithms pass to the SAT solver the following formula, respectively.*

$$\begin{aligned}
\varphi_s &= \{C_1 \vee a_1, \ldots, C_m \vee a_m, C_{m+1}, \ldots, C_{m+m'}\} \\
&\quad \cup \bigcup_{A_r \in \{A_1,\ldots,A_s\}} CNF\Big(\sum_{i\in A} a_i \geqslant \big|\{A_j \mid j \leqslant r \wedge A_j \subseteq A_r\}\big|\Big) \\
&\quad \cup \bigcup_{B \in SC\{A_1,\ldots,A_s\}} CNF\Big(\sum_{i\in B} a_i \leqslant \big|\{A_j \mid j \leqslant s \wedge A_j \subseteq B\}\big|\Big) \\
\hat{\varphi}_s &= \Big\{C_1 \vee \bigvee_{1\in A_j} b_1^j, \ldots, C_m \vee \bigvee_{m\in A_j} b_m^j, C_{m+1}, \ldots, C_{m+m}\Big\} \\
&\quad \cup \bigcup_{j=1}^{s} CNF\Big(\sum_{i\in A_j} b_i^j = 1\Big)
\end{aligned}$$

where, if $i \in A_j$, then b_i^j is the variable added by the Fu and Malik algorithm to clause i, at iteration j.

Lemma 30. *Let φ_s and $\hat{\varphi}_s$ be the formulas described in Lemma 29. If A is a core of φ_s, then A is also a core of $\hat{\varphi}_s$.*

Proof. If A is not a core of $\hat{\varphi}_s$, then there exists an optimal interpretation $\hat{I}$ of $\hat{\varphi}_s$ that satisfies all hard clauses, all cardinality constraints of $\hat{\varphi}_s$, and all soft clauses $C_i \vee \bigvee_{i\in A_j} b_i^j$ where $i \in A$. Let $\hat{I}$ be this interpretation. Define the following interpretation for the variables of φ_s:

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

We will prove the following facts:

1. Since $\hat{I}$ satisfies the hard clause $C_{m+i} \in \hat{\varphi}_s$, for $i = 1, \ldots, m'$, then I satisfies the hard clause $C_{m+i} \in \varphi_s$. This is trivial because I and $\hat{I}$ assign the same values to the original variables that both formulas have in common.
2. Since $\hat{I}$ satisfies the cardinality constraints of $\hat{\varphi}_s$, then I satisfies the cardinality constraints of φ_s.
   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 $i = 1, \ldots, m$, we have $\sum\{\hat{I}(b_i^j) \mid j = 1, \ldots, s \wedge i \in A_j\} \leqslant 1$. Therefore, the following maximal value and the sum coincides and we have

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

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

$$\sum_{i\in A_k} I(a_i) = \sum_{i\in A_k} \sum_{\substack{i\in A_j \\ j=1,\ldots s}} \hat{I}\big(b_i^j\big) \geqslant \sum_{\substack{A_j\subset A_k \\ j\leqslant k}} \sum_{i\in A_j} \hat{I}\big(b_i^j\big) = \sum_{\substack{A_j\subset A_k \\ j\leqslant k}} 1 = \big|\{A_j \mid j \leqslant k \wedge A_j \subseteq A_k\}\big|$$

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

$$CNF\Big(\sum_{i\in A_k} a_i \geqslant \big|\{A_j \mid j \leqslant k \wedge A_j \subseteq A_k\}\big|\Big)$$

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

$$\sum_{i \in B} I(a_i) = \sum_{i \in B} \sum_{\substack{i \in A_j \\ j=1,\ldots s}} \hat{I}(b_i^j) = \sum_{i \in B} \sum_{\substack{A_j \subseteq B \\ j=1,\ldots s}} \hat{I}(b_i^j) = \sum_{\substack{A_j \subseteq B \\ j=1,\ldots s}} \sum_{i \in A_j} \hat{I}(b_i^j) = \sum_{\substack{A_j \subseteq B \\ j=1,\ldots s}} 1$$
$$= \left|\{A_j \mid j = 1, \ldots, s \wedge A_j \subset B\}\right|$$

because, by definition of cover, $i \in B$ and $i \in A_j$ implies $A_j \subseteq B$.
Hence, for any cover $B \in \mathrm{SC}\{A_1, \ldots, A_s\}$, I satisfies the at-most cardinality constraints

$$CNF\left(\sum_{i \in B} a_i \geqslant \left|\{A_j \mid j \leqslant s \wedge A_j \subseteq B\}\right|\right)$$

3. For any $i \in A$, since $\hat{I}$ satisfies the soft clause $C_i \vee \bigvee_{i \in A_j} b_i^j \in \hat{\varphi}_s$, then I satisfies the soft clause $C_i \vee a_i \in \varphi_s$.
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 $C_i \vee a_i \in \varphi_s$.

Therefore, I satisfy all hard clauses, all cardinality constraints of φ_s, and all soft clauses $C_i \vee a_i$ where $i \in A$. This contradicts that A is a core of φ_s. □

The previous lemma ensures that if PM2 finds a core, then Fu and Malik also finds a core. Hence, if PM2 does not stop, then Fu and Malik does not stop, either. Therefore, the values computed by PM2 are smaller than the values calculated by Fu and 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 and Malik algorithm.

Lemma 31. *Let φ_s and $\hat{\varphi}_s$ be the formulas described in Lemma 29. If φ_s is satisfiable, then $\hat{\varphi}_s$ is satisfiable.*

Proof. Let I be an interpretation satisfying φ_s. In particular, I satisfies $CNF(\sum_{i=1}^{m} a_i \leqslant s)$, and all hard and soft clauses. Therefore, I satisfies all the *original* soft clauses C_i where $I(a_i) = \mathit{false}$, and there are at least $m - s$ of such clauses. We have $\mathrm{cost}(\{(C_1, 1), \ldots, (C_m, 1), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}) \leqslant s$. Since Fu and 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 now after these s steps, hence $\hat{\varphi}_s$ is satisfiable. □

Theorem 32. *PM2 (see Algorithm 5) is a correct algorithm for Partial MaxSAT.*

Proof. As we have said, the proof of correctness relies on the correctness of the Fu and Malik algorithm. We prove that the Fu and Malik algorithm may simulate the PM2 algorithm. Let $\varphi = \{(C_1, 1), \ldots, (C_m, 1), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ be the original partial MaxSAT formula.

In the first iteration, PM2 and Fu and Malik call the SAT solver with the parameter

$$\varphi_0 = \{C_1 \vee a_1, \ldots, C_m \vee a_m, C_{m+1}, \ldots, C_{m+m}\} \cup \bigcup_{i=1}^{m} CNF(a_i \leqslant 0)$$
$$\hat{\varphi}_0 = \{C_1, \ldots, C_m, C_{m+1}, \ldots, C_{m+m}\}$$

respectively. Both formulas are trivially equivalent. Therefore, if the first one is satisfiable and PM2 returns cost zero, then the second one is also satisfiable, and Fu and Malik also returns cost zero.

Inductively, assume that Fu and Malik has simulated PM2 during s iterations, finding the same sequence of cores $A_1, \ldots, A_s$. Lemma 29 describes the formula passed to the SAT solver in the next $s + 1$ iteration. There are two possibilities, (1) either the SAT solver finds a core for φ_s. In such case, Lemma 30 ensures that Fu and Malik could find the same core. Or, (2) the SAT solver says that φ_s is satisfiable. In such case, Lemma 31 ensures that the SAT solver will also say satisfiable for $\hat{\varphi}_s$ and both algorithms will stop returning cost equal to s. □

11. The WPM2 algorithm

The WPM2 algorithm computes the optimal of a weighted partial MaxSAT formula. Like the name suggests, it is a generalization of PM2 to weighted MaxSAT formulas. However, this is not completely true. As we discuss below, WPM2 can

Algorithm 6: The pseudo-code of the WPM2 algorithm. The code of function newbound is in a forthcoming subsection.

