J Heuristics (2016) 22:1–53
DOI 10.1007/s10732-015-9300-7

Exploiting subproblem optimization in SAT-based MaxSAT algorithms

Carlos Ansótegui[1] · **Joel Gabàs**[1] · **Jordi Levy**[2]

Received: 20 October 2014 / Revised: 23 July 2015 / Accepted: 16 August 2015 /
Published online: 15 September 2015

Abstract The Maximum Satisfiability (MaxSAT) problem is an optimization variant of the Satisfiability (SAT) problem. Several combinatorial optimization problems can be translated into a MaxSAT formula. Among exact MaxSAT algorithms, SAT-based MaxSAT algorithms are the best performing approaches for real-world problems. We have extended the WPM2 algorithm by adding several improvements. In particular, we show that by solving some subproblems of the original MaxSAT instance we can dramatically increase the efficiency of WPM2. This led WPM2 to achieve the best overall results at the international MaxSAT Evaluation 2013 (MSE13) on industrial instances. Then, we present additional techniques and heuristics to further exploit the information retrieved from the resolution of the subproblems. We exhaustively analyze the impact of each improvement what contributes to our understanding of why they work. This architecture allows to convert exact algorithms into efficient incomplete algorithms. The resulting solver had the best results on industrial instances at the incomplete track of the latest international MSE.

Keywords Constraint optimization · Satisfiability · Maximum Satisfiability

✉ Carlos Ansótegui
carlos@diei.udl.es

Joel Gabàs
joel.gabas@udl.cat

Jordi Levy
levy@iiia.csic.es

[1] DIEI, Universitat de Lleida, Lleida, Spain

[2] IIIA-CSIC, Barcelona, Spain

1 Introduction

Combinatorial optimization problems arise in many domains: scheduling and planning, software and hardware verification, knowledge compilation, probabilistic modeling, bioinformatics, energy systems, smart cities, social networks, computational sustainability, etc. From a computational point of view, many optimization problems are NP-hard meaning that is unlikely that they admit a polynomial-time algorithm. The good news is that some real problems are already efficiently solved by state-of-the-art Constraint Programming techniques (Rossi et al. 2006) and many others are only slightly beyond the reach of these techniques.

In recent decades, Satisfiability (SAT) solvers (Biere et al. 2009) have progressed spectacularly in performance thanks to better implementation techniques and conceptual enhancements, such as Conflict Driven Clause Learning-based algorithms, which are able to reduce the size of the search space significantly in many instances of real NP-Complete problems. Every year, the community celebrates competitions where the performance of all these solvers is compared, and every year we contemplate how the number of solvable problems increases, and how instances that were considered very difficult, become easy. Thanks to these advances, nowadays the best SAT solvers can tackle industrial problems with hundreds of thousands of variables and millions of clauses. We use the term industrial in the sense of practical or real-world applications. Some extensions of SAT that have also attracted the interest of the scientific community in recent years include: Pseudo-Boolean (PB) satisfiability, Satisfiability Modulo Theories (SMT), satisfiability of Quantified Boolean Formulas, Maximum Satisfiability (MaxSAT), Model Counting (#SAT), etc. There exist also solvers for all these extensions of SAT, and competitions where they are tested.

The MaxSAT problem is the optimization version of SAT. The idea behind this formalism is that sometimes not all the constraints 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 (WPMS) problem. In this case, we can divide the constraints into two groups: the clauses (constraints) that must be satisfied (hard), and the ones that may or may not be satisfied (soft). In the last group, we may associate different weights with the clauses, where the weight is the cost of falsifying the clause. The idea is that not all constraints 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. The WPMS problem is a natural combinatorial optimization problem, and it has been already applied in many domains (Ansótegui et al. 2013b; Morgado et al. 2013a).

In the MaxSAT community, we find two main classes of complete algorithms: branch and bound (Heras et al. 2007; Kügel 2010; Li et al. 2009; Lin and Su 2007; Lin et al. 2008) and SAT-based (Ansótegui et al. 2012; Davies and Bacchus 2011; Heras et al. 2011; Honjyo and Tanjo 2012; Koshimura et al. 2012; Martins et al. 2011, 2012; Morgado et al. 2012).

SAT-based approaches clearly dominate on industrial and some crafted instances, as we can see in the results of the international MaxSAT Evaluation (Argelich et al. 2006-2004). SAT-based MaxSAT algorithms basically reformulate a MaxSAT instance into a sequence of SAT instances. By solving these SAT instances the MaxSAT problem can be solved (see (Ansótegui et al. 2013b; Morgado et al. 2013a) for further information). Intuitively, this sequence is built such that it can be split into two parts where the instances in the first part are all unsatisfiable while the instances in the second one are all satisfiable. By locating the phase transition point, i.e., the last unsatisfiable instance and the first satisfiable instance, we can locate the optimum of the optimization problem. As we will see, once we solve an unsatisfiable SAT instance we can refine the lower bound, while when we solve a satisfiable SAT instance we can refine the upper bound. Among SAT-based MaxSAT algorithms we also find two main classes: (i) those that focus the search on refining the lower bound, and exploit the information of unsatisfiable cores and, (ii) those that focus the search on refining the upper bound, and exploit the information of the satisfiable assignments. Both approaches have strengths and weaknesses. Our current work aims to find an efficient balance between both approaches.

In this paper, we present the improved version of the SAT-based MaxSAT algorithm WPM2 (Ansotegui et al. 2010) (see Sect. 4) presented originally in Ansótegui et al. (2013a), a more detailed experimental analysis of the new algorithm, further improvements and an incomplete version. The aim of this paper is not only to present a method that performs well, but also to understand why this is the case. This way we will be able to identify the interaction with other future improvements in the field and whether they are complementary or not to this work.

In our experimental investigation, our reference point is the original WPM2 algorithm which solves 959 out of 2078 instances from the whole benchmark of industrial and crafted instances of the MSE13.

With respect to the improvements we have incorporated, first of all, we extend the original WPM2 algorithm by applying the stratification approach described in Ansótegui et al. (2012), what results in solving 100 additional instances. Second, we introduce a new criteria to decide when soft clauses can be hardened (Ansótegui et al. 2012; Morgado et al. 2012), that provides 68 additional solved instances. Finally, our most effective contribution is to introduce a new strategy that focuses search on solving subproblems of the original MaxSAT instance. In order to define these subproblems we use the information provided by the unsatisfiable cores we obtain during the solving process. The improved WPM2 algorithm is parametric on the approach we use to solve these subproblems. This allows to combine the strength of exploiting the information extracted from unsatisfiable cores and other optimization approaches. By solving these smaller optimization problems we get the most significant boost in our improved WPM2 algorithm. In particular, we experiment with an Integer Linear Programming (ILP) approach, corresponding to the strategy shown in Sect. 3, and three MaxSAT approaches: (i) refine the lower bound for these subproblems with the *subsetsum* function (Cormen et al. 2009; Ansotegui et al. 2010), (ii) refine the upper bound with the strategy applied in minisat+ (Eén and Sörensson 2006), SAT4J (Berre 2006), qmaxsat (Koshimura et al. 2012) or ShinMaxSat (Honjyo and Tanjo 2012), and (iii) a binary search scheme (Heras et al. 2011; Cimatti et al. 2010; Fu and Malik 2006) where the lower bound and upper bound are refined as in the previous approaches. The best performing approach in our experimental analysis is the second one and it allows to solve up to 296 additional instances. As a summary, the overall increase in performance we achieved so far compared to the original WPM2 is about 464 additional solved instances, or 48 % more. These improvements led WPM2 to be the overall best performing approach on the industrial instances of the MSE13.

To further explain the good results of our approach we include and extend the study originally presented in Ansotegui (2013a) on the structure of the unsatisfiable cores obtained during the search process of SAT-based algorithms (see Sect. 6). We explain how the improved WPM2 algorithm takes advantage of this structure.

We have also explored how we can exploit information retrieved from the subproblems that are solved. When the strategy used to optimize the subproblems is able to produce satisfying assignments, i.e., it refines the upper bound (see approaches (ii) and (iii) above), we can use these assignments as a heuristic to guide and boost the search. This improvement allows to solve 50 additional instances. The overall increase in performance compared to the original WPM2 is 514 additional solved instances. Actually, if we take into account the timeout of 7200 s (2 h) used in our experiments, we obtain an overall speed-up of about three orders of magnitude (see Sect. 6). Moreover, we show that high quality satisfying assignments can be obtained in a reasonably short time, giving us naturally an incomplete approach. Our experimental results confirm that the incomplete version of our exact improved WPM2 algorithm would have dominated the track for incomplete solvers at the MSE13. Furthermore, at MSE14, even though it was not the best complete approach, it dominates the others as incomplete.

From the perspective of coming up with an efficient implementation of our approach, it is obvious that SAT-based MaxSAT algorithms have to be implemented on top of an efficient solver based on SAT technology. This solver has to be capable of returning an unsatisfiable core when the input instance is unsatisfiable and a satisfying assignment when the instance is satisfiable. Moreover, SAT-based MaxSAT algorithms require the addition of linear PB constraints as a result of the reformulation process of the original problem into a sequence of SAT instances. These PB constraints are used to bound the cost of the optimal assignment. Currently, in most state-of-the-art SAT-based MaxSAT solvers, PB constraints are translated into SAT. However, there is no known SAT encoding which can guarantee the original propagation power of the constraint, i.e, what we call arc-consistency, while keeping the translation low in size. The best approach so far, has a cubic complexity (Bailleux et al. 2009). This can be a bottleneck for WPM2 (Ansotegui et al. 2010) and also for other algorithms such as, BINCD (Heras et al. 2011) or SAT4J (Berre 2006).

In order to treat PB constraints with specialized inference mechanisms and a moderate cost in size, while preserving the strength of SAT techniques for the rest of the problem, we use the SMT technology (Sebastiani 2007) (see Sect. 5). Related work in this sense can be found in Nieuwenhuis and Oliveras (2006). Also, in Ansótegui et al. (2011) a Weighted Constraint Satisfaction Problems (WCSP) solver implementing the original WPM1 (Ansotegui et al. 2009) algorithm is presented.

Finally, we have also seen the development of successful methods for solving combinatorial problems by applying techniques from Operations Research. Although the literature shows us that some NP-hard problems are more suitable for logic-based approaches while others are more efficiently solved with integer programming techniques, our current study would not be exhaustively conducted without showing the performance of integer programming techniques on MaxSAT problems. Actually, we can easily reformulate the WPMS problem into an ILP problem (see Sect. 3) and apply an ILP approach. We show that the ILP approach is not competitive on the industrial instances where a logic-based approach like the one proposed here is more efficient.

This paper proceeds as follows. Section 2 formally defines the main concepts that are used in the paper. Section 3 presents the translation of WPMS into ILP. Section 4 describes the WPM2 algorithm and the new improvements. Section 5 describes the SMT problem and discusses some implementation details of the SMT-based MaxSAT algorithms. Section 6 presents the experimental evaluation. Finally, Sect. 7 shows the conclusions and the future work.

2 Preliminaries

Definition 1 A *literal* l is either a Boolean variable x or its negation $\overline{x}$. A *clause* c is a disjunction of literals. A *SAT formula* is a set of clauses that represents a Boolean formula in Conjunctive Normal Form (CNF), i.e., a conjunction of clauses.

Definition 2 A *weighted clause* is an ordered pair (c, w), where c is a clause and w is a natural number or infinity (indicating the cost of falsifying c, see Definitions 5 and 6). If w is infinite the clause is *hard*, otherwise it is *soft*.

Definition 3 A *Weighted Partial MaxSAT (WPMS) formula* is an ordered multiset of weighted clauses:

$$\varphi = \langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty) \rangle$$

where the first s clauses are soft and the last h clauses are hard. The presence of soft clauses with different weights makes the formula Weighted and the presence of hard clauses makes it Partial. The ordered multiset of weights of the soft clauses in the formula is noted as $w(\varphi)$. The top weight of the formula is noted as $W(\varphi)$, and defined as $W(\varphi) = \sum w(\varphi) + 1$. The set of indexes of soft clauses is noted as $S(\varphi)$ and the set of indexes of hard clauses is noted as $H(\varphi)$. When it is clear to which formula φ these soft (hard) clauses belong, we also refer to these sets of indexes as S (H). Finally, the set of variables occurring in the formula is noted as $\mathrm{var}(\varphi)$.

Example 1 Given the WPMS formula $\varphi = \langle (x_1, 5), (x_2, 3), (x_3, 3), (\overline{x}_1 \vee \overline{x}_2, \infty), (\overline{x}_1 \vee \overline{x}_3, \infty), (\overline{x}_2 \vee \overline{x}_3, \infty) \rangle$, we have that $w(\varphi) = \langle 5, 3, 3 \rangle$, $W(\varphi) = 12$, $S(\varphi) = \{1, 2, 3\}$, $H(\varphi) = \{4, 5, 6\}$ and $\mathrm{var}(\varphi) = \{x_1, x_2, x_3\}$.

Definition 4 Given a WPMS formula φ and a set of indexes A, φ_A is the WPMS formula that contains the clauses (c_i, w_i) such that $i \in A$. By φ_S we refer the set of soft clauses and by φ_H to the set of hard clauses.

Example 2 Given the WPMS formula φ of Example 1, we have that $\varphi_{\{1,3,5\}} = \langle (x_1, 5), (x_3, 3), (\overline{x}_1 \vee \overline{x}_3, \infty) \rangle$, $\varphi_S = \langle (x_1, 5), (x_2, 3), (x_3, 3) \rangle$ and $\varphi_H = \langle (\overline{x}_1 \vee \overline{x}_2, \infty), (\overline{x}_1 \vee \overline{x}_3, \infty), (\overline{x}_2 \vee \overline{x}_3, \infty) \rangle$.

Definition 5 An *assignment* for a set of Boolean variables X is a function $\mathcal{I} : X \rightarrow \{0, 1\}$, that can be extended to literals, (weighted) clauses, SAT formulas and WPMS formulas as follows:

$$\begin{array}{l}
\mathcal{I}(\overline{x}) = 1 - \mathcal{I}(x) \\
\mathcal{I}(l_1 \vee \ldots \vee l_m) = \max\{\mathcal{I}(l_1), \ldots, \mathcal{I}(l_m)\} \\
\mathcal{I}(\{c_1, \ldots, c_n\}) = \min\{\mathcal{I}(c_1), \ldots, \mathcal{I}(c_n)\} \\
\mathcal{I}((c, w)) = w\,(1 - \mathcal{I}(c)) \\
\mathcal{I}(\langle (c_1, w_1), \ldots, (c_{s+h}, w_{s+h}) \rangle) = \sum_{i=1}^{s+h} \mathcal{I}((c_i, w_i))
\end{array}$$

We will refer to the value returned by an assignment $\mathcal{I}$ on a weighted clause or a WPMS formula as the cost of $\mathcal{I}$.

Given a WPMS formula φ and a set of indexes A, we will refer to $\mathcal{I}_A$ as an assignment for φ_A.

Definition 6 We say that an assignment $\mathcal{I}$ satisfies a clause or a SAT formula if the value returned by $\mathcal{I}$ is equal to 1. In the case of SAT formulas, we will refer also to this assignment as a satisfying assignment or solution. Otherwise, if the value returned by $\mathcal{I}$ is equal to 0, we say that $\mathcal{I}$ falsifies the clause or the SAT formula.

Definition 7 The *SAT problem* for a SAT formula φ is the problem of finding a solution for φ. If a solution exists the formula is satisfiable, otherwise it is unsatisfiable.

Definition 8 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.

Example 3 Given the SAT formula: $\varphi = \{(x_1), (x_2), (x_3), (\overline{x}_1 \vee \overline{x}_2), (\overline{x}_1 \vee \overline{x}_3), (\overline{x}_2 \vee \overline{x}_3)\}$ we have that $\{(x_1), (x_2), (x_3), (\overline{x}_1 \vee \overline{x}_2)\} \subseteq \varphi$ is an unsatisfiable core and $\{(x_1), (x_2), (\overline{x}_1 \vee \overline{x}_2)\} \subseteq \varphi$ is a minimal unsatisfiable core.

Definition 9 A SAT algorithm for the SAT problem, takes as input a SAT formula φ and returns an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = 1$ if the formula is satisfiable. Otherwise, it returns an unsatisfiable core φ_C.

Given unlimited resources of time and memory, we say that a SAT algorithm is *complete* if it terminates for any SAT formula. Otherwise, we say that it is *incomplete*.

Definition 10 The *optimal cost (or optimum)* of a WPMS formula φ is $\text{cost}(\varphi) = \min\{\mathcal{I}(\varphi) \mid \mathcal{I} : \text{var}(\varphi) \rightarrow \{0, 1\}\}$ and an *optimal assignment* is an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = \text{cost}(\varphi)$. We will refer to this assignment as a solution for φ if $\mathcal{I}(\varphi) \neq \infty$. Any cost above (below) *cost*(φ) is called an upper (lower) bound for φ.

Example 4 Given the WPMS formula φ of Example 1, we have that $\text{cost}(\varphi) = \min\{6, 8, 11, \infty\} = 6$ and the optimal assignment $\mathcal{I}$ maps $\langle x_1, x_2, x_3 \rangle$ to $\langle 1, 0, 0 \rangle$.

Definition 11 The *Weighted Partial MaxSAT problem* for a WPMS formula φ is the problem of finding a solution for φ. If a solution does not exist the formula is unsatisfiable.

Definition 12 A WPMS algorithm for the WPMS problem, takes as input a WPMS formula φ and returns an assignment $\mathcal{I}$, such that, $\mathcal{I}(\varphi) \geq cost(\varphi)$.

Given unlimited resources of time and memory, we say that a WPMS algorithm is *complete or exact* if for any input WPMS formula φ and returned $\mathcal{I}, \mathcal{I}(\varphi) = cost(\varphi)$. Otherwise, we say it is *incomplete*.

Definition 13 An *integer linear Pseudo-Boolean (PB) constraint* is an inequality of the form $w_1x_1 + \cdots + w_nx_n\ op\ k$, where $op \in \{\leq, \geq, =, >, <\}$, k and w_i are integer coefficients, and x_i are Boolean variables. A *cardinality constraint* is a PB constraint where the coefficients w_i are equal to 1. A PB *At-Most (At-Least) constraint* is a PB constraint where op is $\leq$ ($\geq$).

Example 5 $5x_1 + 3x_2 + 3x_3 \ \leq 6$ is a PB At-Most constraint and $5x_1 + 3x_2 + 3x_3 \ \geq 6$ is PB At-Least constraint.

3 Translation of weighted partial MaxSAT into ILP

In order to solve a WPMS problem, a first reasonable approach consists in reformulating the WPMS problem as an ILP problem and applying an ILP solver. Several encodings can be found in the literature (Li and Manyà 2009; Manquinho et al. 2009; Ansótegui and Gabàs 2013; Davies and Bacchus 2013).

Here, we describe the precise encoding we used in Ansótegui and Gabàs (2013). Given a WPMS formula, $\langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty)\rangle$, we can translate it into an ILP instance, as follows:

$$\begin{aligned}
&\text{Minimize:} \sum\nolimits_1^s w_i \cdot b_i \\
&\text{Subject to:} \\
&ILP(\varphi'_H \cup \varphi'_S) \\
&0 \leq v_i \leq 1,\ v_i \in var(\varphi'_H \cup \varphi'_S)
\end{aligned}$$

The *Minimize* section of the ILP formulation defines the objective function of the problem. This corresponds to the aggregated cost of the falsified soft clauses in the WPMS formula we want to minimize. In order to identify which soft clauses are falsified by a given assignment to the original x_i variables of the problem, we introduce an indicator variable b_i for each soft clause (c_i, w_i). These b_i are also known as reification, relaxation or blocking variables.

The *Subject to* section includes the constraints that have to be satisfied under any assignment. This corresponds to the original set of hard clauses of the WPMS formula represented by $\varphi'_H = \cup_{j=s+1}^{s+h} c_j$, and the set of clauses connecting the soft clauses of the WPMS formula with their respective indicator variable represented by $\varphi'_S = \cup_{i=1}^{s} CNF(\overline{c}_i \leftrightarrow b_i)$. Notice that by enforcing consistency, we ensure that b_i is true iff the soft clause c_i is falsified. Function $CNF(\varphi)$ transforms φ into Conjunctive Normal Form while function $ILP(\varphi)$ maps every clause $c_i \in \varphi$ into a linear inequality with operator $>$. The left-hand side of that linear inequality corresponds to the sum of the literals in c_i once mapped into integer terms, such that, literal $x(\overline{x})$ is mapped to integer term $x(1-x)$. The right-hand side corresponds to constant 0. After moving the constants to the right, the right-hand side corresponds to constant $-k$, where k is the number of negative literals in clause c_i. Finally, we add the bounding constraints that ensure that every integer variable in the ILP formulation has domain $\{0, 1\}$. It can be easily seen that, since we are minimizing, the implication $b_i \rightarrow \overline{c}_i$ from $\overline{c}_i \leftrightarrow b_i$ is unnecessary. Notice that, by the same nature of the minimization problem, variables b_i will be 0 (false) whenever it is possible and we do not need the part of the double implication that ensures that $c_i \rightarrow \overline{b}_i$, i.e., $b_i \rightarrow \overline{c}_i$.

Example 6 Given the WPMS formula $\varphi = \langle (x_1 \vee x_2, 2), (x_1 \vee \overline{x}_2, 3), (\overline{x}_1 \vee x_2, \infty), (\overline{x}_1 \vee \overline{x}_2, \infty) \rangle$, the corresponding ILP formulation is:

Minimize: $2 \cdot b_1 + 3 \cdot b_2$

Subject to:

$$\begin{array}{lll}
x_1 + x_2 + b_1 > 0; & \# \, x_1 \vee x_2 \vee b_1 & \equiv \overline{(x_1 \vee x_2)} \rightarrow b_1 \\
\quad -x_1 - b_1 > -2; & \quad \# \, \overline{x}_1 \vee \overline{b}_1 & \\
\quad -x_2 - b_1 > -2; & \quad \# \, \overline{x}_2 \vee \overline{b}_1 & \equiv b_1 \rightarrow \overline{(x_1 \vee x_2)} \\
x_1 - x_2 + b_2 > -1; & \# \, x_1 \vee \overline{x}_2 \vee b_2 & \equiv \overline{(x_1 \vee \overline{x}_2)} \rightarrow b_2 \\
\quad -x_1 - b_2 > -2; & \quad \# \, \overline{x}_1 \vee \overline{b}_2 & \\
\quad x_2 - b_2 > -1; & \quad \# \, x_2 \vee \overline{b}_2 & \equiv b_2 \rightarrow \overline{(x_1 \vee \overline{x}_2)} \\
-x_1 + x_2 > -1; & \# \, \overline{x}_1 \vee x_2 & \\
-x_1 - x_2 > -2; & \# \, \overline{x}_1 \vee \overline{x}_2 &
\end{array}$$

$0 \leq x_1 \leq 1;\ 0 \leq x_2 \leq 1;\ 0 \leq b_1 \leq 1;\ 0 \leq b_2 \leq 1;$

4 Original WPM2 algorithm and improvements

In this section, we present the complete SAT-based MaxSAT algorithm WPM2 (Ansotegui et al. 2010) for the WPMS problem and how it has been improved.

At a high description level, given an input WPMS formula φ, the original WPM2 algorithm (Ansotegui et al. 2010), described in Algorithm 1, iteratively calls a SAT solver querying whether there is an assignment to φ with a cost less than or equal to a certain k. The initial value of k is 0 and the last value is exactly the optimal cost of φ, i.e., $cost(\varphi)$. Notice that all SAT queries with a $k < cost(\varphi)$ must have a negative answer while all the queries with $k \geq cost(\varphi)$ must have a positive answer. Therefore, our optimization problem can be reformulated as identifying where the phase transition from negative answers to positive answers occurs.

More formally, the query sent to the SAT solver is a SAT formula φ^k that is satisfiable iff there is an assignment, say $\mathcal{I}$, such that $\mathcal{I}(\varphi) \leq k$. In order to construct such a formula, we need to detect which soft clauses are falsified under a certain assignment $\mathcal{I}$, sum up their cost and compare with k.

To detect if a clause is falsified, we extend every soft clause (c_i, w_i) with a unique auxiliary Boolean indicator variable b_i obtaining $(c_i \vee b_i, w_i)$. Notice that, if c_i is false, then in order to maintain consistency b_i must be true. Therefore, these b_i variables work as indicator variables that become true if a clause c_i is falsified.

To add the weights of the falsified soft clauses and compare the cost with k we use PB constraints that are translated into a SAT formula.

Example 7 Before we go into detail on algorithm WPM2, let us describe how a naive SAT-based MaxSAT algorithm would work. Let us consider a simple WPMS formula representing the Weak Pigeon-Hole Problem (Razborov 2001; Raz 2002) of 5 pigeons and 1 hole. Variable x_i is true if the ith pigeon is in the hole.

$$\varphi = \langle (x_1, 5), (x_2, 5), (x_3, 3), (x_4, 3), (x_5, 1) \ \rangle \ \cup \langle CNF(\sum x_i \leq 1, \infty)\rangle$$

The weight of the soft clauses indicates the cost of not having the ith pigeon in the hole and the hard clauses indicate that at most one pigeon can be in the hole. A naive SAT-based MaxSAT solver would search for the optimal cost, solving a sequence of φ^k SAT instances which are satisfiable iff there is an assignment to φ, i.e., to the x_i variables, with a cost less than or equal to k. In order to build such φ^k SAT instances, all the soft clauses are extended and the PB At-Most constraint, $5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \leq k$, is used to bound the aggregated cost of the falsified clauses.

$$\begin{aligned}\varphi^k = \{\ (x_1 \vee b_1), (x_2 \vee b_2), (x_3 \vee b_3), (x_4 \vee b_4), (x_5 \vee b_5)\ \} \cup CNF(\textstyle\sum x_i \leq 1) \cup \\ CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \leq k)\end{aligned}$$

Since $cost(\varphi) = 12$, we know that the SAT instances between φ^0 to φ^{11} are unsatisfiable, while the SAT instances between φ^{12} to φ^{18} (the top weight $W(\varphi) = 18$) are satisfiable. A naive binary search between 0 and 18 would check the satisfiability of this sequence: φ^9, φ^{13}, φ^{11} and φ^{12}. Since φ^{11} and φ^{12} are unsatisfiable and satisfiable, respectively, we have found the phase transition point and we can conclude that the optimal cost is 12.

The first characteristic of the original WPM2 algorithm is that it works with two sets of PB constraints: AM and AL. The set AM of PB (At-Most) constraints are used to bound the aggregated cost of the falsified soft clauses. In particular, AM is a set of PB constraints that bound the cost of non-overlapping subsets of soft clauses. More precisely, an $am \in AM$ is a PB constraint of the form $\sum_{i \in A} w_i\, b_i \leq k$ where A is a subset of the indexes of the soft clauses, b_i and w_i are the corresponding indicator variables and weights of the ith soft clause and k the concrete bound for that subset of soft clauses. When describing algorithms, we will use the object oriented programming notation $am.A$ and $am.k$ to refer to the set A and integer k related to an At-Most constraint $am \in AM$. The idea of having multiple and smaller PB At-Most constraints instead of a single one was introduced in Ansótegui et al. (2009).

The set AL of PB (At-Least) constraints are used to impose that the aggregated cost of the falsified clauses in a given subset of soft clauses must be at least some natural number. These are redundant constraints and are not necessary for the soundness of the algorithm but help to improve the performance of the SAT solver. More precisely, an $al \in AL$ is a PB constraint of the form $\sum_{i \in A} w_i\, b_i \geq k$.

For the sake of space and readability, we will use the notation $\langle A, w(\varphi_A), \leq, k\rangle$ and At-Most constraint instead of $\sum_{i \in A} w_i\, b_i \leq k$ and PB At-Most constraint in the algorithms. Mutatis mutandis for the PB At-Least constraints.

In the following, we go into detail on the original WPM2 algorithm (Algorithm 1). First of all, we check whether the set of hard clauses (φ_H) is satisfiable (Algorithm 1 line 1). If it is unsatisfiable, we can already stop since there is no solution to φ. Otherwise, the main loop of the algorithm starts (Algorithm 1 line 3). Notice that we exit this loop iff the *sat* function returns satisfiable. In that case, we have found a solution.

Algorithm 1: Original WPM2.

```
Input: $\varphi = \langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty) \rangle$
1: if $\langle \text{UNSAT}, \_, \_ \rangle = sat(\varphi_H, \_, \_)$ then return $\langle \infty, \emptyset \rangle$
2: $\langle AL, AM \rangle := \langle \emptyset, \emptyset \rangle$
3: while $true$ do
4:     $\langle st, C, \mathcal{I} \rangle := sat(\varphi, AL, AM)$
5:     if $st = \text{SAT}$ then
6:         return $\langle \mathcal{I}(\varphi), \mathcal{I} \rangle$
7:     else
8:         $\langle A, k \rangle := optimize(\varphi_S, AL, AM, C)$
9:         $AL := \{\langle A, w(\varphi_A), \geq, k \rangle\} \cup AL$
10:        $AM := \{\langle A, w(\varphi_A), \leq, k \rangle\} \cup AM \setminus \{am \in AM\}_{am.A \subseteq A}$
```

Function sat(φ, AL, AM)

```
1: $\langle A, k \rangle := \langle \bigcup_{am \in AM} am.A, \sum_{am \in AM} am.k \rangle$
2: $\varphi^k := \{\varphi.c_i\}_{i \notin A} \cup \{\varphi.c_i \vee b_i\}_{i \in A} \cup CNF(AL \cup AM)$
3: $\langle st, \varphi_C^k, \mathcal{I} \rangle := satsolver(\varphi^k)$
4: $C := \{i \in S(\varphi) \mid (\varphi.c_i \in \varphi_C^k) \vee (\varphi.c_i \vee b_i \in \varphi_C^k)\}$
5: return $\langle st, C, \mathcal{I} \rangle$
```

Function optimize(φ, AL, AM, C)

```
1: $A := \bigcup_{am \in AM,\ am.A \cap C \neq \emptyset} am.A \cup C$
2: $k := lb := subsetsum(w(\varphi_A), \sum_{am \in AM,\ am.A \subseteq A} am.k + 1)$
3: while $true$ do
4:     $\langle st, \_, \_ \rangle := sat(\_, AL, \{\langle A, w(\varphi_A), \leq, k \rangle\})$
5:     if $st = \text{SAT}$ then return $\langle A, lb \rangle$
6:     else $k := lb := subsetsum(w(\varphi_A), k + 1)$
```

The *sat* function (Algorithm 1 line 4) builds the SAT formula at the current iteration and sends it to the SAT solver. As we can see, a SAT formula φ^k is built by extending soft clauses with indicator variables and aggregating to φ^k the conversion to *CNF* of the PB constraints into the sets AL and AM (*sat* line 2). Actually, only a subset of the soft clauses is relaxed, i.e., extended with indicator variables b_i. In particular, those that have appeared previously in some unsatisfiable core. This is done because of efficiency issues.

The SAT solver outputs a triplet $\langle st, \varphi_C^k, \mathcal{I}\rangle$ (*sat* line 3). If the SAT solver returns *satisfiable* ($st = \text{SAT}$), φ_C^k is empty and $\mathcal{I}$ is the satisfying assignment (solution) found by the SAT solver. If the SAT solver returns *unsatisfiable* ($st = \text{UNSAT}$), φ_C^k is the unsatisfiable core found by the solver and $\mathcal{I}$ is empty. Finally, the *sat* function returns the status, st, the indexes of the soft clauses in the unsatisfiable core C (*sat* line 4) if $st = \text{UNSAT}$, and the satisfying assignment $\mathcal{I}$ if $st = \text{SAT}$.

When the *sat* function returns $st = \text{SAT}$ (Algorithm 1 line 5) we return the optimal cost and the optimal assignment (Algorithm 1 line 6) and the algorithm ends. When the *sat* function returns $st = \text{UNSAT}$ (Algorithm 1 line 7) we analyze the unsatisfiable core returned by the solver and we compute, within the *optimize* function (Algorithm 1 line 8), which is the set of indexes of soft clauses and bound $\langle A, k\rangle$ to construct the new At-Most constraint $\langle A, w(\varphi_A), \leq, k\rangle$ and update the AL and AM sets (Algorithm 1 lines 9 and 10). Technically speaking, the AL constraints give lower bounds on $cost(\varphi)$, while the AM constraints enforce that all solutions of the set of constraints $AL \cup AM$ are the solutions of AL of minimal cost. This ensures that any solution returned by $sat(\varphi, AL, AM)$, if there is any, has to be an optimal assignment of φ.

Within the *optimize* function, basically, we check which non-relaxed soft clauses and At-Most constraints on relaxed clauses are involved in the unsatisfiable core. The union of the indexes of all these clauses, set A (*optimize* line 1), gives us the indexes of the new At-Most constraint. Notice that the non-relaxed soft clauses involved in A will be relaxed in the next step within the *sat* function (*sat* line 2). In order to finish the building process of the new At-Most constraint, we have to compute its independent term k (*optimize* lines 2–6). Intuitively, the next k has to be the next lower bound candidate for the subproblem defined by the soft clauses related to A, i.e., φ_A . Notice that the previous candidate was the sum of the k's ($\sum_{am.A\subseteq A}^{am\in AM} am.k$) of the At-Most constraints involved into the unsatisfiable core, which were proven by the SAT solver to be too restrictive. In order to obtain the next candidate, we need to find the minimum integer k that satisfies the next conditions: (i) k is a linear combination of the weights involved in φ_A, (ii) k is greater than or equal to $(\sum_{am.A\subseteq A}^{am\in AM} am.k) + 1$ and (iii) the new At-Most constraint $\langle A, w(\varphi_A), \leq, k\rangle$ is consistent with the set of constraints in AL.

As we can see in the *optimize* function, (i) and (ii) are enforced by the *subsetsum* function (*optimize* lines 2 and 6) while (iii) is checked by the *sat* function (*optimize* line 4). The main idea is that the subset sum problem (Cormen et al. 2009) is progressively solved until we get a solution that also satisfies the AL constraints.

Actually, the *optimize* function represents the following optimization problem:

$$\begin{array}{ll}
\text{Minimize} \sum_{i\in A} w_i \cdot b_i & \text{(i)} \\
\text{Subject to:} & \\
\sum_{i\in A} w_i \cdot b_i \geq k' & \text{(ii)} \\
AL & \text{(iii)} \\
0 \leq b_i \leq 1, i \in A &
\end{array}$$

where $k' = (\sum_{am.A\subseteq A}^{am\in AM} am.k) + 1$. Notice that, by removing the AL constraints, we get the subset sum problem.

Finally, once we obtain $\langle A, k\rangle$ (Algorithm 1 line 8) to construct the new At-Most constraint $\langle A, w(\varphi_A), \leq, k\rangle$, we update the AL and AM sets. Basically, we replace in the AM set the At-Most constraints whose respective soft clauses are already considered in the new At-Most constraint (Algorithm 1 line 10). Notice that the new At-Most constraint enforces that the cost of any assignment to the subproblem φ_A has at most cost k. Optionally, we extend the AL set with an additional redundant constraint stating that the cost of any assignment to the subproblem φ_A has to be at least k (Algorithm 1 line 9).

For further information and detailed proofs on the soundness and completeness of the original WPM2 algorithm see Ansotegui et al. (2010).

Example 8 The original WPM2 algorithm performs the following iterations on the pigeon-hole formula presented in Example 7.

$$\varphi = \langle (x_1, 5), (x_2, 5), (x_3, 3), (x_4, 3), (x_5, 1)\rangle \cup \langle CNF(\sum x_i \leq 1, \infty)\rangle$$

In the first iteration, the SAT formula φ^0 is sent to the SAT solver within the *sat* function. The SAT solver certifies that it is unsatisfiable, and returns an unsatisfiable core (noted with dots $\bullet$) that involves the soft clauses 1 and 3, and the set of hard clauses. Semantically speaking, this core tells us that pigeons 1 and 3 can not be at the same time in the hole. The *optimize* function computes the new At-Most constraint ($5b_1 + 3b_3 \leq 3$) that corresponds to the subproblem $\varphi_{\{1,3\}}$. Notice that the corresponding At-Most constraint for a subproblem restricts to k the aggregated cost of its falsified soft clauses. Obviously, the first optimal candidate k for $\varphi_{\{1,3\}}$ is 3 since we will try first to leave out the pigeon with the minimum weight. Then, the soft clauses are relaxed with an additional variable and the corresponding constraints are added to AL and AM. These changes (noted with triangles $\blacktriangleleft$) transform φ^0 into φ^3. The second iteration is similar to the first one, but with the soft clauses 2 and 4. The new At-Most constraint computed by the *optimize* function is ($5b_2 + 3b_4 \leq 3$).