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** $(\infty, \emptyset)$ ▷Hard clauses are unsatisfiable
$BV := \{b_1, \ldots, b_m\}$ ▷Set of blocking variables
$\varphi_w := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$ ▷Protect all soft clauses
$L := \emptyset$ ▷Set of Cores
$AL := \{w_1 b_1 \geqslant 0, \ldots, w_m b_m \geqslant 0\}$ ▷Set of at-least constraints
while *true* **do**
  $AM := \emptyset$ ▷Set of at-most constraints
  $\text{cost} := 0$
  foreach $(\sum_{i \in B} w_i b_i \geqslant k) \in AL$ **do**
    if $B \in \text{SC}(L)$ **then**
      $\text{cost} := \text{cost} + k$
      $AM := AM \cup \{\sum_{i \in B} w_i b_i \leqslant k\}$ ▷Add new at-most cardinality constraint
  $(st, \varphi_c) := SAT(\varphi_w \cup CNF(AL \cup AM))$ ▷Call the SAT solver
  if $st = \text{SAT}$ **then return** $(\text{cost}, \varphi_w \cup CNF(AL \cup AM))$
  $A := \{i \in \{1, \ldots, m\} \mid (C_i \vee b_i) \in \varphi_c \wedge b_i \in BV\}$ ▷Indexes of soft clauses of the core
  $A := \bigcup_{\substack{A' \in L \\ A' \cap A \neq \emptyset}} A'$ ▷Extend the new core ensuring $A \in \text{SC}(L)$
  $L := L \cup \{A\}$
  $k := \text{newbound}(AL, A)$ ▷New bound
  $AL := AL \cup \{\sum_{i \in A} w_i b_i \geqslant k\}$ ▷Add new at-least cardinality constraint

deduce weaker at-least constraints, which means that, if we use WPM2 in an unweighted formula, we will obtain poorer results than using PM2.

Like in PM2, we extend every soft clause C_i with a unique fresh auxiliary blocking variable b_i, and work with a set AL of at-least linear pseudo-boolean constraints on the variables b_i, and a similar set AM of at-most constraints, that are modified at every iteration of the algorithm. Therefore, the formula sent to the SAT solver has the form $\varphi_w \cup CNF(AL \cup AM)$, where $\varphi_w = \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$, and the linear constraints have been encoded as CNF formulas. In order to understand the purpose of AL and AM, we introduce the notion of partial solution.

Definition 33 *(Partial solution and cost).* An assignment $I : \{b_1, \ldots, b_m\} \to \{0, 1\}$ is called a *partial solution* of a weighted partial MaxSAT formula $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$, if the formula $I(\varphi_w) = \{C_i \mid I(b_i) = 0\} \cup \{C_{m+1}, \ldots, C_{m+m'}\}$ is satisfiable.

The *cost of a partial solution* I of φ is $\text{cost}(I) = \sum_{i=1}^{m} w_i I(b_i)$.

The at-least constraints AL exclude assignments to b_i's that are not partial solutions. The at-most constraints AM enforce that all solutions of the set of constraints $AL \cup AM$ are the solutions of AL of minimal cost, if AL has any solution. This ensures that any solution of $\varphi_w \cup CNF(AL \cup AM)$, if there is any, is an optimal assignment of φ.

The pseudo-code of WPM2 is described in Algorithm 6. The algorithm starts with $AL = \{w_1 b_1 \geqslant 0, \ldots, w_m b_m \geqslant 0\}$ and the corresponding $AM = \{w_1 b_1 \leqslant 0, \ldots, w_m b_m \leqslant 0\}$ that ensures that the unique solution of $AL \cup AM$ is $b_1 = \cdots = b_m = 0$ with cost 0.[7] At every iteration, the algorithm calls a SAT solver with $\varphi_w \cup CNF(AL \cup AM)$. If it returns SAT, then the assignment is a Max-SAT solution of φ. If it returns UNSAT, then we use the information of the unsatisfiable core obtained by the SAT solver to enlarge the set AL, excluding more interpretations on the b_i's that are not partial solutions. We update AM conveniently, to ensure that solutions to the new constraints $AL \cup AM$ are still minimal solutions of the new AL constraint set. Notice that AM depends on AL, and it is modified accordingly when AL is extended. Moreover, in every iteration, the set of solutions of $\{b_1, \ldots, b_m\}$ defined by AL is decreased, whereas the set of solutions of AM is increased. The constraints of AL are of the form $\sum_{i \in B} w_i b_i \geqslant k$. Therefore, they are characterized by a set of indexes $B \subseteq \{1, \ldots, m\}$, called its *base*, and a value $k \in \mathbb{N}$, called its *bound*. Similarly, the inequalities of AM are of the form $\sum_{i \in B} w_i b_i \leqslant k$. The bases of the at-most constraints AM are the set of covers SC. Moreover, the cost K is equal to the sum of the bounds of the at-most constraints, $K = \sum\{k \mid (\sum_{i \in B} w_i b_i \leqslant k) \in AM\}$.

If we compare PM2 (Algorithm 5) and WPM2 (Algorithm 6), we see that both algorithms are line-by-line equal, except in two parts. First, instead of computing the bound of at-least constraints as $k = |\{A' \in L \mid A' \subseteq A\}|$, we computed it as $k = \text{newbound}(AL, A)$. The bound of at-most constraints is also computed accordingly, ensuring that they exclude solutions of at-least constraints that are not minimal. Although the cost is computed differently, it is not a real difference because in both cases it is equal to the sum of the bounds of the at-most constraints. Second, in the case of WPM2, we add a re-computation

[7] In fact, for efficiency, in our implementation blocking variables b_i's are not introduced in clauses until they appear in some core and are incorporated into a non-trivial at-least constraint.

(extension) of the core (see line 16 in Algorithm 6) as $A := \bigcup_{\substack{A' \in L \\ A' \cap A \neq \emptyset}} A'$. This ensures that the new core belongs to the set of covers of the old cores. If we compute the new bound as newbound$(AL, A) = |\{(\sum_{i \in A'} w_i b_i \geqslant k) \in AL \mid A' \subseteq A\}| + 1$ then WPM2 computes the same bounds for unweighted instances as PM2. Therefore, the first fact does not suppose a weakness of WPM2 w.r.t. PM2, and WPM2 can be considered as a generalization of PM2. However, the second fact makes WPM2 a new algorithm, and in fact, makes WPM2 weaker than PM2 when dealing with unweighted formulas. This is because in WPM2 we have to extend the cores, obtaining a weaker at-least constraints.

The key point in WPM2 is how to calculate the value newbound(AL, B). Later, we will describe how to compute this new bound in practice. In the following example we show the execution of our algorithm and describe some problems that arise when trying to compute the new bound.

Example 34. Suppose that we have the following MaxSAT formula $\varphi = \{(C_1, 10), (C_2, 4), (C_3, 8), (C_4, 2)\}$. The extended formula is $\varphi_w = \{C_1 \vee b_1, C_2 \vee b_2, C_3 \vee b_3, C_4 \vee b_4\}$. Suppose that in the first two iterations we get the cores $A_1 = \{1, 2\}$ and later $A_2 = \{3, 4\}$. The constraints will be $AL = \{10\,b_1 + 4\,b_2 \geqslant 4,\ 8\,b_3 + 2\,b_4 \geqslant 2\}$ and $AM = \{10\,b_1 + 4\,b_2 \leqslant 4,\ 8\,b_3 + 2\,b_4 \leqslant 2\}$. And the set of covers will be $SC = \{\{1, 2\}, \{3, 4\}\}$.