Iteration 1

$$\begin{array}{ll}
\varphi^0 = \{ & (x_1), \bullet \\
& (x_2), \\
& (x_3), \bullet \\
& (x_4), \\
& (x_5)\} \cup \\
& CNF(\sum x_i \leq 1) \bullet
\end{array}
\qquad
\begin{array}{ll}
\varphi^3 = \{ & (x_1 \vee b_1), \blacktriangleleft \\
& (x_2), \\
& (x_3 \vee b_3), \blacktriangleleft \\
& (x_4), \\
& (x_5)\} \cup \\
& CNF(\sum x_i \leq 1) \cup \\
& CNF(5b_1 + 3b_3 \geq 3) \blacktriangleleft \cup \\
& CNF(5b_1 + 3b_3 \leq 3) \blacktriangleleft
\end{array}$$

$$sat(\varphi, AL, AM) = \langle \text{UNSAT}, \{1,3\}, \emptyset\rangle;\ \mathcal{I}(\varphi) > 0; \qquad optimize(\varphi_S, AL, AM, \{1,3\}) = \langle \{1,3\}, 3\rangle;$$

Iteration 2

$$\begin{array}{ll}
\varphi^3 = \{ & (x_1 \vee b_1), \\
& (x_2), \bullet \\
& (x_3 \vee b_3), \\
& (x_4), \bullet \\
& (x_5)\} \cup \\
& CNF(\sum x_i \leq 1) \bullet \\
& CNF(5b_1 + 3b_3 \geq 3) \cup \\
& CNF(5b_1 + 3b_3 \leq 3)
\end{array}
\qquad
\begin{array}{ll}
\varphi^6 = \{ & (x_1 \vee b_1), \\
& (x_2 \vee b_2), \blacktriangleleft \\
& (x_3 \vee b_3), \\
& (x_4 \vee b_4), \blacktriangleleft \\
& (x_5)\} \cup \\
& CNF(\sum x_i \leq 1) \cup \\
& CNF(5b_1 + 3b_3 \geq 3) \cup \\
& CNF(5b_2 + 3b_4 \geq 3) \blacktriangleleft \cup \\
& CNF(5b_1 + 3b_3 \leq 3) \cup \\
& CNF(5b_2 + 3b_4 \leq 3) \blacktriangleleft
\end{array}$$

$$sat(\varphi, AL, AM) = \langle \text{UNSAT}, \{2,4\}, \emptyset\rangle;\ \mathcal{I}(\varphi) > 3; \qquad optimize(\varphi_S, AL, AM, \{2,4\}) = \langle \{2,4\}, 3\rangle;$$

In the third iteration, we get an unsatisfiable core for φ^6 with the soft clauses 1 and 2, the two At-Most constraints from previous subproblems and the set of hard clauses. The *optimize* function returns the subset of indexes that defines the new subproblem $\varphi_{\{1,2,3,4\}}$, with the soft clauses from the core and from the previous subproblems that intersect with it. It also returns the next optimal candidate for $\varphi_{\{1,2,3,4\}}$, $k = 8$, which satisfies the three conditions: (i) it is a linear combination of the weights of $\varphi_{\{1,2,3,4\}}$, (ii) it is greater than or equal to the addition of the subsumed previous subproblem optimal candidates plus one $(3 + 3 + 1)$ and (iii) its corresponding At-Most constraint $(5b_1 + 5b_2 + 3b_3 + 3b_4 \le 8)$ is consistent with the set of constraints in AL. We have to replace the two At-Most constraints involved in the core with the new one, that is less restrictive. Notice that the previous constraints only allowed the two pigeons with weight 3 to be out of the hole. On the other hand, the current constraint allows both one pigeon with weight 3 and one with weight 5 to be out of the hole. However, this is not enough to solve the subproblem since at-least three pigeons should be out of the hole.

Iteration 3

$$\begin{aligned}\varphi^6 = \{\ & (x_1 \vee b_1\ ),\ \bullet \\ & (x_2 \vee b_2\ ),\ \bullet \\ & (x_3 \vee b_3\ ), \\ & (x_4 \vee b_4\ ), \\ & (x_5\ \ \ \ )\,\} \cup \\ & CNF(\textstyle\sum x_i \le 1)\ \bullet \cup \\ & CNF(5b_1 + 3b_3 \ge 3) \cup \\ & CNF(5b_2 + 3b_4 \ge 3) \cup \\ & CNF(5b_1 + 3b_3 \le 3)\ \bullet \\ & CNF(5b_2 + 3b_4 \le 3)\ \bullet\end{aligned}
\qquad
\begin{aligned}\varphi^8 = \{\ & (x_1 \vee b_1\ ), \\ & (x_2 \vee b_2\ ), \\ & (x_3 \vee b_3\ ), \\ & (x_4 \vee b_4\ ), \\ & (x_5\ \ \ \ )\,\} \cup \\ & CNF(\textstyle\sum x_i \le 1) \cup \\ & CNF(5b_1 + 3b_3 \ge 3) \cup \\ & CNF(5b_2 + 3b_4 \ge 3) \cup \\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \ge 8)\ \blacktriangleleft \cup \\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \le 8)\ \blacktriangleleft\end{aligned}$$

$sat(\varphi, AL, AM) =$
$\langle \textsc{unsat}, \{1, 2\}, \emptyset\rangle;$
$\mathcal{I}(\varphi) > 6;$

$optimize(\varphi_S, AL, AM, \{1, 2\}) =$
$\langle\{1, 2, 3, 4\}, 8\rangle;$

In the fourth and the fifth iteration the *sat* function provides a core involving the soft clauses in $\varphi_{\{1,2,3,4\}}$. The *optimize* function provides new bounds 10 and 11, respectively. This last bound does allow three pigeons to be out of the hole. However, this is not yet enough to solve the whole problem.

Iteration 4

$sat(\varphi, AL, AM) =$
$\langle \textsc{unsat}, \{1, 2, 3, 4\}, \emptyset\rangle;$
$\mathcal{I}(\varphi) > 8;$

$optimize(\varphi_S, AL, AM, \{1, 2, 3, 4\}) =$
$\langle\{1, 2, 3, 4\}, 10\rangle;$

Iteration 5

$sat(\varphi, AL, AM) =$
$\langle \textsc{unsat}, \{1, 2, 3, 4\}, \emptyset\rangle;$
$\mathcal{I}(\varphi) > 10;$

$optimize(\varphi_S, AL, AM, \{1, 2, 3, 4\}) =$
$\langle\{1, 2, 3, 4\}, 11\rangle;$

In the sixth iteration, we get a core involving all the soft clauses, the At-Most constraint and the set of hard clauses. The At-Most constraint corresponding to subproblem $\varphi_{\{1,2,3,4\}}$, allows three pigeons out of the hole, but one is still in the hole. Since pigeon 5 is also in the hole, there is a conflict. We get the new constraint involving all the soft clauses and the new bound 12. In the seventh iteration, we get that φ^{12} is satisfiable and the original WPM2 algorithm ends. Notice that indeed 12 is the minimum cost that allows four pigeons out of the hole.

Iteration 6

$$\begin{array}{ll}
\varphi^{11} = \{ & (x_1 \vee b_1), \bullet \\
 & (x_2 \vee b_2), \bullet \\
 & (x_3 \vee b_3), \bullet \\
 & (x_4 \vee b_4), \bullet \\
 & (x_5 \quad) \bullet \} \cup \\
 & CNF(\sum x_i \leq 1) \bullet \cup \\
 & CNF(5b_1 + 3b_3 \geq 3) \cup \\
 & CNF(5b_2 + 3b_4 \geq 3) \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 8) \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 10) \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11) \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 11) \bullet
\end{array}$$

$sat(\varphi, AL, AM) =$
$\langle \textsc{unsat}, \{1, 2, 3, 4, 5\}, \emptyset \rangle;$
$\mathcal{I}(\varphi) > 11;$

$$\begin{array}{ll}
\varphi^{12} = \{ & (x_1 \vee b_1), \\
 & (x_2 \vee b_2), \\
 & (x_3 \vee b_3), \\
 & (x_4 \vee b_4), \\
 & (x_5 \vee b_5) \blacktriangleleft \} \cup \\
 & CNF(\sum x_i \leq 1) \cup \\
 & CNF(5b_1 + 3b_3 \geq 3) \cup \\
 & CNF(5b_2 + 3b_4 \geq 3) \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 8) \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 10) \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11) \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 + 1b_5 \geq 12) \blacktriangleleft \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 + 1b_5 \leq 12) \blacktriangleleft
\end{array}$$

$optimize(\varphi_S, AL, AM, \{1, 2, 3, 4, 5\}) =$
$\langle \{1, 2, 3, 4, 5\}, 12 \rangle;$

Iteration 7

$sat(\varphi, AL, AM) = \langle \textsc{sat}, \emptyset, \mathcal{I} \rangle;\ cost(\varphi) = 12;$

In what follows, we present how the original WPM2 algorithm can be improved by the application of the stratified approach, the hardening of soft clauses, the optimization of subproblems, and the exploitation of the assignments whose costs are upper bounds on the subproblems. The first three improvements were already applied in Ansótegui et al. (2013a).

Finally, we show that a collateral result of optimizing these subproblems by refining the upper bound is to turn the original WPM2 complete algorithm into an incomplete algorithm given a fixed amount of time or memory resources. We present the improvements in Algorithm 2 by extending Algorithm 1. The improvements are underlined.

4.1 Stratified approach

The first improvement corresponds to the stratified approach introduced in Ansótegui et al. (2012) for algorithm WPM1. The stratified approach (Algorithm 2 lines 2, 4, 7 and 9) consists in sending to the SAT solver only a subset of the soft clauses, i.e., φ_M (Algorithm 2 line 4). For the sake of clarity, we will refer to this subset as a module. More precisely, a module M is the set of indexes of the clauses to be sent to the SAT solver. Therefore, when the *sat* function returns $st = \textsc{sat}$, it means that we have solved the subproblem φ_M. If φ_M is equal to φ, then we have solved the whole problem and we can finish (Algorithm 2 line 7). A crucial point is how we extend our current module. This action is performed by the *newmodule* function (Algorithm 2 line 9).

In our current approach, we follow the stratified strategy applied in Ansótegui et al. (2012). There, a module M is formed by the indexes of the clauses whose weight is greater than or equal to a certain weight w_{max}. Initially, w_{max} is ∞. In order to extend the current module, we apply the diversity heuristic (Algorithm 2 line 9) which supplies us with an efficient method to calculate how we have to reduce the value of w_{max}. In particular, when there is a low diversity of weights in $w(\varphi \setminus \varphi_M)$, w_{max} is decreased to $max\ w(\varphi \setminus \varphi_M)$, while when there is a high diversity of weights, w_{max} decreases faster to keep the diversity of $w(\varphi \setminus \varphi_M)$ low. A similar approach with an alternative heuristic for grouping clauses can be found in Martins et al. (2012). In Morgado et al. (2013b), it is proposed to only consider in the working formula or the current module, those clauses that have been falsified by a satisfying assignment corresponding to an upper bound.

4.2 Clause hardening

The hardening of soft clauses in MaxSAT SAT-based solvers has been previously studied in Borchers and Furman (1998), Li et al. (2007), Larrosa et al. (2008), Heras et al. (2008), Marques-Silva et al. (2011), Ansótegui et al. (2012), and Morgado et al. (2012). Inspired by these works we study a hardening scheme for WPM2. While clause hardening was reported to have no positive effect in WPM1 (Ansótegui et al. 2012), we will see that it boosts efficiency in WPM2.

The clause hardening (Algorithm 2 lines 2, 8 and 11) consists in considering hard those soft clauses whose satisfiability we know does not need to be reconsidered. We need some lemma ensuring that falsifying those soft clauses would lead us to non-optimal assignments. In the case of WPM1, all soft clauses satisfying $w_i > W$, where $W = \sum\{w_i \mid (c_i, w_i) \in \varphi \wedge w_i < w_{max}\}$ is the sum of weights of clauses not sent to the SAT solver, can be hardened.

The correctness of this transformation is ensured by the following lemma:

Algorithm 2: Improved WPM2.

```
    Input: $\varphi = \langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty) \rangle$
1:  if $\langle \text{UNSAT}, \_, \_, \_ \rangle = sat(\varphi_H, \_, \_, \_)$ then return $\langle \infty, \emptyset \rangle$
2:  $\langle AL, AM\underline{, \beta, \mathcal{I}_S, H', M} \rangle := \langle \emptyset, \emptyset\underline{, \emptyset, \emptyset, \emptyset, newmodule(\varphi, H(\varphi))} \rangle$
3:  while $true$ do
4:      $\langle st, C, \underline{\mathcal{I}_M, \beta} \rangle := sat(\varphi_{\underline{M}}, AL, AM\underline{, \beta})$
5:      if $st =$ SAT then
6:          $\underline{\mathcal{I}_S := \operatorname*{arg\,min}^{\mathcal{I} \in \{\mathcal{I}_M, \mathcal{I}_S\}} \mathcal{I}(\varphi)}$
7:          $\underline{\text{if } \varphi_M = \varphi \text{ then}}$ return $\langle \mathcal{I}_{\underline{M}}(\varphi), \mathcal{I}_{\underline{M}} \rangle$
8:          $\underline{H' := harden(\varphi, AM, M)}$
9:          $\underline{M := newmodule(\varphi, M)}$
10:     else
11:         $\langle A, k\underline{, \mathcal{I}_A, \mathcal{I}_S} \rangle := optimize(\varphi_{S\underline{\cup H}}, AL, AM, C\underline{, H', \mathcal{I}_S})$
12:         $\underline{\beta := \{b_i \mid \mathcal{I}_A(\varphi.c_i) = 0\}^{i \in A} \cup \{\overline{b_i} \mid \mathcal{I}_A(\varphi.c_i) = 1\}^{i \in A} \cup \beta_{S \setminus A}}$
13:         $AL := \{\langle A, w(\varphi_A), \geq, k \rangle\} \cup AL$
14:         $AM := \{\langle A, w(\varphi_A), \leq, k \rangle\} \cup AM \setminus \{am \in AM\}_{am.A \subseteq A}$
```

Lemma 1 (Lemma 24 in Ansótegui et al. (2013b))

Let $\varphi_1 = \{(c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty)\}$ *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*

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

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

However, this lemma is not useful in the case of WPM2 because we do not proceed by transforming the formula, like in WPM1. Therefore, we generalize this lemma. For this, we need to introduce the notion of *optimal candidate* of a formula.

Function sat($\varphi, AL, AM\underline{, \beta}$)

```
1: $\langle A, k\rangle := \langle \bigcup_{am\in AM} am.A, \sum am.k\rangle$
2: $\varphi^k := \{\varphi.c_i\}_{i\notin A} \cup \{\varphi.c_i \vee b_i\}_{i\in A} \cup CNF(AL \cup AM)\underline{\cup\beta}$
3: $\underline{\textbf{repeat}}$
4:     $\langle st, \varphi^k_C, \mathcal{I}\rangle := satsolver(\varphi^k)$
5:     $\underline{\langle \beta, \varphi^k\rangle := \langle \beta \setminus \varphi^k_C, \varphi^k \setminus \beta\rangle}$
6: $\underline{\textbf{until } (\beta \cap \varphi^k_C = \emptyset)}$
7: $C := \{i \in S(\varphi) \mid (\varphi.c_i \in \varphi^k_C) \vee (\varphi.c_i \vee b_i \in \varphi^k_C)\}$
8: return $\langle st, C, \mathcal{I}\underline{, \beta}\rangle$
```

Function optimize($\varphi, AL, AM, C\underline{, H', \mathcal{I}_S}$)

```
1: $A := \underline{(}\ \bigcup_{am\in AM,\ am.A\cap C\neq\emptyset} am.A\ \cup C\underline{)\setminus H'}$
2: $k := lb := subsetsum(w(\varphi_A), \sum_{am\in AM,\ am.A\subseteq A} am.k + 1)$
3: $\underline{ub := W(\varphi_A)}$
4: while $true$ do
5:     $\underline{k := strategy(\overset{\substack{refine\\ lower\ bound}}{lb}, \overset{binary\ search}{\frac{lb+ub}{2}}, \overset{\substack{refine\\ upper\ bound}}{ub-1})}$
6:     $\langle st, \_, \mathcal{I}_A, \_\rangle := sat(\underline{\varphi_{A\cup H\cup H'}}, AL, \{\langle A, w(\varphi_A), \le, k\rangle\}\underline{\cup\{am \in AM\}^{am.A\subseteq H'}}, \_)$
7:     if $st = $ SAT then
8:         $\underline{\langle \mathcal{I}_S, ub\rangle := \langle \operatorname{arg\,min}_{\mathcal{I}\in\{\mathcal{I}_A,\mathcal{I}_S\}} \mathcal{I}(\varphi), \mathcal{I}_A(\varphi_A)\rangle}$
9:         $\underline{\textbf{if } lb = ub \textbf{ then}}$ return $\langle A, lb\underline{, \mathcal{I}_A, \mathcal{I}_S}\rangle$
10:    else
11:        $k := lb := subsetsum(w(\varphi_A), k + 1)$
12:        $\underline{\textbf{if } lb = ub \textbf{ then return } \langle A, lb, \mathcal{I}_A, \mathcal{I}_S\rangle}$
```

Definition 14 Given a WPMS formula $\varphi = \langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty) \rangle$, we say that k is an optimal candidate of φ if there exists a subset $A \subseteq \{1, \ldots, s\}$ such that $\sum_{i \in A} w_i = k$.

Notice that, for any assignment $\mathcal{I}$ of the variables of φ, we have that $\mathcal{I}(\varphi)$ is an optimal candidate of φ. However, if k is an optimal candidate, there does not exist necessarily an assignment $\mathcal{I}$ satisfying $\mathcal{I}(\varphi) = k$. Notice also that, given φ and k, finding the *next optimal candidate*, i.e., finding the smallest $k' > k$ such that k' is an optimal candidate of φ is equivalent to the subset sum problem.

Lemma 2 *Let $\varphi_1 \cup \varphi_2$ be a WPMS formula and k_1 and k_2 values such that: cost$(\varphi_1 \cup \varphi_2) = k_1 + k_2$ and any assignment $\mathcal{I}$ satisfies $\mathcal{I}(\varphi_1) \geq k_1$ and $\mathcal{I}(\varphi_2) \geq k_2$. Let k' be the smallest possible optimal candidate of φ_2 such that $k' > k_2$. Let φ_3 be a set of soft clauses with $W = \sum\{w_i \mid (c_i, w_i) \in \varphi_3\}$.*

Then, if $W < k' - k_2$, then any optimal assignment $\mathcal{I}'$ of $\varphi_1 \cup \varphi_2 \cup \varphi_3$ assigns $\mathcal{I}'(\varphi_2) = k_2$

Proof Let $\mathcal{I}'$ be any optimal assignment of $\varphi_1 \cup \varphi_2 \cup \varphi_3$. On the one hand, as for any other assignment, we have $\mathcal{I}'(\varphi_2) \geq k_2$.

On the other hand, any of the optimal assignments $\mathcal{I}$ of $\varphi_1 \cup \varphi_2$ can be extended (does not matter how) to the variables of var$(\varphi_3) \setminus$ var$(\varphi_1 \cup \varphi_2)$, such that

$$\mathcal{I}(\varphi_1 \cup \varphi_2 \cup \varphi_3) = \mathcal{I}(\varphi_1) + \mathcal{I}(\varphi_2) + \mathcal{I}(\varphi_3) \leq k_1 + k_2 + W < k_1 + k' \qquad (1)$$

Now, assume that $\mathcal{I}'(\varphi_2) \neq k_2$, then $\mathcal{I}'(\varphi_2) \geq k'$. As any other assignment, $\mathcal{I}'(\varphi_1) \geq k_1$. Hence, $\mathcal{I}'(\varphi_1 \cup \varphi_2 \cup \varphi_3) \geq k_1 + k' > \mathcal{I}(\varphi_1 \cup \varphi_2 \cup \varphi_3)$, but this contradicts the optimality of $\mathcal{I}'$. Therefore, $\mathcal{I}'(\varphi_2) = k_2$. □

Example 9 It may seem that the condition of Lemma 2 is hard to satisfy unless φ_1 and φ_2 are over disjoint sets of variables. This is not the case, and here we present a simple example where φ_1 and φ_2 share variables and Lemma 2 holds:

$$\varphi_H = CNF(((x_1 + x_2 \leq 1), \infty), ((x_3 + x_4 \leq 1), \infty), ((x_1 + x_2 + x_3 + x_4 \leq 2), \infty))$$

$$\varphi_1 = \langle (x_1, 1), (x_2, 1) \rangle \cup \varphi_H$$

$$\varphi_2 = \langle (x_3, 1), (x_4, 1) \rangle \cup \varphi_H$$

$$\varphi_1 \cup \varphi_2 = \langle (x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1) \rangle \cup \varphi_H$$

Notice that the condition of Lemma 2 is satisfied for φ_1 and φ_2, since $k_1 = cost(\varphi_1) = 1$, $k_2 = cost(\varphi_2) = 1$ and $k_1 + k_2 = cost(\varphi_1 \cup \varphi_2) = 2$.

In order to apply this lemma we have to consider formulas $\varphi_1 \cup \varphi_2$ ensuring cost$(\varphi_1 \cup \varphi_2) = k_1 + k_2$ and $\mathcal{I}(\varphi_1) \geq k_1$ and $\mathcal{I}(\varphi_2) \geq k_2$, for any assignment $\mathcal{I}$. This can be easily ensured, in the case of WPM2, if both φ_1 and φ_2 are subproblems. Then, we only have to check if the next optimal candidate k' of φ_2 exceeds the previous one k_2 more than the sum W of the weights of the clauses not sent to the SAT solver. In such a case, we can consider all soft clauses of φ_2 and their corresponding AM constraint with k_2 as hard clauses. In other words, we do not need to recompute the optimal k_2 of φ_2.

The $harden(\varphi, AM, M)$ function (Algorithm 2 line 8) returns the set of indexes of soft clauses H' that needs to be considered hard based on the previous analysis according to: the current set of At-Most constraints AM, the next optimal candidates of these constraints and the sum of the weights W of soft clauses beyond the current w_{max}, i.e., not yet sent to the SAT solver.

Function harden(φ, AM, M)

```
1: $H' = \{i \mid \langle A, w(\varphi_A), \leq, k\rangle \in AM_{\{i\}} \wedge \sum w(\varphi \setminus \varphi_M) < subsetsum(w(\varphi_A), k+1) - k\}$
2: return $H'$
```

Finally, in the *optimize* function (Algorithm 2 line 11) we introduce H' since as we will see in the next subsection, we need to know which are all the hard clauses to this point of the execution.

4.3 Subproblem optimization

As we have mentioned earlier, one key point in WPM2 is how to compute $\langle A, k\rangle$ within the *optimize* function (Algorithm 2 line 11) to construct the new At-Most constraint ($\langle A, w(\varphi), \geq, k\rangle$). In the original WPM2 algorithm, the idea was to compute the next lower bound candidate, k, for the subproblem φ_A. We can go further and set k to the optimal cost of the subproblem $\varphi_{A\cup H}$, i.e., $k = cost(\varphi_{A\cup H})$.[1]

In order to do this, while taking advantage of the AL constraints generated so far, we only have to extend the definition of the minimization problem corresponding to the original *optimize* function, by adding $\varphi_{A\cup H}$ to the *Subject to* section.

To solve $\varphi_{A\cup H}$, we can use any complete approach related to MaxSAT, such as, MaxSAT branch and bound algorithms, MaxSAT SAT-based algorithms, saturation under the MaxSAT resolution rule (Larrosa and Heras 2005; Bonet et al. 2006), or we can use other solving techniques such as PB solvers or ILP techniques, etc. Therefore the improved WPM2 algorithm is parametric on any suitable optimization solving approach. In this work, we experimented with an ILP approach, corresponding to the strategy shown in Sect. 3, and three MaxSAT approaches (new *optimize* line 5) that we describe in the next lines.

The first and natural approach consists in iteratively refining (increasing) the lower bound ($k = lb$) on $cost(\varphi_{A\cup H})$ by applying the *subsetsum* function as in the original WPM2 (new *optimize* lines 5 and 11). The procedure stops when lb satisfies the constraints $AL \cup \varphi_{A\cup H}$ (new *optimize* line 9). Notice that, since we have included $\varphi_{A\cup H}$ into the set of constraints, we will get an optimal assignment or solution for $\varphi_{A\cup H}$.

The second approach consists in iteratively refining (decreasing) the upper bound following the strategy applied in minisat+ (Eén and Sörensson 2006), SAT4J (Berre

[1] For the sake of clarity, we will obviate in the following mentioning the hardened soft clauses (H') due to the clause hardening technique (see Sect. 4.2).

2006), qmaxsat (Koshimura et al. 2012) or ShinMaxSat (Honjyo and Tanjo 2012). The upper bound ub is initially set to the top weight of φ_A. Then, we iteratively check whether there exists an assignment for $\varphi_{A\cup H}$ with cost $k = ub - 1$. Whenever we get a satisfying assignment ($\mathcal{I}_A$) we update ub to $\mathcal{I}_A(\varphi_A)$, i.e., the sum of the weights w_i of those soft clauses falsified under the satisfying assignment (new *optimize* line 5). Notice that since $\mathcal{I}_A$ is a satisfying assignment, it follows that $\mathcal{I}_A(\varphi_A) = \mathcal{I}_A(\varphi_{A\cup H})$. If we get an unsatisfiable answer, the previous upper bound ($\mathcal{I}_A(\varphi_A)$) is the optimal cost for $\varphi_{A\cup H}$ (new *optimize* lines 11 and 12).

The third approach applies a binary search scheme (Heras et al. 2011; Cimatti et al. 2010; Fu and Malik 2006) on k (new *optimize* line 5). We additionally refine the lower bound (lb) as in our first approach and the upper bound ($\mathcal{I}_A(\varphi_A)$) as in the second approach. The search ends when lb and $\mathcal{I}_A(\varphi_A)$ are equal (new *optimize* lines 9 and 11).

A final remark is that, if we combine this technique with the previous hardening technique, then we simply have to take into account the set of indexes of soft clauses H' that became hard. As we can see in the new *optimize* function (Algorithm 2 line 11), we first compute the set A as in the original function, but excluding the soft clauses that became hard H' (new *optimize* line 1). Then, we call the *sat* function but adding $\varphi_{A\cup H\cup H'}$ and the set of At-Most constraints involved in H', i.e., $\{am \in AM\}_{am.A\subseteq H'}$ (new *optimize* line 6).

The worst case complexity, in terms of the number of calls to the SAT solver, of the improved WPM2 algorithm is the number of times that the *optimize* function is called (bounded by the number of soft clauses) multiplied by the number of SAT calls needed in each call to the *optimize* function. This latter number is logarithmic on the sum of the weights of the clauses of the core if we use a binary search, hence essentially the number of clauses. Therefore, the worst case complexity, when using a binary search to solve the subproblems, is quadratic on the number of soft clauses.

In order to see that the number of calls to the *optimize* function is bounded by the number of clauses we just need to recall that WPM2 merges the At-Most constraints. Consider a binary tree where the soft clauses are the leaves, and the internal nodes represent the merges (calls to the *optimize* function). A binary tree of n leaves has n−1 internal nodes.

Solving all the subproblems exactly can be very costly since these are NP-hard problems. Notice that some of these subproblems can be integrated soon into another subproblem which we will also solve. A reasonable strategy would be to solve a subproblem when it appears for the second time, meaning that the associated k is not the optimal cost. However, in practice, we found that the more efficient strategy was to solve a subproblem only if it incorporates a previous subproblem.

Example 10 The improved WPM2 algorithm performs different iterations from the ones of the original WPM2 in Example 8 on the pigeon-hole formula presented in Example 7.

$$\varphi = \langle (x_1, 5), (x_2, 5), (x_3, 3), (x_4, 3), (x_5, 1)\rangle \cup \langle CNF(\sum x_i \leq 1, \infty)\rangle$$

In contrast to the original, the improved WPM2 algorithm performs fewer calls to the *sat* function with an unsatisfiable response. Therefore, it performs fewer calls to the *optimize* function and fewer updates to AL and AM. Besides, the *sat* function deals with formulas φ_M with fewer soft clauses and sets AL and AM with shorter constraints. We find the first difference in the first iteration where, applying the stratified approach, we consider only a subproblem φ_M with those clauses whose weight $w_i \geq 5$. The first unsatisfiable core (noted with dots $\bullet$), involves the soft clauses 1 and 2, and the set of hard clauses. The *optimize* function computes the new At-Most constraint that corresponds to the subproblem $\varphi_{\{1,2\}\cup H}$. The optimal cost k for this subproblem is 5, since this is the weight of both pigeons. Notice that, applying the stratified approach, we get a better first lower bound for the formula. The soft clauses are relaxed and the corresponding constraints are added to AL and AM (noted with triangles $\blacktriangleleft$). In the second iteration, we get that φ_M^5 is satisfiable. Since φ_M is not equal to φ, we compute a new module M with those clauses whose weight $w_i \geq 3$.

Iteration 1

$$\begin{array}{ll}
\varphi_M^0 = \{\ (x_1),\ \bullet & \varphi_M^5 = \{\ (x_1 \vee\ b_1\ ),\ \blacktriangleleft \\
\quad (x_2),\ \bullet\ \} \cup & \quad (x_2 \vee\ b_2\ ),\ \blacktriangleleft\ \} \cup \\
\quad CNF(\sum x_i \leq 1)\ \bullet & \quad CNF(\sum x_i \leq 1) \cup \\
 & \quad CNF(5b_1 + 5b_2 \geq 5)\ \blacktriangleleft \cup \\
 & \quad CNF(5b_1 + 5b_2 \leq 5)\ \blacktriangleleft \\
 & \\
sat(\varphi_M, AL, AM, \_) = & optimize(\varphi, AL, AM, \{1,2\}, \{\}, \_) = \\
\langle \text{UNSAT}, \{1,2\}, \emptyset, \_\rangle; & \langle \{1,2\}, 5, \_, \_\rangle; \\
\mathcal{I}(\varphi) > 0; &
\end{array}$$

Iteration 2

$$sat(\varphi_M, AL, AM, \_) = \langle \text{SAT}, \emptyset, \mathcal{I}, \_\rangle;\ \varphi_M \neq \varphi \rightarrow \mathcal{I}(\varphi) \geq 5;\ M = \{1,2,3,4\};$$

The third iteration is similar to the first one, but with the soft clauses 3 and 4.

Iteration 3

$$\begin{array}{ll}
\varphi_M^5 = \{\ (x_1 \vee\ b_1\ ), & \varphi_M^8 = \{\ (x_1 \vee\ b_1\ ), \\
\quad (x_2 \vee\ b_2\ ), & \quad (x_2 \vee\ b_2\ ), \\
\quad (x_3\qquad),\ \bullet & \quad (x_3 \vee\ b_3\ ),\ \blacktriangleleft \\
\quad (x_4\qquad),\ \bullet\ \} \cup & \quad (x_4 \vee\ b_4\ ),\ \blacktriangleleft\ \} \cup \\
\quad CNF(\sum x_i \leq 1)\ \bullet \cup & \quad CNF(\sum x_i \leq 1) \cup \\
\quad CNF(5b_1 + 5b_2 \geq 5) \cup & \quad CNF(5b_1 + 5b_2 \geq 5) \cup \\
\quad CNF(5b_1 + 5b_2 \leq 5) & \quad CNF(3b_3 + 3b_4 \geq 3)\ \blacktriangleleft \cup \\
 & \quad CNF(5b_1 + 5b_2 \leq 5) \cup \\
 & \quad CNF(3b_3 + 3b_4 \leq 3)\ \blacktriangleleft \\
 & \\
sat(\varphi_M, AL, AM, \_) = & optimize(\varphi, AL, AM, \{3,4\}, \{\}, \_) = \\
\langle \text{UNSAT}, \{3,4\}, \emptyset, \_\rangle; & \langle \{3,4\}, 3, \_, \_\rangle; \\
\mathcal{I}(\varphi) > 5; &
\end{array}$$

It is in the fourth iteration where we can appreciate the impact of subproblem optimization. In Example 8, we needed three iterations to get the bound 11 for the At-Most constraint corresponding to the subproblem $\varphi_{\{1,2,3,4\}}$. Incorporating the hard clauses to the *optimize* function and solving the subproblem $\varphi_{\{1,2,3,4\}\cup H}$, we get directly the optimal cost 11 in just one iteration. In the fifth iteration, we get that φ_M^5 is satisfiable. As φ_M is not equal to φ we have to compute a new module again. Before doing that, we apply the hardening technique. We get that the soft clauses 1, 2, 3 and 4, and their corresponding At-Most constraint can be considered as hard (H'). This is because the current k of this At-Most constraint is 11 $(5 + 3 + 3)$ and its next k' candidate is 13 $(5 + 5 + 3)$, while the addition of weights of those soft clauses not yet in M, i.e., {5}, is only 1. So, reconsidering this At-Most constraint would lead to an assignment with a higher cost than falsifying all the soft clauses not yet in M. Semantically speaking, the increase in cost of any non-allowed distribution of pigeons 1, 2, 3 and 4, is always higher than the cost of allowing the pigeon 5 to be out of the hole. After applying the hardening technique, we compute the new module M (clauses whose weight $w_i \geq 1$) that corresponds to the whole problem (i.e., in the next iteration with a satisfiable response from the *sat* function, we will have found the solution).

Iteration 4

$$
\begin{array}{ll}
\varphi_M^8 = \{\ (x_1 \vee\ b_1\ ),\ \bullet & \varphi_M^{11} = \{\ (x_1 \vee\ b_1\ ), \\
\quad (x_2 \vee\ b_2\ ),\ \bullet & \quad (x_2 \vee\ b_2\ ), \\
\quad (x_3 \vee\ b_3\ ),\ \bullet & \quad (x_3 \vee\ b_3\ ), \\
\quad (x_4 \vee\ b_4\ ),\ \bullet\ \} \cup & \quad (x_4 \vee\ b_4\ ), \cup \\
\quad CNF(\sum x_i \leq 1)\ \bullet \cup & \quad CNF(\sum x_i \leq 1) \cup \\
\quad CNF(5b_1 + 5b_2 \geq 5) \cup & \quad CNF(5b_1 + 5b_2 \geq 5) \cup \\
\quad CNF(3b_3 + 3b_4 \geq 3) \cup & \quad CNF(3b_2 + 3b_4 \geq 3) \cup \\
\quad CNF(5b_1 + 5b_2 \leq 5)\ \bullet \cup & \quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11)\ \blacktriangleleft \cup \\
\quad CNF(3b_3 + 3b_4 \leq 3)\ \bullet & \quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 11)\ \blacktriangleleft
\end{array}
$$

$$
\begin{array}{ll}
sat(\varphi_M, AL, AM, \_) = & optimize(\varphi, AL, AM, \{1,2,3,4\}, \{\}, \_) = \\
\langle \text{UNSAT}, \{1,2,3,4\}, \emptyset, \_\rangle; & \langle \{1,2,3,4\}, 11, \_, \_\rangle; \\
\mathcal{I}(\varphi) > 8; &
\end{array}
$$

Iteration 5

$$sat(\varphi_M, AL, AM, \_) = \langle \text{SAT}, \emptyset, \mathcal{I}, \_\rangle;\ \ \varphi_M \neq \varphi \rightarrow \mathcal{I}(\varphi) \geq 11;\ H' = \{1,2,3,4\};\ \ M = \{1,2,3,4,5\};$$

In the sixth iteration, the *sat* function returns a core with all the soft clauses, the At-Most constraint and the set of hard clauses. Applying the hardening technique, the soft clauses with indexes in H' and the corresponding At-Most constraint are considered as hard. Therefore, the new At-Most constraint computed by the *optimize* function involves only the soft clause 5 and has the new bound $k = 1$. In the seventh iteration, we get that φ_M^{12} is satisfiable. Since φ_M is equal to φ, 12 is the solution to the problem.

Iteration 6

$$
\begin{array}{ll}
\varphi_M^{11} = \{\ (x_1 \vee b_1\ ),\ \bullet & \varphi_M^{12} = \{\ (x_1 \vee\ b_1\ ), \\
\quad (x_2 \vee b_2\ ),\ \bullet & \quad (x_2 \vee\ b_2\ ), \\
\quad (x_3 \vee b_3\ ),\ \bullet & \quad (x_3 \vee\ b_3\ ), \\
\quad (x_4 \vee b_4\ ),\ \bullet & \quad (x_4 \vee\ b_4\ ), \\
\quad (x_5 \qquad)\ \bullet\ \} \cup & \quad (x_5 \vee\ b_5\ )\ \blacktriangleleft \} \cup \\
\quad CNF(\sum x_i \leq 1)\ \bullet \cup & \quad CNF(\sum x_i \leq 1) \cup \\
\quad CNF(5b_1 + 5b_2 \geq 5) \cup & \quad CNF(5b_1 + 5b_2 \geq 5) \cup \\
\quad CNF(3b_2 + 3b_4 \geq 3) \cup & \quad CNF(3b_2 + 3b_4 \geq 3) \cup \\
\quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11) \cup & \quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11) \cup \\
\quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 11)\ \bullet & \quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 11) \cup \\
 & \quad CNF(1b_5 \geq 1)\ \blacktriangleleft \cup \\
 & \quad CNF(1b_5 \leq 1)\ \blacktriangleleft
\end{array}
$$

$$
\begin{array}{ll}
sat(\varphi_M, AL, AM, \_) = & optimize(\varphi, AL, AM, \{1,2,3,4,5\}, \{1,2,3,4\}, \_) = \\
\langle \text{UNSAT}, \{1,2,3,4,5\}, \emptyset, \_\rangle & \langle \{5\}, 1\rangle; \\
\mathcal{I}(\varphi) > 11; &
\end{array}
$$

Iteration 7

$$sat(\varphi_M, AL, AM, \_) = \langle \text{SAT}, \emptyset, \mathcal{I}, \_\rangle;\ \varphi_M = \varphi \rightarrow cost(\varphi) = 12;$$

4.4 Exploiting satisfying assignments from subproblems

Whenever we obtain an upper bound or solution for a subproblem, we can obtain an upper bound for the whole problem. Consider $\varphi_{A \cup H}$, where A is a subset of the indexes of the soft clauses in a WPMS formula φ and H the set of indexes of the hard clauses, as the subproblem we want to solve. According to the search strategy we use in the *optimize* function (see Sect. 4.3) we may obtain assignments $\mathcal{I}_A$ (Algorithm 2 line 6) that are upper bounds or directly a solution for $\varphi_{A \cup H}$. If we extend this assignment by assigning a random value in $\{0, 1\}$ to every variable in $var(\varphi) \setminus \text{var}(\varphi_{A \cup H})$, then it is not difficult to see that $\mathcal{I}_A(\varphi) \geq cost(\varphi)$, i.e., $\mathcal{I}_A(\varphi)$ is an upper bound for φ. Therefore, by comparing $\mathcal{I}_A(\varphi)$ with the cost of the best assignment found so far $\mathcal{I}_S(\varphi)$ (Algorithm 2 line 8), the improved WPM2 algorithm becomes naturally an incomplete approach. It is incomplete in the sense that it reports the assignment with the best cost found within restricted time and memory resources.

Also, due to the stratification approach, once we obtain a satisfying assignment $\mathcal{I}_M$ for a module M (Algorithm 2 line 4), we may have obtained a better upper bound for φ. Therefore, following the same idea, we update conveniently $\mathcal{I}_S$ (Algorithm 2 line 6) as in the *optimize* function.