Now, suppose that, in the third iteration, the SAT solver gives us the core $A_3 = \{2, 3\}$. Notice that, even though b_2 must be true, this core A_3 may be obtained because we assume that the SAT solver does not have to return minimal cores. A_3 intersects with the two old covers, therefore we will have a unique new cover $B = \{1, 2, 3, 4\}$. Notice that from AL, and the existence of a new core, we can infer $10\,b_1 + 4\,b_2 + 8\,b_3 + 2\,b_4 > 6$, where 6 is the sum of the two bounds in the removed at-most constraints. However, AL has no model I satisfying $I(10\,b_1 + 4\,b_2 + 8\,b_3 + 2\,b_4) = 7$. Therefore, we can improve the inequality by finding the smallest value $k > 6$ such that there exist a model I of the new $AL \cup \{10\,b_1 + 4\,b_2 + 8\,b_3 + 2\,b_4 \geqslant k\}$. This value is $k = 12$, which we obtain setting $I(b_1) = I(b_4) = 0$ and $I(b_2) = I(b_3) = 1$.

11.1. Correctness of the WPM2 algorithm

Next we state some basic properties of covers.

Recall that, since SC(L) is a partition of $\{1, \ldots, m\}$, for any $i = 1, \ldots, m$, there exists one and only one set in SC(L) containing i, and for any $A \in L$, there exists one and only one set in SC(L) containing A.

Lemma 35. *For any two sets of cores L_1 and L_2, if $L_1 \subseteq L_2$ then any cover $B \in \mathrm{SC}(L_2)$ satisfies $B = \bigcup_{\substack{A \in \mathrm{SC}(L_1) \\ A \subseteq B}} A$.*

Proof. Notice that, if $L_1 \subseteq L_2$, then for any $A \in \mathrm{SC}(L_1)$ there exists one and only one set $B \in \mathrm{SC}(L_2)$ such that $A \subseteq B$. Therefore, for any $A \in \mathrm{SC}(L_1)$ and any $B \in \mathrm{SC}(L_2)$, if $B \cap A \neq \emptyset$ then $A \subseteq B$, and the lemma follows. □

Definition 36 *(Bound).* For any set of at-least constraints AL, and set $B \subseteq \{1, \ldots, m\}$, we define its bound as

$$\mathrm{bound}(AL, B) = \max\left\{k \in \mathbb{N} \;\middle|\; AL \vdash \sum_{i \in B} w_i\, b_i \geqslant k\right\}$$

where $AL \vdash C$ means that all assignments to b_i's satisfying the constraints AL also satisfy the constraint C.

For instance, for the set at-least constraints AL of Example 34, we have bound$(AL, \{1, 2, 3, 4\}) = 6$.

The following lemma basically rephrases the definition of bound in a more convenient way.

Lemma 37. *For any set of constraints AL, and set $B \subseteq \{1, \ldots, m\}$,* bound$(AL, B) = k$ *iff*

1. *the bound is probable*: $AL \vdash \sum_{i \in B} w_i\, b_i \geqslant k$, *and*
2. *feasible*: $AL \nvdash \sum_{i \in B} w_i\, b_i > k$, *i.e. exists an interpretation $I : \{b_1, \ldots, b_m\} \to \{0, 1\}$ such that $I(AL) = true$ and $I(AL \vdash \sum_{i \in B} w_i\, b_i = k) = true$.*

As we said in the previous section, the set of at-most constraints AM depends on the set AL.

Definition 38. Given a set of at-least constraints AL, the set of at-most constraints AM associated to AL is

$$AM = \left\{\sum_{i \in B} w_i\, b_i \leqslant \mathrm{bound}(AL, B) \;\middle|\; B \in \mathrm{SC}(L)\right\}$$

where $L = \{A \mid (\sum_{i \in A} w_i\, b_i \geqslant k) \in AL\}$.

Lemma 39. *For any set of at-least constraints AL and any subset of covers* $\{B_1, \ldots, B_s\} \subseteq \mathrm{SC}(L)$*, where L is the set of bases of AL, we have*

$$\mathrm{bound}\left(AL, \bigcup_{i=1}^{s} B_i\right) = \sum_{i=1}^{s} \mathrm{bound}(AL, B_i)$$

Hence, the bound of $\bigcup_{i=1}^{s} B_i$ *is the sum of the bounds of the constraints* $\sum_{j\in B_i} w_j b_j \leqslant k_i \in AM$ *associated to AL.*

Proof. The inequality $\mathrm{bound}(AL, \bigcup_{i=1}^{s} B_i) \geqslant \sum_{i=1}^{s} \mathrm{bound}(AL, B_i)$ is trivial, and holds for any set $\{B_1, \ldots, B_s\}$ of pairwise disjoint sets, even if they are not covers. However the opposite inequality is more difficult to prove.

On one hand, by Lemma 37, we have $AL \vdash \sum_{j\in B_i} w_j b_j \geqslant \mathrm{bound}(AL, B_i)$. Since B_i's are pairwise disjoint, adding these inequalities we get $AL \vdash \sum_{i=1}^{s}\sum_{j\in B_i} w_j b_j = \sum_{j\in\bigcup_{i=1}^{s} B_i} w_j b_j \geqslant \sum_{i=1}^{s} \mathrm{bound}(AL, B_i)$.

On the other hand, again by Lemma 37, for every $i = 1, \ldots, s$ we have an interpretation I_i such that $I_i(AL) = \mathit{true}$ and $I_i(\sum_{j\in B_i} w_j b_j \leqslant \mathrm{bound}(AL, B_i)) = \mathit{true}$. Now, construct a new interpretation I defined as $I(x) = I_i(x)$, if $x \in B_i$; and $I(x) = I_1(x)$, if $x \notin \bigcup_{i=1}^{s} B_i$. Notice that all constraints of AL have all their variables inside one of the sets B_i. Therefore, $I(AL) = \mathit{true}$. Notice also that $I(\sum_{j\in B_i} w_j b_j) = I_i(\sum_{j\in B_i} w_j b_j) = \mathrm{bound}(AL, B_i)$. Therefore, $I(\sum_{i=1}^{s}\sum_{j\in B_i} w_j b_j) = \sum_{i=1}^{s} \mathrm{bound}(AL, B_i)$.

These two facts, by Lemma 37, prove that statement of the lemma. □

The following lemma allows us to strengthen the set of at-least constraints. For any set of indexes B, we always have $\varphi \vdash \sum_{i\in B} w_i b_i \geqslant \mathrm{bound}(AL, B)$. The lemma states that, whenever we find a core A, we can improve this inequality for B being the union of covers intersecting with A, i.e. the cover of $\mathrm{SC}(\{A_1, \ldots, A_r, A\})$ that contains A. Therefore, we can enlarge AL getting $AL' = AL \cup \{\sum_{i\in B} w_i b_i \geqslant \mathrm{bound}(AL, B) + 1\}$. However, it could be the case that AL' had no models satisfying $\sum_{i\in B} w_i b_i = \mathrm{bound}(AL, B) + 1$ (see Example 34). In this case, we enlarge AL with $\sum_{i\in B} w_i b_i = \mathrm{newbound}(AL, B)$, where $\mathrm{newbound}(AL, B)$ is the minimum integer greater than $\mathrm{bound}(AL, B) + 1$ such that AL' has a model satisfying $\sum_{i\in B} w_i b_i = \mathrm{newbound}(AL, B)$.