Obviously, this incomplete approach makes sense if we are able to obtain quickly good quality upper bounds. We will show that this is the case in the experimental section (see Sect. 6). However, let us visualize the behavior of the improved WPM2 algorithm on a particular instance. In Fig. 1, we show the upper bounds $\mathcal{I}_A(\varphi)$ and lower bounds obtained during the execution of the improved WPM2 on an industrial Partial MaxSAT formula φ. The upper bound refinement strategy was used for the subproblem optimization phase.

For the sake of comparison, we also show the lower bounds obtained with the original WPM2 algorithm. The objective is to show that the improved WPM2 algorithm not only is able to provide good upper bounds, it also converges faster to the optimal cost.

In the x-axis, we show the elapsed seconds of the search in a logarithmic scale. The y-axis correspond to the range of the objective function (from 0 to top weight). In the upper half (from optimum to top weight) we show the value of the obtained upper bounds and in the lower half (from optimum to 0) we show the value of the lower bounds.

As we can see, the original WPM2 does not provide any upper bound until the optimum is found. In contrast, the improved WPM2 does provide several upper bounds coming from the subproblem optimization phase.

Both algorithms provide lower bounds. Every lower bound update corresponds to a new k for a subproblem obtained after a call to the *optimize* function and is followed by a call to the *sat* function to check the satisfiability of the whole problem. Notice that the subproblem optimization occurs always between lower bound updates. In the

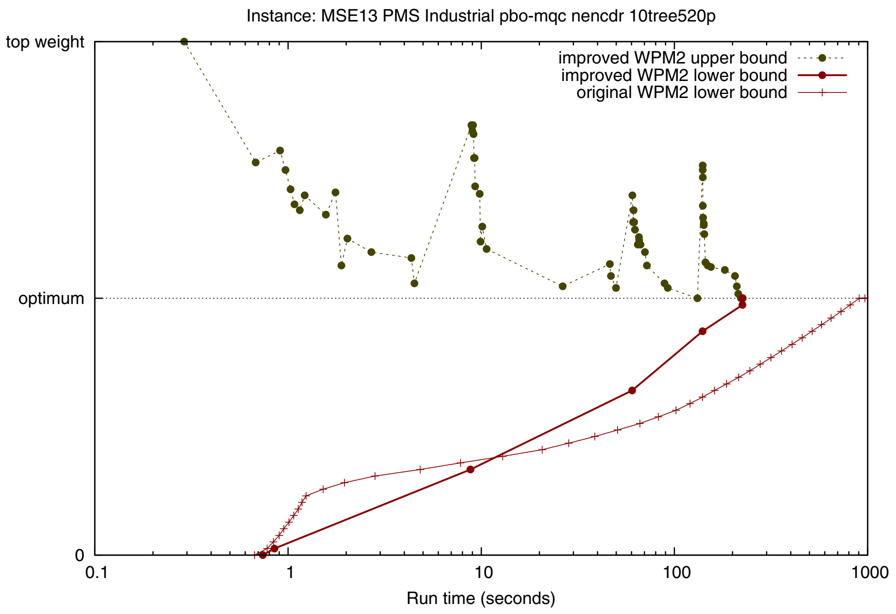


Fig. 1 Upper and lower bounds obtained with the original and the improved WPM2

improved WPM2 algorithm, during the subproblem optimization phase, the upper bound for the particular subproblem is always improved, i.e., it decreases. However, if we extend this assignment to the whole problem this monotonic behavior is not guaranteed. This is why, in Fig. 1, during the subproblem optimization phase, the upper bounds for the whole problem tend to decrease but can also increase.

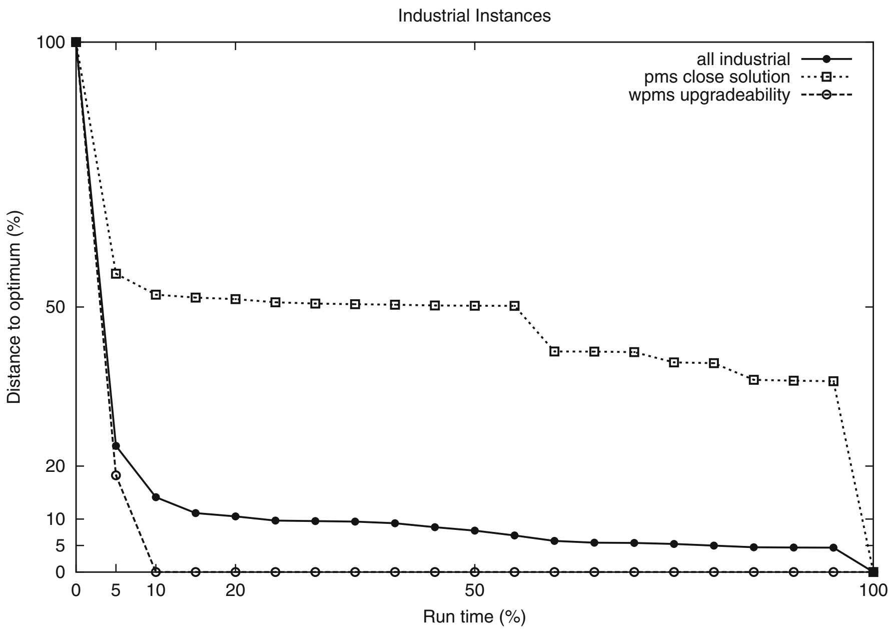
With respect to the quality of these upper bounds, notice that, in less than 5 s, an upper bound very close to the optimum (less than 6 % error) is obtained. Also, an upper bound $\mathcal{I}_A(\varphi)$ equal to the optimal cost $cost(\varphi)$ (0 % error) is obtained in 132 s, earlier than the solution itself (226 s). This is interesting because it means that we can obtain very high quality assignments or even a solution before solving exactly the problem, i.e., certifying that there is no any other assignment with a lower cost.

Finally, we can see that, thanks to subproblem optimization, the improved WPM2 only needs 6 calls (lower bounds updates) on the whole problem, while the original WPM2 needs more than 40.

For more detailed information, we analyze the quality of upper bounds for all the instances of our experimentation in Sect. 6 (Figs. 2 and 3).

We can further exploit the satisfying assignments obtained during the search. In modern SAT solvers, the polarity of the decision variables is chosen according to the most recent polarity they were assigned in a previous partial assignment. This technique is called phase saving (Pipatsrisawat and Darwiche 2007) and its main goal is to avoid redoing work. Notice that, during backtracking, many variable assignments are undone and the suitable polarity needs to be revealed again during search.

In our SAT-based MaxSAT algorithms, we perform independent queries to a SAT solver. Therefore, some suitable information can be lost. For example, in Sect. 5 we discuss how to preserve learned clauses by using the SAT solvers in incremental



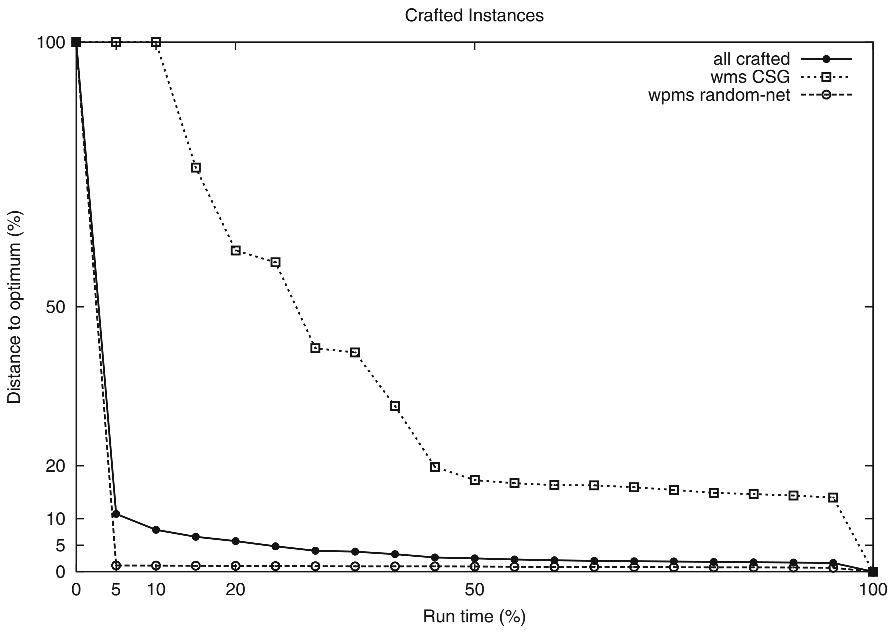
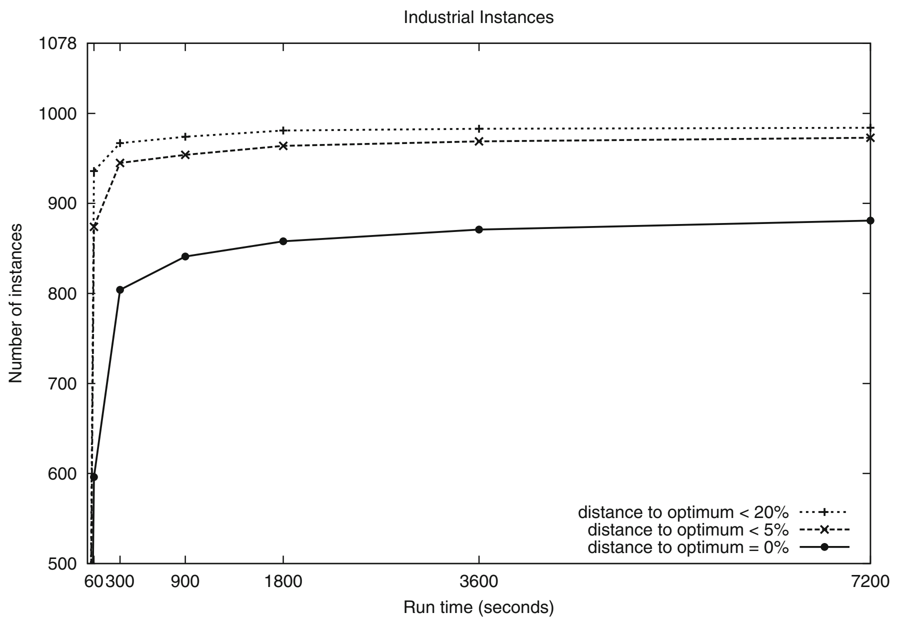


Fig. 2 Upper bound quality relative to the resolution time (at the top industrial instances, at the bottom crafted instances)



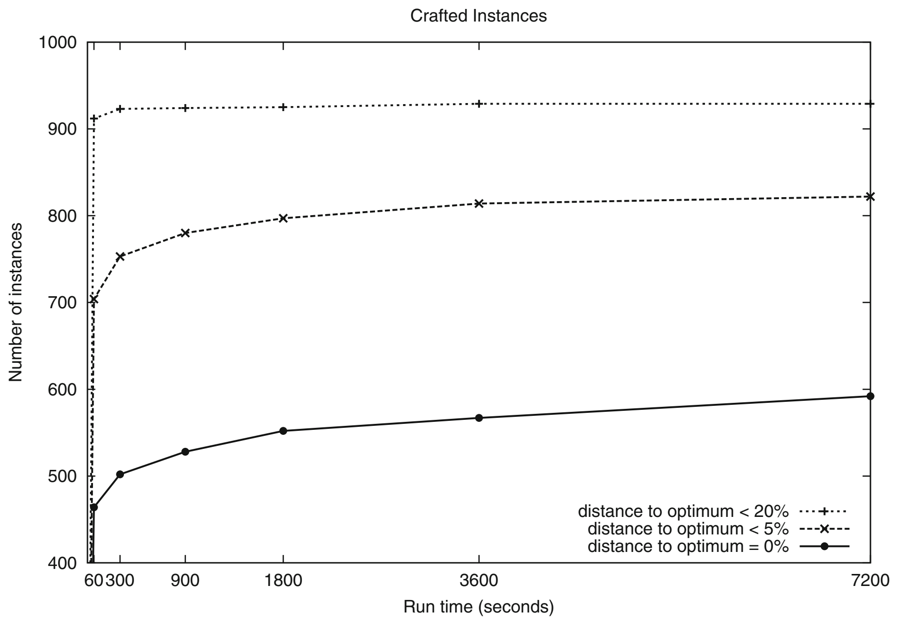


Fig. 3 Number of instances on which a certain upper bound quality is reached (at the top industrial instances, at the bottom crafted instances)

mode. Here, the idea is to use the optimal assignment $\mathcal{I}_A$ for a subproblem $\varphi_{A\cup H}$ (Algorithm 2 line 11) to guide the search in the next call to the SAT solver. Basically, the set β (Algorithm 2 line 12) is updated to contain the unit clauses that represent whether the ith soft clause was satisfied ($\overline{b}_i$) or falsified (b_i) by the $\mathcal{I}_A$ of the most recent subproblem it took part in. We expect assignments $\mathcal{I}_A$ to be more informed as search proceeds and therefore be able to guess the satisfaction status of soft clauses in optimal assignments for φ. In the *sat* function the set β is appended to the set of clauses sent to the SAT solver (new *sat* line 2). Although this gives us extra propagation, it may be the case that our guess is wrong, therefore we iteratively call the SAT solver until no unit clause in β is involved in the unsatisfiable core (new *s*at lines 3, 5 and 6). Notice that the unit clauses that do not appear in any core, do remain in the set β providing us extra propagation power (Algorithm 2 line 4). Therefore, we use the optimal assignments from subproblems to guess the phase of the variables in an optimal assignment to the whole problem φ. In addition, we can also use the satisfying assignments $\mathcal{I}_A$, obtained within the subproblem optimization, to guide the search during this phase.

5 Engineering efficient SMT-based MaxSAT solvers

We have implemented both the last version of the WPM1 algorithm (Ansótegui et al. 2012) and the improved WPM2 algorithm on top of the Yices SMT solver (Dutertre and de Moura 2014).

An SMT formula is a generalization of a Boolean formula in which some propositional variables have been replaced by predicates with predefined assignments from background theories such as, e.g., linear integer arithmetic. For example, an SMT formula can contain clauses like $x_1 \vee x_2 \vee (b_1 + 2 \leq b_1)$ and $(b_1 \geq 2 \cdot b_2 + 3 \cdot b_3)$, where x_1 and x_2 are Boolean variables and b_1, b_2 and b_3 are integer variables. Predicates over non-Boolean variables, such as linear integer inequalities, are evaluated according to the rules of a background theory. Leveraging the advances made in SAT solvers in the last decade, SMT solvers have proved to be competitive with classical decision methods in many areas. Most modern SMT solvers integrate a SAT solver with decision procedures (theory solvers) for sets of literals belonging to each theory. This way, we can hopefully get the best of both worlds: in particular, the efficiency of the SAT solver for the Boolean reasoning and the efficiency of special-purpose algorithms for the theory reasoning.

Another reasonable choice would be to use a PB solver, which can be seen as a particular case of an SMT solver specialized on the theory of PB constraints (Manquinho et al. 2009, 2010). However, if we also want to solve problems modeled with richer formalisms like WCSP, the SMT approach seems a better choice since we can take advantage of a wide range of theories (Ansótegui et al. 2011).

Among the theories considered in the SMT-LIB (Barrett et al. 2010) (SMT Library) we are interested in QF_LIA (Quantifier-Free *Linear Integer Arithmetic*). With the QF_LIA theory we can model the PB constraints that SAT-based MaxSAT algorithms generate during their execution. To this end, the PB variables can be declared as integer variables whose domain is {0, 1}. Therefore, for the SMT-based MaxSAT algorithm, we just need to replace the conversions to *CNF* by the proper linear integer arithmetic predicates. As we can see in Example 11, the SMT-LIB language v2.0 is a standard language with a prefix notation where the operator is placed at the beginning of the predicates.

Example 11 Given the SAT instance of Example 8 Iteration 3, $\varphi^6 = \{(x_1 \vee b_1), (x_2 \vee b_2), (x_3 \vee b_3), (x_4 \vee b_4), (x_5)\ \} \ \cup CNF(\sum x_i \ \le\ 1) \cup CNF(5 \cdot b_1 + 3 \cdot b_3 \ge 3) \cup CNF(5 \cdot b_2 + 3 \cdot b_4 \ge 3) \cup CNF(5 \cdot b_1 + 3 \cdot b_3 \le 3) \cup CNF(5 \cdot b_2 + 3 \cdot b_4 \le 3)$ The SMT instance φ^6 in the SMT-LIB language v2.0 under QF_LIA (Barrett et al. 2010) would be as follows:

```
; Set QF_LIA theory

(set-logic QF_LIA)

; Declaration of variables

(declare-fun x1 () Bool)
(declare-fun x2 () Bool)
(declare-fun x3 () Bool)
(declare-fun x4 () Bool)
(declare-fun x5 () Bool)
(declare-fun b1 () Int)
(declare-fun b2 () Int)
(declare-fun b3 () Int)
(declare-fun b4 () Int)
(declare-fun b5 () Int)

; Bounds for PB variables

(assert (>= b1 0))
(assert (<= b1 1))
(assert (>= b2 0))
(assert (<= b2 1))
(assert (>= b3 0))
(assert (<= b3 1))
(assert (>= b4 0))
(assert (<= b4 1))
(assert (>= b5 0))
(assert (<= b5 1))

; Soft clauses

(assert (or x1 (= b1 1)))
(assert (or x2 (= b2 1)))
(assert (or x3 (= b3 1)))
(assert (or x4 (= b4 1)))
(assert x5)

; Hard clauses

(assert (or (not x1) (not x2))
(assert (or (not x1) (not x3))
(assert (or (not x1) (not x4))
(assert (or (not x1) (not x5))
(assert (or (not x2) (not x3))
(assert (or (not x2) (not x4))
(assert (or (not x2) (not x5))
(assert (or (not x3) (not x4))
(assert (or (not x3) (not x5))
(assert (or (not x4) (not x5))

; PB constraints

(assert (>= (+ (* b1 5) (* b3 3)) 3))
(assert (>= (+ (* b2 5) (* b4 3)) 3))
(assert (<= (+ (* b1 5) (* b3 3)) 3))
(assert (<= (+ (* b2 5) (* b4 3)) 3))

; Check satisfiability

(check-sat)
```

As suggested in Fu and Malik (2006) and Martins et al. (2011), we can preserve some learned lemmas from previous iterations that may help to reduce the search space. In order to do that, we execute the SMT solver in incremental mode. Within this mode, we can call the solve routine and add new clauses (assertions) on demand, while preserving learned lemmas. However, notice that our algorithms delete parts of the formula between iterations. For example, when we have to update the AM set in the WPM2 algorithm (see Sect. 4) by deleting some At-Most constraints. Therefore, we also have to take care of any learned lemma depending on them.

The Yices SMT solver gives the option of marking assertions as *retractable*. If the SMT solver does not support the deletion of assertions but supports the usage of assumptions, we can replace every retractable assertion c, with $a \rightarrow c$, where a is an assumption. Before each call, we activate the assumptions of assertions that have not been retracted by the algorithm. Notice that assertions that do have been retracted will have a pure literal $(\overline{a})$ such that a has not been activated. Therefore, the solver can safely set to false a, deactivating the clause. Moreover, any learned lemma on those assertions will also include $\overline{a}$. For example, Z3 and Mathsat SMT solvers do not allow to delete clauses, but they allow the use of assumptions.