Lemma 40. *Let* $\varphi^e = \{C_1 \vee b_1, \ldots, C_m \vee b_m,\ C_{m+1}, \ldots, C_{m+m'}\}$ *be an extended formula and AL be a set of at-least constraints. Let AM be the set of at-most constraints associated to AL.*

If, for some core $A \subseteq \{1, \ldots, m\}$*, we have*

1. $\{C_i \vee b_i\}_{i\in A} \cup \{C_{m+1}, \ldots, C_{m+m'}\} \cup AL \cup AM$ *is unsatisfiable*
2. $\varphi_w \vdash AL$

then

$$\varphi_w \vdash \sum_{i\in B} w_i b_i \geqslant \mathrm{bound}(AL, B) + 1$$

where $B \in \mathrm{SC}(L \cup \{A\})$*, L is the set of bases of AL, and* $A \subseteq B$*.*

Moreover,

$$\varphi_w \vdash \sum_{i\in B} w_i b_i \geqslant \mathrm{newbound}(AL, B)$$

where

$$\mathrm{newbound}(AL, B) = \mathrm{bound}\big(AL',\ B\big)$$

$$AL' = AL \cup \left\{\sum_{i\in B} w_i b_i \geqslant \mathrm{bound}(AL, B) + 1\right\}$$

Proof. Suppose that $\varphi_w \nvdash \sum_{i\in A} w_i b_i \geqslant \mathrm{bound}(AL, B) + 1$. This means that there exists an assignment I such that $I(\varphi_w) = \mathit{true}$ and $I(\sum_{i\in B} w_i b_i \geqslant \mathrm{bound}(AL, B) + 1) = \mathit{false}$. We have $I(\sum_{i\in B} w_i b_i \leqslant \mathrm{bound}(AL, B)) = \mathit{true}$. Since $\varphi_w \vdash AL$, $I(AL) = \mathit{true}$. By definition of bound, we have

$$I\left(\sum_{i\in B} w_i b_i = \mathrm{bound}(AL, B)\right) = \mathit{true} \tag{2}$$

Since $I(AL) = \mathit{true}$, by definition of bound, we have

$$I\left(\sum_{i\in B'} w_i b_i \geqslant \mathrm{bound}\big(AL, B'\big)\right) = \mathit{true} \tag{3}$$

for any $B' \in \mathrm{SC}(L)$. On the other side, since $B \in \mathrm{SC}(L \cup \{A\})$, by Lemmas 35 and 39

$$B = \bigcup_{\substack{B' \in \mathrm{SC}(L) \\ B' \subseteq B}} B'$$

$$\mathrm{bound}(AL, B) = \sum_{\substack{B' \in \mathrm{SC}(L) \\ B' \subseteq B}} \mathrm{bound}\big(AL, B'\big) \tag{4}$$

From (2), (3) and (4) we have

$$I\left(\sum_{i \in B'} w_i\, b_i \leqslant \mathrm{bound}\big(AL, B'\big)\right) = \mathit{true}$$

for any $B' \in \mathrm{SC}(L)$ satisfying $B' \subseteq B$. Hence, I satisfies all the at-most constraints of AM that make reference to the covers B' included in B. We have to modify the interpretation I to get an interpretation that also satisfies the constraints of AM that make reference to covers not included in B.

By definition of bound, there exists an interpretation I' such that $I'(AL) = \mathit{true}$ and $I'(AM) = \mathit{true}$. Let I'' be another interpretation defined as $I''(x) = I(x)$ if x is a variable of $\{C_1, \ldots, C_{m+m'}\}$ or $\{b_i\}_{i \in B}$, and $I''(x) = I'(x)$, otherwise. Notice that all clauses of $AL \cup AM \cup \{C_i \vee b_i\}_{i \in A} \cup \{C_{m+1}, \ldots, C_{m+m'}\}$ either contain variables of one subset or from the other. Therefore, I'' satisfies all these clauses. This contradicts the first assumption of the lemma. □

Example 41. Suppose that we have the situation described in Example 34. The list of cores is $L = \{\{1, 2\}, \{3, 4\}\}$, and the constraints

$$AL = \{10\,b_1 + 4\,b_2 \geqslant 4,\ 8\,b_3 + 2\,b_4 \geqslant 2\}$$

$$AM = \{10\,b_1 + 4\,b_2 \leqslant 4,\ 8\,b_3 + 2\,b_4 \leqslant 2\}$$

The third core $A = \{2, 3\}$ generates a new cover $B = \{1, 2, 3, 4\}$ that belongs to the set of covers of $\{\{1, 2\},\ \{3, 4\},\ \{2, 3\}\}$. We have $\mathrm{bound}(AL, \{1, 2, 3, 4\}) = 6$. The first part of Lemma 40 allows us to conclude that $\varphi_w \vdash 10\,b_1 + 4\,b_2 + 8\,b_3 + 2\,b_4 \geqslant 7$. After adding this constraint we get $AL' = AL \cup \{10\,b_1 + 4\,b_2 + 8\,b_3 + 2\,b_4 \geqslant 7\}$, using the second part of the lemma, we get $\mathrm{bound}(AL', \{1, 2, 3, 4\}) = \mathrm{newbound}(AL, \{1, 2, 3, 4\}) = 12$ and $\varphi_w \vdash 10\,b_1 + 4\,b_2 + 8\,b_3 + 2\,b_4 \geqslant 12$.

Theorem 42. *Given a Weighted Partial MaxSAT problem, the algorithm WPM2 computes an optimal assignment of minimal cost.*

Proof. Once the algorithm finds a satisfying assignment I for $\varphi_w \cup \mathit{CNF}(AL \cup AM)$, it returns $K = \sum\{k' \mid \sum_{i \in B'} w_i\, b_i \leqslant k' \in AM\}$. Since the bases of at-most constraints are a partition of $\{1, \ldots, m\}$ and I satisfies AM, the cost of I is bounded by K. By Lemma 39, $K = \mathrm{bound}(AL, \{1, \ldots, m\})$, and by Lemma 37, $AL \vdash \sum_{i=1}^{m} w_i\, b_i \geqslant K$. Lemma 40 ensures that the new at-least constraint added to AL in each iteration does not exclude partial solutions: $\varphi_w \vdash AL$, i.e. it only excludes values of the b_i's such that φ_w is unsatisfiable. Therefore, the cost K of I is smaller than the cost of any partial solution of φ_w, hence of the minimal cost of φ. Termination of the algorithm is ensured by the fact that the value of $\sum\{k' \mid \sum_{i \in B'} w_i\, b_i \leqslant k' \in AM\}$ is increased in every iteration, and it is bounded by the minimal cost of φ. □

11.2. Computation of the new bound

The value of $\mathrm{bound}(AL, B)$ can be computed easily using the at-most constraints as

$$\mathrm{bound}(AL, B) = \sum \left\{ k' \;\middle|\; \sum_{i \in B'} w_i\, b_i \leqslant k' \in AM \wedge B' \subseteq B \right\}$$

However, the computation of $\mathrm{newbound}(AL, B)$ is not so simple. Recall the definition:

$$\mathrm{newbound}(AL, B) = \mathrm{bound}\big(AL',\ B\big)$$
$$\text{where } AL' = AL \cup \left\{ \sum_{i \in B} w_i\, b_i \geqslant \mathrm{bound}(AL, B) + 1 \right\}$$

Given AL and B, the calculation of $\mathrm{newbound}(AL, B)$ is an NP-complete optimization problem. This can be seen by a reduction from the following version of the subset sum problem: given $\{w_1, \ldots, w_n\}$ and k, minimize $\sum_{j=1}^{n} w_j\, x_j$ subject to $\sum_{j=1}^{n} w_j\, x_j > k$ and $x_j \in \{0, 1\}$. This is equivalent to computing $\mathrm{newbound}(AL, B)$, where the weights are w_j, $B = \{1, \ldots, n\}$ and $AL = \{\sum_{j=1}^{n} w_j\, x_j \geqslant k\}$.

In our implementation we use the following function to compute newbound.

Algorithm 7: A possible implementation of the newbound function.

```
1: function newbound(AL, B)
2: k := bound(AL, B)
3: repeat
4: |  k = subsetsum({w_i | i ∈ B}, k)
5: until SAT(CNF(AL ∪ {∑_{i∈B} w_i b_i = k}))
6: return k
```

Notice that the satisfiability check of $AL \cup \{\sum_{i\in B} w_i\, b_i = k\}$ is necessary for soundness (see Example 41). Also notice that the subset sum problem, even though it is NP-complete, in practical situations can be computed very efficiently. In our implementation we use an algorithm based on dynamic programing described in [35]. As many other numerical problems, it is pseudo-polynomial.

11.3. Complexity of the algorithm

Example 23 shows that WPM1 (without the stratified heuristics) may need $\mathcal{O}(W)$ calls to a SAT solver, where $W = \sum_{i=1}^{m} w_i$ is the sum of the finite weights, even for a constant number of clauses. Therefore, the performance of these MaxSAT algorithms depends on the sequence of cores computed by the SAT solver. In the case of WPM2, the number of calls to the SAT solver is also $\mathcal{O}(W)$ in the worst case, as stated in [19] (Theorem 1). However, this is only true if we use the new bound function described in Algorithm 7. In fact, we use this implementation because we have seen that, in practice, not in the worst case, it gives good results. If we use a trivial binary search, as described in Section 3, then the number of calls to the SAT solver is $\mathcal{O}(m \log W)$, since there are m possibly distinct covers, and for each one we need at most $\log W$ calls to the SAT solver to complete the binary search. Moreover, a bit more intelligent binary search, allows us to replace $\log W$ by m.

12. Experimental results

We conducted our experimentation on the same environment as the MaxSAT evaluation [8] (processor 2 GHz). These are the specifications: operating system Rocks Cluster 4.0.0 Linux 2.6.9, processor AMD Opteron 248 Processor 2 GHz, memory 1 GB and compilers GCC 3.4.3 and javac JDK 1.5.0. The time and memory resources of our experimentation were increased with respect to the resources of the MaxSAT evaluation. We augmented the timeout from half hour to two hours, and the memory limit from 0.5 GB to 1 GB, since some solvers we compared with require more memory.

The solvers that implement our Weighted Partial MaxSAT algorithms are built on top of the SAT solver picosat (v.924) [11]. The solver *wpm1* implements the original WPM1 algorithm [4]. The cardinality constraints introduced by WPM1 are translated into SAT through the regular encoding [6]. This encoding ensures 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 6, 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 9. The solver *wpm1* was submitted to the 2009 and 2010 MaxSAT evaluations, and the solver $wmp1_s$ was submitted to the 2011 evaluation, winning the Weighted Partial MaxSAT category for industrial instances. The hardening of soft clauses (lines 8 and 9 in Algorithm 4) had no impact in our implementations' performance.

The solver *pm2* implements the original PM2 algorithm [4,3]. This solver was submitted to the 2009 evaluation, winning the Partial MaxSAT category for industrial instances. The cardinality constraints introduced by *pm2* are translated into SAT through the sequential counter encoding [38]. The solvers $pm2_{sn}$ and $pm2_{cn}$ use the sorting network and cardinality network encoding described in [10], respectively.

The solver *wpm2* implements the original WPM2 algorithm [5]. It computes newbound(AL, B) by a reduction to the subset sum problem. The pseudo-boolean linear constraints are translated into SAT through the miniSAT$^+$ tool from [14].

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 at least the best three solvers of the 2011 MaxSAT evaluation for each category. The solvers submitted to the MaxSAT 2011 evaluation we experimented with are the following: *akms* (akmaxsat), *akms_ls* (akmaxsat_ls), *incwmaxsatz*, *wmaxsatz09*, *qms0.11* and *qms0.4* (QMaxSat versions 0.11 and 0.4), *sat4j* (SAT4J MAXSAT 2.2.0), *wbo1.6* and *pwbo1.1*. From *qms0.11* to *pwbo1.1*, all solvers are SAT-based MaxSAT solvers, while the rest are branch and bound based solvers.

We also compare with other solvers which did not compete but have been reported to exhibit good performance, such as *binc* and *bincd* [19], *maxhs* [13] and the Weighted CSP solver *toulbar2* [37].

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 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. Notice that the different sets of families do no have the same number instances. Therefore, additionally, we include the mean ratio of solved instances in each family.

Table 1
Experimental results.

(a) UnWeighted – Industrial

Instance set	#	*wpm1*	*pm2*	*wpm1$_b$*	*pm2$_{cn}$*	*maxhs*	*pm2$_{sn}$*	*wbo1.6*	*binc*	*bincd*	*sat4j*	*toulbar2*
circuit-debug	3	**27.8(3)**	51.2(2)	28.62(3)	53.5(2)	62.6(3)	53.5(2)	45.99(1)	84.8(1)	79.9(1)	0(0)	0(0)
sean-safarpour	52	**848(30)**	417(26)	309(23)	366(21)	154(18)	105(15)	206(10)	292(10)	175(9)	0(0)	0(0)
Total solved	55	**33**	28	26	23	21	17	11	11	10	0	0
Mean solved ratio		**78.5**	58.0	72.0	53.5	67.0	47.5	26.0	26.0	25.0	0.0	0.0

(b) UnWeighted – Crafted

Instance set	#	*akmaxsat_ls*	*incwmaxsatz*	*wmaxsatz09*	*toulbar2*	*pm2*	*pm2$_{sn}$*	*pm2$_{cn}$*	*bincd*	*wpm1*	*wbo1.6*	*wpm1$_b$*	*binc*	*maxhs*	*sat4j*
maxcut-140-630-0.7	50	**179(50)**	431(50)	507(50)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
maxcut-140-630-0.8	50	**132(50)**	311(50)	393(50)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
dimacs-mod	62	**162(53)**	212(53)	176(52)	643(45)	207(10)	864(10)	243(9)	140(6)	0.02(4)	0.01(4)	38.4(5)	54.9(4)	24.6(4)	0.84(2)
spinglass	5	11.74(3)	28.74(3)	**5.29(3)**	2.56(2)	3.70(2)	15.8(2)	12.9(2)	2.03(1)	0.24(2)	4.07(2)	2.43(1)	2.70(1)	631(1)	52.1(1)
Total solved	167	**156**	156	155	47	12	12	11	7	6	6	6	5	5	3
Mean solved ratio		**86.25**	86.25	85.75	28.0	14.0	14.0	13.5	7.25	11.5	11.5	7.0	6.5	6.5	5.75