From the point of view of incrementality it is also quite recommendable to reuse as much as possible the PB constraints we modify during the search, since we will also be able to reuse learned clauses depending on them. Pioneering works in this sense can be found in Bofill et al. (2013) and Andres et al. (2012). Our current implementation does not incorporate these complementary improvements, but it would certainly benefit from them.

6 Experimental results

In this section we present an intensive experimental investigation on the industrial and crafted instances of the MaxSAT Evaluation 2013 (MSE13) (Argelich et al. 2006-2004). We provide results for the improved WPM2 SMT-based MaxSAT solver and the best solvers of the MSE13. We run our experiments on a cluster with Intel Xeon CPU E7-8837 @ 2.67GHz processors and a memory limit of 3.5 GB. These are exactly the same specs as in the MSE13.

The instance set of the MSE13 is divided in four categories depending on the variant of the MaxSAT problem: MaxSAT (MS), Partial MaxSAT (PMS), Weighted MaxSAT (WMS) or WPMS. In addition, instances are further divided according to their nature into three subcategories: random, crafted or industrial (we are actually only interested in industrial and crafted). In each subcategory, instances are grouped by families.

We use the same set of instances for experiments on complete and incomplete solvers. The experimental results for complete and incomplete solvers are presented in Sects. 6.1 and 6.2, respectively.

6.1 Complete solvers

In this subsection, we analyze the performance of the improved WPM2 complete solver. First of all, we present the summarized results of the complete solvers at MSE13 in Tables 1 and 2. Second, we analyze the impact of each improvement on the original WPM2 algorithm in Table 3 (full detailed information in Tables 11 and 12). Then, we compare the results of the improved WPM2 solver with the best solvers of the MSE13 in Table 4 (full detailed information in Tables 13 and 14). Finally, in Tables 5 and 6, we further analyze the results of the improved WPM2 solver, discussing how it is able to exploit more efficiently the structure of the instances.

Tables 1 and 2 show the results of the MSE13 (with a timeout of 1800 s). Since families have different numbers of instances, we considered it was more fair to present the solvers ordered by mean family ratio of solved instances. In Table 1 we see the four best performing solvers on each industrial and crafted subcategory. In Table 2 we see the best performing solvers on the whole set of industrial and crafted instances. We have excluded $ISAC+$, since it is a portfolio based solver and our intention here is to compare ground solvers. Notice that $ISAC+$ already includes some of the ground solvers. The ground solvers with the best overall performance were: $wpm2$-13$*$ which actually corresponds to a WPM2 variation ($wpm2_{shua}*$ in Table 3), $maxhs13$ which consists in an hybrid SAT-ILP approach described in Davies and Bacchus (2011), $qms2$ which is based on an upper bound refinement described in Koshimura et al. (2012), $optim\text{-}ni$ and $msunc$ which implement the core-guided binary search algorithm described in Heras et al. (2011) and Morgado et al. (2012), $pwbo2.3$ which takes advantage of parallel processing as described in Martins et al. (2011), Martins et al. (2012), $wpm1$-2013 which is the SMT-based version of the improved WPM1 algorithm described in Ansótegui et al. (2012), ilp which translates WPMS into ILP and applies the MIP solver IBM-CPLEX studio124 as described in Sect. 3, and $wmsz09$ which implements a branch and bound algorithm described in Li et al. (2006), Li et al. (2007), Li et al. (2009). Further information about solvers and authors can be found in Argelich et al. (2006-2004).

We have completed the evaluation of the MSE13 by providing results of some solvers on those categories where originally they did not take part. For example, results

Table 1 MSE13 best solvers ordered by mean ratio of solved instances

		MS		PMS		WMS		WPMS
Random	1.	$msz13f$	1.	$ISAC+$	1.	$ckm\text{-}s$	1.	$ISAC+$
	2.	$ISAC+$	2.	$wmsz09$	2.	$ISAC+$	2.	$wmsz09$
	3.	$ckm\text{-}s$	3.	$wmsz+$	3.	$msz13f$	3.	$wmsz+$
	4.	$wmsz+$	4.	$ckm\text{-}s$	4.	$wmsz+$	4.	$ckm\text{-}s$
Crafted	1.	$ahms$	1.	$ISAC+$	1.	$ISAC+$	1.	$maxhs13$
	2.	$ISAC+$	2.	$qms2\text{-}m$	2.	$wmsz+$	2.	$ISAC+$
	3.	$msz13f$	3.	$qms2\text{-}mt$	3.	$wmsz09$	3.	$ilp13$
	4.	$ckm\text{-}s$	4.	$antom\_s1$	4.	$msz13f$	4.	$wpm1$-13
Industrial	1.	$pmifu$	1.	$ISAC+$	1.	–	1.	$ISAC+$
	2.	$wpm1$-11	2.	$qms2\text{-}mt$	2.	–	2.	$wpm1$-13
	3.	$ISAC+$	3.	$wpm2$-13$*$	3.	–	3.	$wpm2$-13$*$
	4.	$optim\text{-}ni$	4.	$optim\text{-}ni$	4.	–	4.	$msunc$

Table 2 MSE13 best solvers

		Ind. (%)	1078	Cra. (%)	1000	Total (%)	2078
1.	$wpm2$-13$*$	**75.0**	**820**	46.3	521	**61.5**	1341
2.	$maxhs13$	59.7	719	59.9	670	59.8	**1389**
3.	$qms2$	68.7	640	47.3	481	58.6	1121
4.	$optim\text{-}ni$	70.1	671	39.7	435	55.8	1106
5.	$msunc$	71.4	784	37.7	429	55.5	1215
6.	$pwbo2.3$	63.0	686	45.5	521	54.7	1207
7.	$wpm1$-13	65.0	743	40.1	452	53.3	1195
8.	$ilp13$	46.3	575	**61.1**	723	53.3	1298
9.	$wmsz09$	19.5	238	59.2	**745**	38.1	983

Mean ratio and number of solved instances; Best results are in bold

Table 3 Impact of WPM2 improvements, compared on the industrial and crafted instances of MSE13 (number and mean ratio of solved instances)

Solvers	MS	PMS	WMS	WPMS	Ind.	MS	PMS	WMS	WPMS	Cra.	Total
$wpm2$	20	429	–	202	651	12	247	10	39	308	959
	50.6 %	67.9 %	–	49.5 %	61.8 %	14.0 %	59.9 %	6.6 %	19.2 %	31.5 %	47.6 %
$wpm2_s$	**21**	467	–	203	691	12	248	10	98	368	1059
	51.6 %	74.2 %	–	52.5 %	66.9 %	14.0 %	58.1 %	6.6 %	29.9 %	34.0 %	51.4 %
$wpm2_{sh}$	**21**	464	–	269	754	12	248	15	98	373	1127
	51.6 %	73.5 %	–	62.4 %	69.0 %	14.0 %	58.1 %	10.3 %	29.9 %	34.6 %	52.8 %
$wpm2_{shia}$	18	239	–	261	518	**18**	248	**23**	253	542	1060
	48.7 %	37.8 %	–	55.4 %	43.2 %	**16.5 %**	46.8 %	**21.0 %**	55.5 %	40.0 %	41.7 %
$wpm2_{shic}$	19	259	–	267	545	17	257	22	258	554	1099
	49.7 %	40.5 %	–	57.2 %	45.5 %	16.0 %	48.7 %	20.6 %	56.5 %	40.8 %	43.5 %
$wpm2_{shla}$	18	480	–	319	817	12	256	16	199	483	1300
	48.7 %	76.3 %	–	74.0 %	73.7 %	14.0 %	61.6 %	12.0 %	50.1 %	42.0 %	58.8 %
$wpm2_{shlc}$	**21**	486	–	326	833	12	255	16	198	481	1314
	51.6 %	75.6 %	–	75.5 %	73.8 %	14.0 %	61.4 %	12.0 %	49.9 %	41.9 %	58.8 %
$wpm2_{shba}$	17	503	–	326	846	15	261	19	264	559	1405
	47.8 %	80.6 %	–	74.6 %	76.6 %	15.2 %	62.1 %	20.3 %	68.4 %	49.2 %	63.7 %
$wpm2_{shbc}$	20	497	–	**339**	856	14	265	19	263	561	1417
	50.6 %	78.7 %	–	**77.4 %**	76.3 %	14.8 %	62.8 %	20.3 %	67.3 %	49.0 %	63.4 %
$wpm2_{shua}*$	16	502	–	326	844	13	270	18	255	556	1400
	46.8 %	80.7 %	–	75.5 %	76.8 %	14.4 %	63.2 %	18.6 %	67.1 %	48.8 %	63.6 %
$wpm2_{shuc}$	18	513	–	334	865	14	271	18	255	558	1423
	48.7 %	81.9 %	–	76.7 %	78.1 %	14.8 %	63.5 %	18.6 %	66.1 %	48.7 %	64.3 %
$wpm2_{shucg}$	20	524	–	336	880	14	267	18	272	571	1451
	50.6 %	82.5 %	–	77.0 %	78.6 %	14.8 %	63.0 %	18.6 %	68.8 %	49.2 %	64.8 %
$wpm2_{shucgo}$	20	**528**	–	333	**881**	14	**272**	18	**288**	**592**	**1473**
	50.6 %	**82.9 %**	–	76.5 %	**78.9 %**	14.8 %	**63.7 %**	18.6 %	**73.7 %**	**51.0 %**	**65.7 %**

Best results are in bold

of solvers $wpm1$-13 and $wpm2$-13$*$ have been added in the MS category. Results of solver $qms2$ have been also added to MS and WMS categories. We had to change the format of these instances so that $qms2$ could read them. These additional results do not change the overall performance, but they give the full picture.

From Table 2, we emphasize that the solver implementing the variation of the WPM2 algorithm ($wpm2$-13$*$) was already the best in family ratio of solved instances on the whole set of industrial and crafted instances at MSE13. Although it solved fewer instances than $maxhs13$ on the whole set, $wpm2$-13$*$ dominated both in family ratio and number of solved instances on the set of industrial instances.

Table 3 shows our first experiment, where we evaluate the impact of each improvement on the original WPM2 algorithm (with a timeout of 7200 s). All the variations on the WPM2 algorithm are implemented on top of the Yices SMT solver (version

Table 4 $wpm2_{shucgo}$ compared to MSE13 best complete solvers on the industrial and crafted instances of MSE13 (number and mean ratio of solved instances)

Solvers	MS	PMS	WMS	WPMS	Ind.	MS	PMS	WMS	WPMS	Cra.	Total
$wpm2_{shucgo}$	20	528	–	333	**881**	14	272	18	288	592	**1473**
	50.6 %	82.9 %	–	76.5 %	**78.9 %**	14.8 %	63.7 %	18.6 %	73.7 %	51.0 %	**65.7 %**
$wpm2$-13$*$	16	502	–	326	844	13	270	18	255	556	1400
	46.8 %	80.7 %	–	75.5 %	76.8 %	14.4 %	63.2 %	18.6 %	67.1 %	48.8 %	63.6 %
$maxhs13$	7	486	–	259	752	7	304	43	**330**	684	1436
	22.4 %	70.6 %	–	53.8 %	62.7 %	12.0 %	71.1 %	45.3 %	**89.1 %**	62.2 %	62.5 %
$qms2$-$g2$	19	**543**	–	119	681	10	284	30	207	531	1212
	34.0 %	**85.0 %**	–	46.5 %	71.2 %	13.2 %	**73.2 %**	22.2 %	58.2 %	50.3 %	61.4 %
$optim$-ni	38	503	–	165	706	7	261	32	165	465	1171
	83.7 %	80.8 %	–	50.7 %	73.2 %	7.4 %	64.6 %	25.0 %	44.8 %	42.7 %	58.8 %
$ilp13$	7	354	–	259	620	46	**333**	76	311	766	1386
	22.4 %	54.1 %	–	55.3 %	52.0 %	37.7 %	68.4 %	64.7 %	78.0 %	**65.5 %**	58.4 %
$wpm1$-13	21	422	–	**351**	794	12	207	11	301	531	1325
	51.6 %	68.4 %	–	**79.6 %**	70.1 %	14.0 %	48.3 %	7.4 %	74.9 %	43.5 %	57.6 %
$pwbo2.33$	7	445	–	269	721	8	254	12	204	478	1199
	22.4 %	71.9 %	–	58.5 %	64.8 %	12.4 %	65.2 %	14.6 %	66.7 %	48.4 %	57.1 %
$msunc$	26	512	–	270	808	8	249	17	164	438	1246
	56.4 %	77.9 %	–	67.0 %	73.5 %	7.8 %	58.5 %	13.6 %	43.6 %	38.2 %	56.9 %
$wmsz09$	0	200	–	85	285	**156**	318	82	234	**790**	1075
	0.0 %	29.0 %	–	18.9 %	24.2 %	86.4 %	60.4 %	61.2 %	56.1 %	63.6 %	42.7 %
$msz13f$	0	152	–	132	284	**156**	317	**83**	232	788	1072
	0.0 %	22.8 %	–	26.0 %	22.0 %	86.4 %	60.2 %	**66.2 %**	52.7 %	63.4 %	41.5 %
$w/pmifu$	**42**	287	–	261	590	5	61	37	125	228	818
	87.5 %	46.2 %	–	50.9 %	50.5 %	6.6 %	35.1 %	27.1 %	30.9 %	27.8 %	39.8 %
ckm-s	0	119	–	12	131	**156**	293	79	23	551	682
	0.0 %	15.2 %	–	7.2 %	12.0 %	**91.0 %**	55.0 %	46.2 %	7.4 %	45.6 %	27.8 %
$ahms$	0	34	–	10	44	146	224	73	179	622	666
	0.0 %	6.0 %	–	5.5 %	5.4 %	86.5 %	40.3 %	42.5 %	44.7 %	49.6 %	26.2 %

Best results are in bold

1.0.29). The different variations (see Sect. 4) and corresponding implementations are named $wpm2$ with different subindexes. Subindex $_s$ stands for stratified approach and $_h$ for clause hardening. Regarding to how we perform the subproblem optimization, $_i$ stands for ILP translation, $_l$ stands for lower bound refinement based on subset sum, $_u$ for upper bound refinement based on satisfying truth assignment, and $_b$ for binary search. Subindex $_a$ stands for optimizing all the subproblems and $_c$ for optimizing only subproblems that contain already extended clauses. Finally, $_g$ stands for guiding the search with the optimal assignments from subproblems and $_o$ for guiding the search also with the satisfying assignments within the subproblem optimization.

The original $wpm2$ has a performance of 47.6 % (959) family ratio (number) of solved instances. By using the stratified approach explained in Sect. 4.1 ($wpm2_s$)

we solve some additional instances in all categories having the highest increase on WPMS crafted subcategory. Overall, we solve 100 more instances. By applying also the clause hardening explained in Sect. 4.2 ($wpm2_{sh}$) we solve 68 more instances, mainly on WPMS industrial subcategory. This last one, with 52.8 % (1127) family ratio (number) of solved instances, increases in performance by 5.2 % (168) compared to $wpm2$.

Regarding the different variations for optimizing the subproblems (see Sect. 4.3), we can see that optimizing the subproblems through ILP ($wpm2_{shia}$) is not competitive on industrial instances. It solves 236 fewer industrial instances than $wpm2_{sh}$. This is expected since, as we will see in Table 4, ilp-13, the approach based on a full translation to ILP, is also not competitive on industrial instances. On crafted instances it performs similar to the other subproblem optimization variations. It solves 169 more crafted instances than $wpm2_{sh}$, but it is not the best performing variation. Notice, however, that on MS and WPMS industrial subcategories solves more instances than ilp-13.

Optimizing subproblems by refining the lower bound ($wpm2_{shla}$), gives us some additional solved instances in all categories, having the highest increases on WPMS industrial subcategory (50) and on WPMS crafted subcategory (101). It is important to highlight that optimizing subproblems with subset sum, instead of applying the subset sum as in the original WPM2 algorithm, leads to a total increase of 173 solved instances compared to $wpm2_{sh}$.

Optimizing subproblems by refining the upper bound ($wpm2_{shua}*$), gives us an additional boost with respect to $wpm2_{shla}$. We get the highest increases in solved instances, on PMS industrial subcategory (22), and on WPMS crafted subcategory (56). Compared to $wpm2_{sh}$, we have a total increase of 273 solved instances. Notice that this variation is the one that competed in the MSE13 and is referred as $wpm2$-13$*$ in Tables 1, 2 and 4. Optimizing with binary search ($wpm2_{shba}$) has almost the same global performance as $wpm2_{shua}*$.

By optimizing only subproblems that do contain clauses that were already extended in previous iterations, we have an increase in family ratio and number of solved instances on all subproblem optimization variations. The upper bound refinement variation ($wpm2_{shuc}$) is the one with best performance, with 64.3 % (1423) family ratio (number) of solved instances. It increases in performance by 11.5 % (296) compared to $wpm2_{sh}$.

Finally, $wpm2_{shucg}$ is the result of extending the previous best variation ($wpm2_{shuc}$) by guiding the search with the optimal assignments from subproblems (see Sect. 4.4). This way, the number of solved instances increases by 28. By guiding the search also with the satisfying assignments within the subproblem optimization ($wpm2_{shucgo}$), the number of solved instances increases by 22 more, solving 50 more instances than $wpm2_{shuc}$. This last variation ($wpm2_{shucgo}$), with 65.7 % (1473) family ratio (number) of solved instances, increases in performance by 18.1 % (514) compared to $wpm2$. This is in percentage 138.0 % (153.6 %). Actually, if we take into account the timeout of 7200 s used in our experiments, we obtain an overall speed-up of 1573 (three orders of magnitude) with respect to $wpm2$. Basically, we compare the total run time of the solvers on all the instances. Not solved instances are assumed to contribute only with the timeout.

Table 5 $wpm2_{shucgo}$, $certify\text{-}opt$ and $sat4j$ on industrial instances

				$wpm2_{shucgo}$			$certify\text{-}opt$			$sat4j$		
Subcategory	#	$\#H$	$\#S$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$
MS	55	0.0	$1.8{\cdot}10^6$	20	0.8	41.2	0	100	1	0	100	1
PMS	627	$3.3{\cdot}10^5$	$1.3{\cdot}10^4$	528	63.6	87.8	267	100	1	247	100	1
WPMS	396	$4.9{\cdot}10^5$	$8.6{\cdot}10^3$	333	33.9	426	59	100	1	55	100	1
Total industrial	1078	$3.7{\cdot}10^5$	$9.4{\cdot}10^4$	881	49.6	212	326	100	1	302	100	1

Table 6 $wpm2_{shucgo}$, $certify\text{-}opt$ and $sat4j$ on crafted instances

				$wpm2_{shucgo}$			$certify\text{-}opt$			$sat4j$		
Subcategory	#	$\#H$	$\#S$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$
MS	167	0.0	$1.2{\cdot}10^3$	14	96.5	98.8	6	100	1	6	100	1
PMS	377	$2.9{\cdot}10^4$	$3.8{\cdot}10^2$	272	92.0	24.3	222	100	1	224	100	1
WMS	116	0.0	$5.3{\cdot}10^3$	18	57.2	5.5	15	100	1	14	100	1
WPMS	340	$2.9{\cdot}10^4$	$5.8{\cdot}10^2$	288	84.4	6.9	205	100	1	203	100	1
Total crafted	1000	$2.1{\cdot}10^4$	$1.2{\cdot}10^3$	592	86.2	28.7	448	100	1	447	100	1

Table 4 shows our second experiment, where we compare the best variation of the improved WPM2 solver ($wpm2_{shucgo}$) with the best solvers of the MSE13 (with a timeout of 7200 s). In particular, we selected the best ground overall performing solvers presented in Table 2 and the best solvers for each subcategory in Table 1.