(c) Partial – Industrial

Instance set	#	*bincd*	*qms0.4*	*pm2$_{cn}$*	*pwbo1.1*	*pm2$_{sn}$*	*pm2*	*qms0.11*	*binc*	*sat4j*	*wpm1$_b$*	*maxhs*	*wbo1.6*	*wpm1*	*toulbar2*
aes	7	502(1)	0(0)	0(0)	0(0)	0(0)	132(1)	0(0)	0(0)	0(0)	0(0)	**952(2)**	0(0)	0(0)	0(0)
bcp-fir	59	183(55)	73.8(54)	36.7(56)	198(51)	42.2(54)	**20.2(58)**	81.7(44)	287(53)	7.79(10)	22.7(53)	575(24)	127(43)	41.3(57)	127(10)
bcp-hipp-yRa1-simp	17	128(14)	**163(16)**	86.5(15)	274(15)	104(15)	96.7(15)	208(16)	289(15)	870(14)	117(15)	640(11)	31.6(10)	24.6(9)	190.5(5)
bcp-hipp-yRa1-su	38	277(29)	**544(32)**	399(29)	537(26)	505(29)	290(29)	271(30)	406(27)	695(9)	140(18)	666(21)	58.2(10)	46(11)	0(0)
bcp-msp	64	**426(33)**	135(26)	1886(31)	404(20)	1877(25)	2072(21)	53.5(26)	520(21)	95.2(12)	143(7)	244(1)	436(4)	12.7(4)	450(19)
bcp-mtg	40	11.13(40)	**0.24(40)**	5.50(40)	1.95(40)	9.34(40)	5.51(40)	**0.24(40)**	18.7(40)	192(27)	28.2(40)	225(6)	98.8(16)	126(11)	2360(7)
bcp-syn	74	86.3(41)	372(35)	4.40(35)	136(40)	17.1(36)	10.9(38)	78.7(34)	48.1(30)	366(25)	31.8(38)	**90.6(60)**	106.3(34)	20.4(32)	433(26)
circuit-trace-compact	4	471(3)	**121(4)**	625(4)	325(3)	543(4)	574(4)	136(4)	473(3)	620(3)	1378(4)	0(0)	0(0)	3534(1)	5149(1)
haplotype-assembly	6	0(0)	2836(4)	32.8(5)	21.6(5)	33.1(5)	33.1(5)	0(0)	0(0)	0(0)	284(2)	28.7(5)	**9.02(5)**	34.2(5)	0(0)
pbo-mqcnencdr	84	861(82)	402(82)	1041(74)	781(84)	1130(73)	1193(65)	170(78)	714(80)	**497(84)**	346(25)	813(34)	226(20)	692(7)	3510(5)
pbo-mqcnlogencdr	84	311(84)	231(78)	597(84)	437(84)	836(84)	1122(84)	227(81)	288(84)	**95.9(84)**	270(44)	126(35)	113(26)	176(28)	2545(10)
pbo-routing	15	11.0(15)	19(15)	0.79(15)	1.15(15)	1.91(15)	1.11(15)	32.2(15)	20.1(15)	423(15)	0.97(15)	63.2(14)	**0.52(15)**	0.82(15)	187(6)
protein_ins	12	255(2)	578(4)	66.2(2)	0.21(1)	45.4(2)	64(2)	**1127(5)**	243(2)	1173(4)	0.69(1)	16.2(1)	13.1(1)	57(1)	0.88(1)
Total solved	504	**399**	390	390	384	382	377	373	370	287	262	214	184	181	90
Mean solved ratio		66.30	**72.92**	72.53	68.84	71.53	72.15	66.53	62.23	50.07	54.69	42.61	38.15	39.46	16.84

(d) Partial – Crafted

Instance set	#	*toulbar2*	*akms_ls*	*qms0.11*	*incwmaxsatz*	*pm2*	*pm2$_{cn}$*	*pm2$_{sn}$*	*sat4j*	*bincd*	*binc*	*wpm1$_b$*	*maxhs*	*wpm1*	*wbo1.6*
frb	25	3315(5)	247(5)	**404(23)**	377(5)	1841(19)	508(16)	689(15)	0(0)	0(0)	0(0)	0(0)	4442(5)	0(0)	0(0)
job-shop	3	0(0)	0(0)	**144(3)**	0(0)	1014(3)	1062(3)	1335(3)	581(3)	717(3)	473(3)	223(2)	0(0)	243(1)	458(1)
maxclic-random	96	84(96)	**2.41(96)**	193(80)	3.48(96)	399(65)	137(58)	289(64)	215(70)	84(63)	58.5(57)	0(0)	2367(9)	0(0)	0(0)
maxclic-structured	62	575(35)	**207(40)**	403(26)	159(36)	769(19)	299(19)	449(19)	1001(21)	138(18)	154(17)	45.3(8)	337(11)	13.3(8)	13.4(10)
maxone-3sat	80	25.9(80)	0.87(80)	391(80)	**0.43(80)**	20(80)	24.5(80)	45.5(80)	944(73)	22(80)	29.3(80)	257(73)	492(46)	3.77(42)	26.6(35)
maxone-structured	60	207(57)	708(37)	**27(60)**	273(56)	1212(51)	1246(57)	1199(49)	5.1(59)	149(58)	73(59)	286(27)	0(0)	11.3(2)	14(1)
min-enc-kbtree	42	1621(9)	**2926(22)**	1343(6)	2474(2)	479(6)	1013(5)	1735(6)	1089(3)	323(3)	107(3)	249(2)	0(0)	0(0)	1346(1)
pseudo-miplib	4	10.5(4)	466(3)	**2.89(4)**	1809(3)	14.0(4)	26.2(4)	25.5(4)	69.1(4)	253(4)	196(4)	10.2(3)	184(4)	0.44(2)	0.33(2)
Total solved	372	**286**	283	282	278	247	242	240	233	229	223	115	75	55	50
Mean solved ratio		61.5	59.125	**78.875**	56.375	71.625	70.125	69.125	62.875	62.25	61.5	36.875	25.5	19.0	18.375

Table 2
Experimental results.

(a) Weighted Partial – Industrial

Instance set	#	$wpm1_{bsd}$	$wpm1_{bs}$	$wpm1_{s}$	$wpm1_{b}$	*wbo1.6*	*wpm1*	*wpm2*	*maxhs*	*bincd*	*sat4j*	*binc*	*toulbar2*
haplotyping-pedigrees	100	**405(95)**	426(95)	378(88)	376(79)	93.5(72)	390(65)	350(42)	1043(34)	196(21)	108(20)	408(27)	383(5)
timetabling	26	**686(9)**	694(9)	585(8)	992(9)	776(4)	1002(7)	707(9)	1238(4)	766(6)	0(0)	1350(4)	0(0)
upgradeability-problem	100	35.8(100)	37.2(100)	36.5(100)	36.9(100)	63.3(100)	114(100)	311(100)	**29(100)**	637(78)	844(30)	0(0)	0(0)
Total solved	226	**204**	204	196	188	176	172	151	138	105	50	31	5
Mean solved ratio		**76.33**	76.33	72.66	71.0	62.33	63.66	58.66	49.66	40.66	16.6	14.0	1.66

(b) Weighted Partial – Crafted

set	#	$wpm1_{bsd}$	*incwmaxsatz*	*akmaxsat*	*toulbar2*	$wpm1_{bs}$	*wmaxsatz09*	*maxhs*	*sat4j*	$wpm1_{s}$	*bincd*	*binc*	*wpm1*	*wbo1.6*	*wpm2*	$wpm1_{b}$
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)	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)	3196(8)	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)	307(40)	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)	158(1)	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)	398(2)	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)	0(0)	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)	196(9)	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)	475(9)	131(7)
Total solved	372	**270**	241	226	225	224	222	204	189	184	125	108	84	82	69	67
Mean solved ratio		**66.5**	56.625	43.625	43.875	53.125	41.375	39.125	38.375	40.5	35.125	28.5	30.25	22.25	23.5	25.5

(c) Table 4 from [13]. Linux upgradibility family forcing diversity of weights

Instance set	#	*maxhs*	*wbo1.6*	$wpm1_{b}$	$wpm1_{bsd}$	*wpm1*	$wpm1_{bs}$	$wpm1_{s}$	*bincd*	*sat4j*	*wpm2*	binc	*toulbar2*
Table 4 [13]	13	**4.41(13)**	5.03(13)	5.54(13)	8.84(13)	19(13)	534(13)	559(13)	56.69(11)	3485.52(6)	32.2 (5)	19.5(1)	0.00(0)
Total solved	13	**13**	13	13	13	13	13	13	11	6	5	1	0

Table 1(a) presents the results for the industrial instances of the UnWeighted MaxSAT category. The solver *wpm1* is the best ranked one, closely followed by *pm2*. Branch and bound based MaxSAT solvers such as *incwmsz*, *akms*, *wmaxsatz09* or *toulbar2* solve at most 2 instances in this category.