We see that $wpm2_{shucgo}$ is the best solver on the industrial set, both in family ratio and number of solved instances. On crafted instances, it is only the fifth in family ratio of solved instances. Although, on crafted instances, ilp is the first in family ratio of solved instances, as we have seen in Table 3, the variation which optimizes the subproblems through an ILP translation ($wpm2_{shia}$) does not improve the performance of the upper bound refinement variation ($wpm2_{shua*}$). We can conclude, however, that $wpm2_{shucgo}$ is the best in family ratio and number of solved instances across all industrial and crafted instances, and so the most robust followed by $maxhs$ and $qms2$.

There can be several explanations for the good performance of $wpm2_{shucgo}$. In the following we extend the study presented in Ansotegui (2013b). One of this explanations is that SAT-based MaxSAT solvers are supposed to take advantage by exploiting the information (learned clauses, At-Least and At-Most constraints, etc) obtained from each SAT instance (φ^k) into which the WPMS instance φ is reformulated.

The conjecture is that this information makes easier the resolution of the SAT instances where k is closer to $cost(\varphi)$. In particular, those ones that are unsatisfiable which tend to be harder to solve. In order to test the plausibility of this conjecture, we introduce a new complete algorithm which takes as input the WPMS formula φ and a cost that we initially set to $cost(\varphi)$. Therefore, this algorithm only needs to certify that the initial cost indeed corresponds to the cost of an optimal assignment. In particular, it just checks that $\varphi^{cost(\varphi)-1}$ is unsatisfiable and $\varphi^{cost(\varphi)}$ is satisfiable. We will refer to this algorithm as $certify\text{-}opt$.

In Tables 5 and 6, we compare $certify\text{-}opt$ with the two basic search schemes of SAT-based MaxSAT solvers: (i) those that focus the search on refining the lower bound, and exploit the information of unsatisfiable cores (solver $wpm2_{shucgo}$) and, (ii) those that focus the search on refining the upper bound, and exploit the information of satisfying assignments (solver $sat4j$ (Berre 2006)). All these approaches were implemented on top of the Yices SMT solver.

We experimented with the whole set of industrial and crafted instances from the MS, PMS, WMS and WPMS categories at the MSE13. Tables 5 and 6 show the results on industrial and crafted instances, respectively. In the tables, # stands for the total number of instances, $\#H$ and $\#S$ stand for the mean number of hard and soft clauses, respectively, and $\#*$ stands for the number of solved instances, within a timeout of 7200 s, for each solver.

The results of our experimentation show that $sat4j$ does not have a better performance than $certify\text{-}opt$. Although $certify\text{-}opt$ solves 25 more instances than $sat4j$, both solvers have almost the same overall performance. This can be explained because the upper bound refinement converges very quickly to the last satisfiable instance $\varphi^{cost(\varphi)}$, but it does not take advantage from any information of the previous satisfiable instances (φ^k with $k \in [cost(\varphi)+1, W(\varphi)]$) in order to solve more efficiently $\varphi^{cost(\varphi)}$ and $\varphi^{cost(\varphi)-1}$. The structure of these instances can be seen in example 7.

Regarding $wpm2_{shucgo}$, it performs much better than $certify\text{-}opt$. For crafted instances, it solves 144 more instances than $certify\text{-}opt$. The difference is more dramatic for industrial instances where it solves 555 more. One of the keys of its success seems to be that it only needs to extend with auxiliary variables those soft clauses that have appeared into an unsatisfiable core. In contrast, $certify\text{-}opt$ or $sat4j$ need to extend all the soft clauses. In the tables, $\%b$ shows the percentage of extended soft clauses during the search. As we can see, both $certify\text{-}opt$ and $sat4j$ always extend 100 % of the soft clauses, while $wpm2_{shucgo}$ only extends a part of them. In particular, on industrial instances, where the difference in performance is higher, it only extend 49.6 % of the soft clauses.

Another key point in the good performance of $wpm2_{shucgo}$ is that, the extended soft clauses in the last query are covered by various At-Most constraints instead of a single and larger one as in $certify\text{-}opt$ or $sat4j$. This was proven to be more efficient in Ansótegui et al. (2009). In the tables, $\#AM$ stands for the number of At-Most constraints added during the search. When we go into detail in Table 5, in the last query of $wpm2_{shucgo}$ on industrial instances, the mean number of At-Most constraints is 212. In contrast, in Table 6, in the last query on crafted instances, the mean number of At-Most constraints is 28.7. This is also consistent with the better performance of $wpm2_{shucgo}$ on industrial instances.

6.2 Incomplete solvers

In this subsection, we analyze the performance of the improved WPM2 incomplete solver. We show the results that it would have obtained on the track for incomplete solvers at the MSE13 in Table 7 (full detailed information in Tables 15 and 16). We

Table 7 $wpm2_{shucgo}$ incomplete compared to MSE13 incomplete solvers on the industrial and crafted instances of MSE13 (number and mean ratio of solved instances)

Solvers	MS	PMS	WMS	WPMS	Ind.	MS	PMS	WMS	WPMS	Cra.	Total
$wpm2_{shucgo}$	18	**511**	–	**336**	**865**	15	260	20	224	519	**1384**
	32.0 %	**80.0 %**	–	**77.0 %**	**76.0 %**	15.0 %	61.0 %	19.0 %	62.0 %	47.0 %	**62.0 %**
ccls	**46**	501	–	198	745	129	205	83	93	510	1255
	91.0 %	79.0 %	–	52.0 %	73.0 %	76.0 %	37.0 %	60.0 %	17.0 %	42.0 %	58.0 %
optim	2	60	–	13	75	**167**	**303**	**115**	**257**	**842**	917
	1.0 %	10.0 %	–	8.0 %	9.0 %	**100.0 %**	**67.0 %**	**99.0 %**	**60.0 %**	**76.0 %**	40.0 %
ira-nov	3	103	–	63	169	9	225	30	167	431	600
	2.0 %	20.0 %	–	28.0 %	21.0 %	8.0 %	61.0 %	20.0 %	45.0 %	40.0 %	29.0 %

Best results are in bold

also discuss the quality of the upper bounds obtained during its execution on industrial and crafted instances in Figs. 2 and 3.

Table 7 shows our first experiment, where we compare the incomplete solver based on the improved WPM2 algorithm ($wpm2_{shucgo}$), with the best incomplete solvers of the MSE13. Results are presented following the same classification criteria as in the MSE13. For each instance, it is computed which are the solvers that obtain the best upper bound within 300 s (5 min). For each solver and subcategory we present the number of instances where the solver reported the best upper bound and the mean family ratio according to this number.

We can see that $wpm2_{shucgo}$ dominates on industrial instances, being the one that reaches the best upper bound 865 times. This is 120 more times than *ccls*, the second one. On crafted instances, both are dominated by *optim*, that in contrast performs not well on industrial instances. When we consider the whole set of industrial and crafted instances, $wpm2_{shucgo}$ is the best one 1384 times, 129 more times than the second one *ccls*.

In the following, we analyze the quality of upper bounds provided by $wpm2_{shucgo}$ on the whole set of industrial and crafted instances. In Fig. 2, we show the mean upper bound quality obtained during the resolution of the instances. Those instances not solved in 7200 s or solved in less than 60 s are discarded. Therefore, only 358 industrial and 129 crafted instances are taken into account. In x-axis we have the relative elapsed running time (100 % corresponds to the total time to solve the instance), and in y-axis we have the relative distance of the upper bounds to the optimum (an upper bound equal to the top weight or to the optimum, has a relative distance of 100 or 0 %, respectively). We refer to the relative distance to the optimum as the *error* in an upper bound (a small error means a high quality).

In the graphic at the top of Fig. 2, we have the experiments on the whole set of industrial instances (MS, PMS and WPMS categories), and on the two industrial families, with the best (WPMS *upgradeability* family) and the worst (PMS *close solutions* family) mean upper bound quality. In one third of the resolution time, the mean error in the upper bounds on the whole set of industrial instances is less than 10 % and in two thirds it is less than 6 %. In particular, for the WPMS *upgradeability* family, in

10 % of the resolution time the mean error is less than 1 %. Also, on some instances (PMS *pbo-mcq* family) the optimum (0 % error) is reached in one second while more than 1000 s are needed to certify it. On the other hand, for the PMS *close solution* family, in 90 % of the resolution time, the mean error is about 37 %.

In the graphic at the bottom of Fig. 2, we have the experiments on the whole set of crafted instances (MS, PMS, WMS and WPMS categories), and on the two crafted families, with the best (WPMS *random-net* family) and the worst (WMS *CSG* family) mean upper bound quality. In one third of the resolution time, the mean error in the upper bounds on the whole set of crafted instances is less than 4 % and in two thirds it is less than 2 %. In particular, for the WPMS *random-net* family, in 5 % of the resolution time the mean error is less than 1 %. On the other hand, for the WMS *CSG* family, in 10 % of the resolution time the mean error is 100 and in 90 % of the resolution time it is about 14 %.

It may seem that we get better results on crafted instances, however we should take into account that, the number of crafted instances solved by $wpm2_{shucgo}$ within 60 and 7200 run time seconds, is lower than the number of industrial instances. For some instances unsolved by $wpm2_{shucgo}$, we can consult the optimal cost found by other MaxSAT solvers (none dominates completely at the MSE13). This allows us to know which is the quality achieved by $wpm2_{shucgo}$ even if it was not able to solve the instance exactly.

In Fig. 3, we experimented with the industrial and crafted instances of MSE13 where any of the solvers in Table 4 was able to find the optimum.[2] We show the number of industrial and crafted instances, where $wpm2_{shucgo}$ reached an upper bound with a relative distance to the optimum (error) of less than 20, 5 and 0 % in a given elapsed run time. In x-axis we show the elapsed time from 0 to 7200 s, and in y-axis we show the number of instances.

In the graphic at the top of Fig. 3, we have the experiments on the industrial instances. We can see that, a high quality of upper bound is reached on a great number of instances in a relative short run time. From 20 to 5 % error, there is almost no difference. In 60 s, an upper bound with an error of less than 5 % is reached on 874 out of 1012 instances. In 300 s it is reached on 945 instances and in 7200 s on 973 instances. With respect to a 0 % error in the upper bound (optimum is not necessarily certified), in 60 s it is reached on 596 instances, in 300 s on 804 instances and in 7200 s on 881 instances.

In the graphic at the bottom of Fig. 3, we have the experiments on the crafted instances. Compared to what happens on the industrial set, there is a greater difference in number of instances, depending on the maximum error that we consider. We can see that, an upper bound with an error of less than 20 % is reached on 912 out of 952 instances in 60 s. However, the number of instances with less than a 5 % error in the upper bound is significantly lower, 704 instances in 60 s, 753 instances in 300 s and 822 instances in 7200 s. With respect to a 0 % error in the upper bound, the difference is even more important. In 60 s it is reached only on 464 instances, in 300 s only on 502 instances and in 7200 s only on 592 instances.

[2] There are only 66 industrial and 48 crafted instances for which was not found.

As a concluding remark, in 300 s (the timeout of the track for incomplete solvers at MSE13), $wpm2_{shucgo}$ has reached an upper bound with an error of less than 5 % on 945 out of 1012 industrial instances. This is consistent with the results in Table 7 where we have shown that $wpm2_{shucgo}$ has the best performance on industrial instances.

6.3 Results at MaxSAT evaluation 2014

In the previous subsections, we have analyzed and described in detail the impact of every improvement incorporated in $wpm2$. Here, we study the performance of the $wpm2014$ solver that took part in MSE14. This solver behaves exactly as $wpm2_{shuc}$ for (Partial) MaxSAT instances, and includes some efficiencies for Weighted (Partial) MaxSAT instances. In particular, we show that although this solver is not as competitive as the newest complete solvers, its incomplete version (as described in this article) ranked the first for the incomplete track.

The sets of instances at MSE14 are almost the same as the ones at MSE13. The main difference is that WMS families were integrated into the WPMS set. The classification criteria remains the same.

Table 8 summarizes the results for the best complete solvers on the whole set of industrial and crafted instances according to MSE14. We have excluded the portfolio

Table 8 Best complete solvers on the industrial and crafted instances at MSE14 (number and mean ratio of solved instances)

Solvers	MS	PMS	WPMS	Ind.	MS	PMS	WPMS	Cra.	Total
maxhs	27	417	280	724	7	328	219	554	1278
	73.0 %	70.5 %	52.6 %	62.6 %	11.8 %	69.2 %	**75.8 %**	**62.2 %**	**64.2 %**
eva500a	41	472	**368**	**881**	9	302	149	460	**1341**
	86.5 %	79.7 %	72.8 %	**78.4 %**	8.5 %	71.3 %	47.8 %	47.8 %	62.8 %
mscg	40	468	363	871	5	310	127	442	1313
	85.5 %	80.0 %	70.4 %	77.9 %	4.3 %	67.6 %	42.8 %	43.3 %	60.4 %
qms-g3	14	454	303	771	11	318	181	510	1281
	29.0 %	78.5 %	67.3 %	72.6 %	9.0 %	**73.4 %**	52.5 %	50.9 %	61.6 %
wpm2014	29	428	359	816	12	297	151	460	1276
	75.0 %	75.5 %	**73.9 %**	75.1 %	9.3 %	69.7 %	43.4 %	45.2 %	59.9 %
open-wbo	**42**	**473**	315	830	11	317	130	458	1288
	87.5 %	**81.1 %**	59.5 %	76.1 %	9.0 %	73.4 %	37.0 %	42.9 %	59.3 %
ilp	1	265	249	515	30	339	**224**	**593**	1018
	0.5 %	40.0 %	45.9 %	39.0 %	20.5 %	62.6 %	75.2 %	61.4 %	50.3 %
scip-ms	0	211	242	453	24	**342**	200	566	1019
	0.0 %	31.5 %	44.8 %	32.9 %	12.5 %	63.4 %	70.4 %	57.8 %	45.5 %
ahmaxsat-ls	0	32	25	57	**156**	294	128	578	635
	0.0 %	5.5 %	7.5 %	5.7 %	**60.5 %**	48.0 %	32.1 %	42.1 %	24.1 %

Best results are in bold

$ISAC+$, since our main aim here is to compare algorithms and ground solvers. We have also included *ilp*, *scip-ms* and *ahmaxsat-ls*, which win in mean family ratio or number of solved instances on some crafted categories. On crafted instances, the best solver in mean family ratio (number) of solved instances is *maxhs* (*ilp*), and on the whole set, it is *maxhs* (*eva*500*a*).

On industrial instances, we can see that *wpm*2014 is the fourth one. The best three solvers on industrial instances were *eva*500*a* (Narodytska and Bacchus 2014), *mscg* (Morgado et al. 2014) and *open-wbo* (Martins et al. 2014). For *open-wbo*, we selected the best version for each category. To our best knowledge *eva*500*a* automatically detects the category and applies a predefined user parametrization. These three solvers are SAT-based MaxSAT solvers too. Without going into detail, we could say that the approaches of *eva*500*a* and *mscg* allow them to generate simpler PB constraints. Moreover, all three new solvers build incrementally these PB constraints, instead of generating them from scratch. In the case of *open-wbo*, PB constraints are explicitly built incrementally. In the case of *eva*500*a* and *mscg* this incrementality comes naturally as a result of the nature of the algorithm. These improvements are complementary to the approach of *wpm*2014 and therefore they could be incorporated. We can yet see that *wpm*2014 achieves the best ratio in WPMS instances.

Table 9 summarizes the results for the best incomplete solvers on the whole set of industrial and crafted instances according to MSE14. Clearly, *wpm*2014 dominates on industrial instances, and it is the best overall solver on industrial and crafted instances. For *optimax* (implementing the BCD algorithm (Morgado et al. 2012)), the second best solver for industrial instances, we also selected the best version for each category.

Since we are showing a partial order, we have also included the solvers that take part in all categories *dist*, *ccls*2014, *ccmpa* and *ahmaxsat-ls*, from which *dist* and

Table 9 Best incomplete solvers on the industrial and crafted instances at MSE13 (number and mean ratio of solved instances)

Solvers	MS	PMS	WPMS	Ind.	MS	PMS	WPMS	Cra.	Total
*wpm*2014	30	**407**	**365**	**802**	15	301	180	496	**1298**
	75.5 %	**71.9 %**	**73.2 %**	**73.2 %**	7.0 %	67.4 %	**51.3 %**	51.3 %	**62.1 %**
*optimax*2	**33**	354	247	634	15	295	149	459	1093
	78.5 %	59.7 %	59.8 %	59.8 %	7.0 %	**68.8 %**	47.2 %	47.2 %	53.4 %
dist	0	177	38	215	171	**354**	**186**	**711**	926
	0.0 %	31.7 %	10.8 %	24.5 %	82.7 %	66.3 %	50.8 %	**61.3 %**	43.2 %
*ccls*2014	0	61	39	100	**176**	299	167	642	742
	0.0 %	10.5 %	15.1 %	11.0 %	**97.2 %**	58.4 %	48.7 %	60.5 %	36.1 %
ccmpa	5	91	52	148	173	281	154	608	756
	4.5 %	17.5 %	22.4 %	17.9 %	87.5 %	54.5 %	38.8 %	52.4 %	35.4 %
*sat*4*j*	5	117	66	188	2	200	82	284	472
	20.5 %	23.6 %	19.3 %	22.3 %	0.5 %	40.3 %	28.6 %	27.1 %	24.7 %

Best results are in bold

Table 10 Comparison as incomplete solvers of *wpm*2014, *open-wbo* and *qms*

	*wpm*2014	*open-wbo*	*qms*
*wpm*2014		**939**	**850**
		(**51 498 390**)	(**52** 423 **375**)
open-wbo	857		**864**
	(43 473 341)		(**45** 458 **361**)
qms	820	827	
	(13 **528** 279)	(19 **522** 286)	

Best results are in bold

*ccls*2014 win in mean family ratio or number of solved instances on some crafted categories.

Finally, we have decided to extend the results of the incomplete track of the MaxSAT Evaluation by comparing to *wpm*2014 the best complete solvers than can provide upper bounds but did not take part at the incomplete track: *qms*2 and *open*-*wbo*. We performed the comparison on industrial instances, where all these solvers were competitive.

In Table 10, we present the dominance relation between pairs of solvers on the the total (MS PMS WPMS) set of industrial instances. For example, *wpm*2014 (*open-wbo*) is able to obtain a better or equal upper bound than *open*-*wbo* (*wpm*2014) on 939 (857) industrial instances of which 51 (43) are MS, 498 (473) are PMS and 390 (341) are WPMS.