Table 1(b) presents the results for the crafted instances of the UnWeighted MaxSAT category. For crafted instances we can observe the opposite behavior, branch and bound based MaxSAT solvers clearly outperform SAT-based solvers.

Although $wpm1_b$ introduces the symmetry breaking clauses, it does not perform as well as *wpm1*, for the two previous categories. A possible explanation is: since there are not hard clauses, the number of soft clauses in the unsatisfiable cores tends to be higher. Therefore, $wpm1_b$ adds many more clauses which increases memory consumption.

Table 1(c) presents the results for the industrial instances of the Partial MaxSAT category. The best solver is *bincd* (399) closely followed by *qms0.4* (390) and $pm2_{cn}$ (390). If we use as the ranking criterion the mean ratio of solved instances, we can see that now the best solver is *qms0.4*(72.92%) followed by $pm2_{cn}$(72.53%) and *bincd*(66.30%) ranks sixth. As with the industrial unweighted instances SAT-based solvers clearly outperform branch and bound based solvers.

Table 1(d) presents the results for the crafted instances of the Partial MaxSAT category. The addition of hard clauses in crafted problems seems to offer some advantage to SAT-based solvers. Although *toulbar2* and $akms_{ls}$ are the best ranked solvers, if we compute the average ratio of solved instances, we see that *qms0.11* is the best solver and *pm2* the second. Solvers *toulbar2* and $akms_{ls}$ rank seventh and ninth, using this criterion.

The addition of breaking symmetry clauses in $wpm1_b$ clearly boosts the performance of the basic solver *wpm1* for partial MaxSAT instances. In particular, for industrial instances, we improve from 181 to 262 solved instances, and for crafted from 55 to 115.

Table 2(a) presents the results for the industrial instances of the Weighted Partial MaxSAT category. As we can see, our original solver *wpm1* performs similarly to *wbo1.6*. However, 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.

In order to study the impact of the stratified approach in more detail, we compared $wpm1_s$ and *wpm1* on the instances that both were able to solve.

For the upgradibility family, *wpm1* performed in average 878 unsatisfiable calls to the SAT solver, while the average reduction of unsatisfiable calls for $wpm1_s$ was about 2.90%. We do not take into account the intermediate satisfiable calls for $wpm1_s$ since they were fast enough. The average gain in time for $wpm1_s$ was 210%.

For the haplotyping-pedigrees family, *wpm1* performed in average 58 unsatisfiable calls to a SAT solver, while the average reduction of unsatisfiable calls for $wpm1_s$ was about 5.61%. The average gain in time for $wpm1_s$ was 65%. Notice that for this family *wpm1* solves 23 instances less, so the previous percentages may be larger provided a bigger cutoff.

Overall, we can conclude that although the stratified approach produces in average less unsatisfiable calls, also these calls seem to be faster. The smaller size of the formula that we send to the SAT solver in $wpm1_s$ may help. Besides, by grouping clauses by weight we may be capturing some structural properties of the instance that help the SAT solver to come up with the proof of unsatisfiability faster.

Table 2(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. They 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 compared to the participants of the MaxSAT 2011 evaluation, and very close to *akmaxsat*. If we compare carefully the results of *wpm1* and $wpm1_{bs}$, we notice that there are two sets of instances 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 [13] 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 generated 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 the experiment with the same instances in Table 2(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, both industrial and crafted. It is interesting to highlight that for the first time a SAT-based MaxSAT solver dominates the Weighted Partial MaxSAT category for crafted instances.

We can also see that *wpm1* is the best approach for unweighted industrial instances closely followed by *pm2*.

With respect to the different implementations of algorithm PM2, we can see that it is still a quite competitive approach. From a point of view of robustness, *qms* and *pm2* dominate the Partial MaxSAT categories. The current solver of algorithm WPM2 [5], is not efficient enough for crafted instances. We would need to explore more efficient encodings for pseudo-boolean constraints into SAT and better strategies for computing the newbound(AL, B).

13. Conclusions

In this work we have presented several SAT-based MaxSAT algorithms which iteratively call a SAT solver. We have shown that these algorithms clearly dominate branch and bound based algorithms on industrial instances and on some crafted instances. Moreover, their design allows to benefit from advances in SAT solvers without almost no additional cost. Therefore, this is a relevant approach for solving optimization problems where the underlying constraints are well-suited for SAT solvers.

We have worked on efficient algorithms which iteratively refine the lower bound thanks to the discovery of unsatisfiable cores. Other search schemes which alternate the refinement of the lower and the upper bound also have great potential, and will be worth exploring in the future.

First, we have presented a more efficient version of the Fu and Malik algorithm thanks to the addition of symmetry breaking clauses on the blocking variables introduced between iterations. This is an idea which can be applied to other algorithms that solve a problem by working on a sequence of reformulations. This reformulations may artificially introduce symmetries making the problem harder, and our ideas may speed up the solutions.

Then, we have presented the algorithm WPM1, which is the weighted version of the Fu and Malik algorithm. Later, we have introduced a stratified approach which prioritizes the discovery of higher quality unsatisfiable cores. This has a significant impact on the performance. It also raises the interesting question of how to define the quality of a core. From the experimental results, we can conclude that the implementation of the WPM1 algorithm with the stratified approach is the most robust approach for weighted partial industrial and crafted instances.

On the other hand, we have presented an alternative version of the Fu and Malik algorithm, the algorithm PM2, which introduces only one blocking variable per soft clause. Although this approach needs to introduce more complicated cardinality constraints on the blocking variables, there are several encodings of cardinality constraints into CNF which guarantee arc-consistency while keeping the size of the encoding tractable in practice. Obviously, these algorithms should be parametric on the encoding of the cardinality constraints since the performance can vary from one family of instances to another. Overall, from the experimental results we can see that the implementation of the PM2 algorithm is quite robust for industrial Partial MaxSAT instances.

Finally, the WPM2 algorithm extends the PM2 algorithm to weighted formulas. Here, we impose pseudo-boolean linear constraints on the blocking variables. Unfortunately, there are not as good encodings for pseudo-boolean linear constraints as for cardinality constraints, in the sense of guaranteeing arc-consistency and a tractable size of the encoding. Therefore, a nice research avenue is to treat these pseudo-boolean constraints directly. This can be achieved by building our algorithms on top of a pseudo-boolean (PB) solver or a Satisfiability Modulo Theory (SMT) solver with the Linear Integer Arithmetic Theory, and the ability to return unsatisfiable cores. Preliminary work in this direction using SMT solvers can be found in [1].

A nice contribution of the WPM2 algorithm, with respect to the WPM1 algorithm, is to isolate the computation of the new bound (see Algorithm 7). This problem is better-suited for Integer Linear Programming (ILP) approaches than for SAT. Notice that, in order to circumvent this computation, WPM1 is forced to duplicate soft clauses increasing the size of the formula. We think that although WPM2 is not yet as competitive as WPM1, the ideas in this algorithm are promising and could be better exploited by the usage of SMT, PB or ILP approaches.

We have provided detailed proofs of correctness for all our algorithms. We think that similar algorithms or more complicated approaches could use our proof ideas as a solid basis for new correctness arguments.

References

[1] C. Ansótegui, M. Bofill, M. Palahí, J. Suy, M. Villaret, A proposal for solving weighted CSPs with SMT, in: Proc. of the 10th Int. Workshop on Constraint Modelling and Reformulation (ModRef 2011), 2011, pp. 5–19.
[2] C. Ansótegui, M.L. Bonet, J. Gabàs, J. Levy, Improving SAT-based Weighted MaxSAT solvers, in: Proc. of the 18th Int. Conf. on Principles and Practice of Constraint Programming (CP'12), 2012, pp. 86–101.
[3] C. Ansótegui, M.L. Bonet, J. Levy, On solving MaxSAT through SAT, in: Proc. of the 12th Int. Conf. of the Catalan Association for Artificial Intelligence (CCIA'09), 2009, pp. 284–292.
[4] C. Ansótegui, M.L. Bonet, J. 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, pp. 427–440.
[5] C. Ansótegui, M.L. Bonet, J. Levy, A new algorithm for Weighted Partial MaxSAT, in: Proc. the 24th National Conference on Artificial Intelligence (AAAI'10), 2010.
[6] C. Ansótegui, F. Manyà, Mapping problems with finite-domain variables to problems with boolean variables, in: Proc. of the 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'04), 2004, pp. 1–15.
[7] J. Argelich, D. Le Berre, I. Lynce, J.P. Marques-Silva, P. Rapicault, Solving Linux upgradeability problems using boolean optimization, in: Proceedings First International Workshop on Logics for Component Configuration (LoCoCo), 2010, pp. 11–22.
[8] J. Argelich, C.M. Li, F. Manyà, J. Planes, MaxSAT evaluations, http://www.maxsat.udl.cat, 2009, 2010, 2011.
[9] R. Asín, R. Nieuwenhuis, http://www.lsi.upc.edu/~rasin/timetabling.html, 2010.
[10] R. Asín, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, Cardinality networks: A theoretical and empirical study, Constraints 16 (2) (2011) 195–221.
[11] A. Biere, PicoSAT essentials, J. Satisf. Boolean Model. Comput. 4 (2008) 75–97.
[12] M.L. Bonet, J. Levy, F. Manyà, Resolution for Max-SAT, Artificial Intelligence 171 (8–9) (2007) 606–618.
[13] J. Davies, F. Bacchus, Solving MAXSAT by solving a sequence of simpler SAT instances, in: Proc. of the 17th Int. Conf. on Principles and Practice of Constraint Programming (CP'11), 2011, pp. 225–239.
[14] N. Eén, N. Sörensson, Translating pseudo-boolean constraints into SAT, J. Satisf. Boolean Model. Comput. 2 (1–4) (2006) 1–26.

[15] D. Frost, R. Dechter, Maintenance scheduling problems as benchmarks for constraint algorithms, Ann. Math. Artif. Intell. 26 (1–4) (1999) 149–170.
[16] Z. Fu, Extending the power of boolean satisfiability: Techniques and applications, Ph.D. thesis, Princeton, 2007.
[17] Z. Fu, S. Malik, On solving the partial Max-SAT problem, in: Proc. of the 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06), 2006, pp. 252–265.
[18] F. Heras, J. Larrosa, A. Oliveras, MiniMaxSat: A new weighted Max-SAT solver, in: Proc. of the 10th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'07), 2007, pp. 41–55.
[19] F. Heras, A. Morgado, J. Marques-Silva, Core-guided binary search algorithms for maximum satisfiability, in: Proc. the 25th National Conference on Artificial Intelligence (AAAI'11), 2011, pp. 284–297.
[20] H.A. Kautz, B. Selman, Planning as satisfiability, in: ECAI, 1992, pp. 359–363.
[21] M. Koshimura, T. Zhang, H. Fujita, R. Hasegawa, QMaxSAT: A partial Max-SAT solver, J. Satisf. Boolean Model. Comput. 8 (1/2) (2012) 95–100.
[22] D. Le Berre, SAT4J, a satisfiability library for Java, http://www.sat4j.org, 2006.
[23] C.M. Li, F. Manyà, N.O. Mohamedou, J. Planes, Exploiting cycle structures in Max-SAT, in: Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'09), 2009, pp. 467–480.
[24] H. Lin, K. Su, C.M. Li, Within-problem learning for efficient lower bound computation in Max-SAT solving, in: Proc. the 23rd National Conference on Artificial Intelligence (AAAI'08), 2008, pp. 351–356.
[25] V. Manquinho, J. Marques-Silva, J. Planes, Algorithms for weighted boolean optimization, in: Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'09), 2009, pp. 495–508.
[26] V.M. Manquinho, R. Martins, I. Lynce, Improving unsatisfiability-based algorithms for boolean optimization, in: Proc. of the 13th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'10), in: Lecture Notes in Computer Science, vol. 6175, Springer, 2010, pp. 181–193.
[27] J. Marques-Silva, J. Argelich, A. Graça, I. Lynce, Boolean lexicographic optimization: Algorithms & applications, Ann. Math. Artif. Intell. 62 (3–4) (2011) 317–343.
[28] J. Marques-Silva, I. Lynce, V.M. Manquinho, Symmetry breaking for maximum satisfiability, in: Proc. of the 15th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08), 2008, pp. 1–15.
[29] J. Marques-Silva, V.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), 2008, pp. 225–230.
[30] J. Marques-Silva, J. Planes, On using unsatisfiability for solving maximum satisfiability, arXiv:0712.1097, 2007.
[31] J. Marques-Silva, J. Planes, Algorithms for maximum satisfiability using unsatisfiable cores, in: Proc. of the Conf. on Design, Automation and Test in Europe (DATE'08), 2008, pp. 408–413.
[32] R. Martins, V.M. Manquinho, I. Lynce, Exploiting cardinality encodings in parallel maximum satisfiability, in: Proc. of the 23rd Int. Conf. on Tools with Artificial Intelligence (ICTAI'11), 2011, pp. 313–320.
[33] C.H. Papadimitriou, Computational Complexity, Academic Internet Publ., 2007.
[34] J.D. Park, Using weighted Max-SAT engines to solve MPE, in: Proc. the 24th National Conference on Artificial Intelligence (AAAI'10), 2002, pp. 682–687.
[35] D. Pisinger, An exact algorithm for large multiple knapsack problems, European J. Oper. Res. 114 (3) (1999) 528–541.
[36] S. Safarpour, H. Mangassarian, A.G. Veneris, M.H. Liffiton, K.A. Sakallah, Improved design debugging using maximum satisfiability, in: FMCAD, IEEE Computer Society, 2007, pp. 13–19.
[37] M. Sanchez, S. Bouveret, S.D. Givry, F. Heras, P. Jégou, J. Larrosa, S. Ndiaye, E. Rollon, T. Schiex, C. Terrioux, G. Verfaillie, M. Zytnicki, Max-CSP competition 2008: toulbar2 solver description, 2008.
[38] C. 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), 2005, pp. 827–831.
[39] D.M. Strickland, E. Barnes, J.S. Sokol, Optimal protein structure alignment using maximum cliques, Oper. Res. 53 (3) (2005) 389–402.
[40] M. Vasquez, J.-K. Hao, A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite, Comput. Optim. Appl. 20 (2001) 137–157.
[41] H. Xu, R.A. Rutenbar, K.A. Sakallah, Sub-SAT: A formulation for relaxed boolean satisfiability with applications in routing, IEEE Trans. Comput. Aided Des. Integrated Circ. Syst. 22 (6) (2003) 814–820.