On the whole set of industrial instances *wpm*2014 outperforms both *open*-*wbo* and *qms*2. Even though *wpm*2014 was not the best complete solver at MSE14, it dominates the other solvers as an incomplete approach. For PMS, we can see that *qms*2 is the best performing approach, but for MS and WPMS, *wpm*2014 performs better. Notice that *qms*2 is an efficient implementation of a SAT-based MaxSAT algorithm that mainly follows and upper-bound refinement strategy but unlike *wpm*2014 needs to place PB constraints on the whole set of soft clauses. So, if the subproblems represent most of the original instance, *qms*2 should be better, but if the subproblems cover only a small fraction, *wpm*2014 has advantages even with a less efficient approach for subproblems. Precisely, thanks to the insights of Table 5 in Sect. 6.1, we know that for PMS instances, $wpm2_{shucgo}$ (*wpm*2014) covers in average a 63.6 % of the soft clauses, while for WPMS, it covers in average only a 33.9 %. This suggests that incorporating the efficiencies of *qms*2 into *wpm*2014 will certainly improve its performance on PMS instances and perhaps on WPMS instances too.

These results confirm that by applying the subproblem optimization technique and exploiting the satisfying assignments from subproblems, a complete MaxSAT solver can be used as an effective incomplete MaxSAT solver.

7 Conclusions and future work

True innovation in heuristic-search research is not achieved from yet another method that performs better than its competitors if there is no understanding as to why the method performs well (Sörensen 2015). Following this principle, we have provided a detailed analysis on how the improved WPM2 solver has experienced a significant boost on solving industrial instances, which is our ultimate goal.

In particular, the subproblem optimization approach and the exploitation of the satisfying assignments from subproblems seem to have the most clear impact on efficiency. Therefore, solving incrementally an optimization problem by solving some of its subproblems is a promising avenue. This way we can focus on a reduced portion of the instance, allowing us to use less complex PB constraints to solve the whole problem.

We have seen that SAT-based MaxSAT solvers are able to exploit the information derived from refining lower bounds and upper bounds. We have confirmed that this is not only crucial to locate quickly better upper bounds, also to certify efficiently that a given upper bound is the optimum.

Moreover, from a more practical point of view, we know that although many NP-hard problems can not be solved exactly, in industry they are mostly focused on obtaining better upper bounds. Therefore, completeness is not always a mandatory requirement to guarantee practical success. In this sense, we have shown how we can turn a complete solver into an efficient incomplete solver by extending the satisfying assignments from the subproblem optimization phases to the whole problem.

As shown in the experimental evaluation, the incomplete version of the improved WPM2 solver would have dominated on industrial instances the track for incomplete solvers at the MSE13. Furthermore, the solver $wpm2014$, which is just a more efficient implementation of the improved WPM2, was the best performing solver on industrial instances on the track for incomplete solvers at the MSE14.

As future work, we will study how to improve the interaction with the optimization of the subproblems. A portfolio that selects the most suitable optimization approach depending on the structure of the subproblems seems another way of achieving additional speed-ups.

From the point of view managing even more efficiently PB constraints, following recent works (Narodytska and Bacchus 2014; Morgado et al. 2014; Martins et al. 2014), it is also quite recommendable to use less complex PB constraints and reuse as much as possible of them when they are modified.

Finally, we have also shown that the SMT technology is an underlying efficient technology for solving the MaxSAT problem. A positive side effect is that our algorithm can be naturally extended to solve the MaxSMT problem.

Acknowledgments Research partially supported by the Ministerio de Economía y Competividad research project TASSAT2: TIN2013-48031-C4-4-P and Google Faculty Research Award program.

Appendix

See Tables 11, 12, 13, 14, 15 and 16.

Table 11 Impact of WPM2 improvements, compared on the industrial instances of MSE13 (number and mean ratio of solved instances)

Family	#	$wpm2$	$wpm2_s$	$wpm2_{sh}$	$wpm2_{shia}$	$wpm2_{shic}$	$wpm2_{shla}$	$wpm2_{shlc}$	$wpm2_{shba}$	$wpm2_{shbc}$	$wpm2_{shua}*$	$wpm2_{shuc}$	$wpm2_{shucg}$	$wpm2_{shucgo}$
MS														
cir-dp	3	402(2)	75.3(2)	70.4(2)	77.7(2)	**55.3(2)**	319(2)	70.1(2)	446(2)	63.2(2)	489(2)	58.5(2)	251(2)	248(2)
sean-s	52	496(18)	681(19)	690(19)	625(16)	397(17)	592(16)	**548(19)**	435(15)	649(18)	356(14)	397(16)	591(18)	493(18)
Total	55	20	**21**	**21**	18	19	18	**21**	17	20	16	18	20	20
		50.6 %	**51.6 %**	**51.6 %**	48.7 %	49.7 %	48.7 %	**51.6 %**	47.8 %	50.6 %	46.8 %	48.7 %	50.6 %	50.6 %
PMS														
aes	7	0.0(0)	0.0(0)	0.0(0)	2857(1)	92.8(1)	0.0(0)	0.0(0)	**11.6(1)**	0.0(0)	524(1)	1975(1)	0.0(0)	0.0(0)
bcp-fir	50	58.5(49)	**19.2(49)**	21.2(49)	103(49)	92.6(49)	131(49)	36.6(49)	50.9(49)	48.2(48)	19.8(49)	30.6(49)	107(49)	52.6(49)
bcp-hysi	17	325(13)	584(13)	607(13)	635(7)	590(7)	175(13)	524(13)	602(16)	646(14)	156(16)	**99.7(16)**	148(16)	120(16)
bcp-hysu	38	229(19)	513(17)	524(17)	76.3(1)	13.8(1)	679(20)	153(18)	398(28)	185(25)	**256(34)**	453(33)	405(33)	511(34)
bcp-msp	50	897(19)	1100(21)	841(20)	570(17)	880(19)	739(22)	322(22)	927(26)	551(26)	804(26)	902(29)	671(28)	**591(29)**
bcp-mtg	40	1538(21)	1607(28)	1221(26)	816(22)	742(20)	916(36)	1487(31)	810(38)	977(33)	**441(39)**	844(36)	809(36)	746(36)
bcp-syn	50	323(24)	417(23)	664(24)	85.4(24)	128(24)	146(22)	834(25)	337(26)	501(25)	121(24)	320(25)	**591(30)**	121(28)
cir-tc	4	273(3)	1064(4)	816(4)	2537(1)	447(1)	163(4)	300(3)	114(4)	129(4)	94.4(4)	95.2(4)	99.1(4)	**90.3(4)**
clo-s	50	489(34)	819(33)	833(33)	4246(3)	748(23)	1049(29)	**771(39)**	1358(22)	1032(32)	1310(17)	1508(27)	1426(34)	629(35)
des	50	405(19)	601(23)	678(23)	0.0(0)	34.7(1)	918(24)	782(23)	1004(28)	598(22)	843(26)	833(23)	837(24)	**1188(29)**
hap-a	6	**1.7(5)**	5.2(5)	5.2(5)	275(5)	182(5)	39.0(5)	7.8(5)	42.0(5)	12.4(5)	51.3(5)	909(5)	33.2(5)	11.7(5)
pac-pms	40	49.2(12)	21.9(38)	21.6(38)	854(40)	62.2(40)	291(36)	29.8(40)	286(36)	**29.0(40)**	287(36)	30.1(40)	40.4(40)	51.0(40)
pbo-mne	50	611(50)	640(50)	644(50)	990(1)	0.0(0)	94.2(50)	120(50)	116(50)	124(50)	279(50)	273(50)	104(50)	**51.2(50)**
pbo-mnl	50	276(50)	284(50)	309(50)	5926(1)	6360(2)	17.4(50)	26.4(50)	65.9(50)	64.5(50)	153(50)	141(50)	47.5(50)	**15.7(50)**
pbo-rou	15	**0.4(15)**	2.2(15)	2.2(15)	1278(6)	677(8)	6.1(15)	3.8(15)	6.7(15)	4.8(15)	6.3(15)	5.4(15)	4.3(15)	4.3(15)
pro-ins	12	2769(11)	2753(11)	2456(10)	0.7(1)	0.6(1)	500(12)	470(12)	251(12)	350(12)	351(12)	236(12)	241(12)	**230(12)**
tpr-Mp	48	476(36)	452(37)	432(37)	294(12)	271(13)	567(43)	505(41)	327(47)	706(46)	**189(48)**	238(48)	300(48)	363(46)
tpr-Op	50	433(49)	468(50)	485(50)	1463(48)	928(44)	111(50)	166(50)	12.9(50)	12.1(50)	9.1(50)	9.5(50)	**7.7(50)**	94.4(50)

Table 11 continued

Family	#	$wpm2$	$wpm2_s$	$wpm2_{sh}$	$wpm2_{shia}$	$wpm2_{shic}$	$wpm2_{shla}$	$wpm2_{shlc}$	$wpm2_{shba}$	$wpm2_{shbc}$	$wpm2_{shua}*$	$wpm2_{shuc}$	$wpm2_{shucg}$	$wpm2_{shucgo}$
Total	627	429	467	464	239	259	480	486	503	497	502	513	524	**528**
		67.9 %	74.2 %	73.5 %	37.8 %	40.5 %	76.3 %	75.6 %	80.6 %	78.7 %	80.7 %	81.9 %	82.5 %	**82.9 %**
WPMS														
hap-ped	100	57.0(24)	201(25)	201(90)	337(51)	470(57)	135(92)	80.0(91)	76.0(95)	156(96)	142(98)	**240(99)**	152(97)	196(98)
pac-wpms	99	143(28)	163(23)	165(23)	351(72)	192(70)	688(62)	367(69)	422(67)	**312(77)**	347(62)	224(69)	200(73)	401(69)
pre-pla	29	35.9(28)	158(27)	151(27)	404(13)	464(13)	34.3(29)	26.9(29)	15.0(29)	14.2(29)	**13.2(29)**	17.9(29)	13.4(29)	14.6(29)
time	26	706(8)	597(8)	560(8)	0.0(0)	476(1)	**448(9)**	820(9)	730(8)	1205(9)	1045(9)	1407(9)	1055(9)	552(9)
upg-pro	100	16.4(100)	14.4(100)	**11.6(100)**	214(100)	57.5(100)	301(100)	79.2(100)	295(100)	77.2(100)	297(100)	78.1(100)	104(100)	105(100)
wcsp-s5d	21	1032(8)	232(10)	254(12)	**135(15)**	390(15)	433(14)	364(14)	318(14)	322(14)	122(14)	8.4(14)	34.2(14)	17.4(14)
wcsp-s5l	21	0.3(6)	408(10)	125(9)	572(10)	364(11)	58.4(13)	603(14)	50.5(13)	135(14)	17.0(14)	23.3(14)	29.8(14)	**11.7(14)**
Total	396	202	203	269	261	267	319	326	326	**339**	326	334	336	333
		49.5 %	52.5 %	62.4 %	55.4 %	57.2 %	74.0 %	75.5 %	74.6 %	**77.4 %**	75.5 %	76.7 %	77.0 %	76.5 %
Total Ind.	1078	651	691	754	518	545	817	833	846	856	844	865	880	**881**
		61.8 %	66.9 %	69.0 %	43.2 %	45.5 %	73.7 %	73.8 %	76.6 %	76.3 %	76.8 %	78.1 %	78.7 %	**78.9 %**

Best results are in bold

Table 12 Impact of WPM2 improvements, compared on the crafted instances of MSE13 (number and mean ratio of solved instances)

Family	#	$wpm2$	$wpm2_s$	$wpm2_{sh}$	$wpm2_{shia}$	$wpm2_{shic}$	$wpm2_{shla}$	$wpm2_{shlc}$	$wpm2_{shba}$	$wpm2_{shbc}$	$wpm2_{shua*}$	$wpm2_{shuc}$	$wpm2_{shucg}$	$wpm2_{shucgo}$
MS														
bip-mc.7	50	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
bip-mc.8	50	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
mc-dm	62	844(10)	797(10)	992(10)	**1390(16)**	1138(15)	688(10)	498(10)	1005(13)	691(12)	359(11)	567(12)	320(11)	854(12)
mc-sg	5	**0.2(2)**	0.2(2)	0.2(2)	1.3(2)	1.7(2)	0.8(2)	0.2(2)	39.4(2)	49.9(2)	14.4(2)	2.6(2)	2.4(2)	17.7(2)
Total	167	12	12	12	**18**	17	12	12	15	14	13	14	13	14
		14.0 %	14.0 %	14.0 %	**16.5 %**	16.0 %	14.0 %	14.0 %	15.2 %	14.8 %	14.4 %	14.8 %	14.4 %	14.8 %
PMS														
frb	25	0.0(0)	0.0(0)	0.0(0)	**1222(1)**	3656(1)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
job-sh	3	60.6(3)	52.3(3)	65.3(3)	0.0(0)	0.0(0)	54.0(3)	47.2(3)	**41.6(3)**	51.9(3)	55.8(3)	67.8(3)	62.0(3)	65.9(3)
mcq-ran	96	445(76)	450(76)	459(76)	**365(93)**	305(90)	517(78)	497(78)	362(82)	484(84)	551(90)	750(90)	377(87)	569(90)
mcq-str	62	727(21)	569(21)	577(21)	**644(32)**	710(28)	806(24)	755(23)	810(26)	1087(27)	816(27)	1008(27)	680(26)	752(28)
mo-3s	80	15.7(80)	16.7(80)	17.4(80)	81.8(80)	33.3(80)	8.8(80)	9.9(80)	5.0(80)	5.1(80)	6.7(80)	6.6(80)	5.8(80)	**4.4(80)**
mo-str	60	145(58)	81.5(59)	83.9(59)	1300(31)	549(47)	30.1(60)	15.4(60)	45.5(60)	12.9(60)	45.0(60)	**9.0(60)**	14.0(60)	19.2(60)
mine-kbt	42	1325(4)	1872(5)	1623(5)	**1065(7)**	2398(7)	2241(6)	2512(6)	1356(5)	1498(6)	1577(5)	847(6)	1276(6)	1373(6)
pse-ml	4	106(4)	97.9(4)	94.8(4)	26.3(4)	**25.3(4)**	63.1(4)	91.0(4)	28.9(4)	28.3(4)	34.0(4)	47.4(4)	32.5(4)	38.1(4)
sch	5	5652(1)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	3203(1)	3845(1)	1439(1)	3652(1)	804(1)	1210(1)	1276(1)	**688(1)**
Total	377	247	248	248	248	257	256	255	261	265	270	271	267	**272**
		59.9 %	58.1 %	58.1 %	46.8 %	48.7 %	61.6 %	61.4 %	62.1 %	62.8 %	63.2 %	63.5 %	63.0 %	**63.7 %**

Table 12 continued

Family	#	$wpm2$	$wpm2_s$	$wpm2_{sh}$	$wpm2_{shia}$	$wpm2_{shic}$	$wpm2_{shla}$	$wpm2_{shlc}$	$wpm2_{shba}$	$wpm2_{shbc}$	$wpm2_{shua}*$	$wpm2_{shuc}$	$wpm2_{shucg}$	$wpm2_{shucgo}$
WMS														
frb	34	128(4)	129(4)	1439(9)	611(10)	**84.7(10)**	615(9)	769(9)	336(9)	253(9)	182(9)	195(9)	87.2(9)	125(9)
ram	15	0.7(1)	0.4(1)	0.4(1)	474(3)	477(3)	684(2)	556(2)	**404(4)**	624(4)	93.0(3)	84.2(3)	1181(4)	117(3)
wmc-dm	62	6.7(5)	63.0(5)	61.6(5)	**1469(9)**	466(8)	1.4(5)	1.4(5)	2.2(5)	5.1(5)	18.3(5)	6.7(5)	11.4(5)	2.3(5)
wmc-sg	5	0.0(0)	0.0(0)	0.0(0)	**44.5(1)**	45.4(1)	0.0(0)	0.0(0)	216(1)	154(1)	348(1)	435(1)	364(1)	409(1)
Total	116	10	10	15	**23**	22	16	16	19	19	18	18	19	18
		6.6 %	6.6 %	10.3 %	**21.0 %**	20.6 %	12.0 %	12.0 %	20.3 %	20.3 %	18.6 %	18.6 %	20.3 %	18.6 %
WPMS														
CSG	10	866(5)	717(5)	731(5)	69.7(4)	168(4)	34.1(6)	56.8(6)	62.2(10)	66.1(10)	71.2(10)	66.8(10)	**39.3(10)**	45.3(10)
auc-pat	86	0.0(0)	344(1)	349(1)	871(76)	808(75)	525(33)	688(33)	952(75)	957(74)	**276(82)**	426(82)	366(82)	316(82)
auc-sch	84	0.0(0)	140(50)	144(50)	2.7(84)	1.8(84)	71.3(84)	71.8(84)	1.6(84)	2.6(84)	1.0(84)	1.2(84)	**0.9(84)**	0.9(84)
mine-pl	56	420(30)	139(38)	144(38)	696(50)	485(50)	2.6(56)	3.2(56)	0.7(56)	0.7(56)	0.8(56)	0.7(56)	**0.6(56)**	0.7(56)
mine-wa	18	2.9(1)	3.6(1)	3.6(1)	0.1(1)	0.0(1)	0.1(1)	0.1(1)	0.0(1)	0.0(1)	2802(2)	1136(2)	0.1(1)	**2304(4)**
pse-ml	12	371(3)	1488(3)	1570(3)	3.9(2)	2.8(2)	12.9(3)	14.3(3)	767(5)	409(4)	**756(5)**	287(4)	242(4)	260(4)
ran-net	74	0.0(0)	0.0(0)	0.0(0)	1084(36)	1364(42)	1968(16)	977(15)	1510(33)	1704(34)	3275(16)	3292(17)	1409(35)	**2593(48)**
Total	340	39	98	98	253	258	199	198	264	263	255	255	272	**288**
		19.2 %	29.9 %	29.9 %	55.5 %	56.5 %	50.1 %	49.9 %	68.4 %	67.3 %	67.1 %	66.1 %	68.8 %	**73.7 %**
Total Cra.	1000	308	368	373	542	554	483	481	559	561	556	558	571	**592**
		31.5 %	34.0 %	34.6 %	40.0 %	40.8 %	42.0 %	41.9 %	49.2 %	49.0 %	48.8 %	48.7 %	49.5 %	**51.0 %**

Best results are in bold

Table 13 $wpm2_{shucgo}$ compared to MSE13 best complete solvers on the industrial instances of MSE13 (number and mean ratio of solved instances)

Family	#	$wpm2_{shucgo}$	$wpm2$-13∗	$msunc$	$optim$-ni	$qms2$-$g2$	$wpm1$-13	$pwbo2.33$	$maxhs13$	$ilp13$	$w/pmifu$	$wmsz09$	$msz13f$	ckm-s	$ahms$
MS															
cir-dp	3	248(2)	489(2)	21.9(2)	10.2(3)	349(1)	413(2)	3359(1)	737(1)	680(1)	**5.6(3)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)
sean-s	52	493(18)	356(14)	139(24)	1025(35)	939(18)	296(19)	1123(6)	492(6)	294(6)	**250(39)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)
Total	55	20	16	26	38	19	21	7	7	7	**42**	0	0	0	0
		50.6 %	46.8 %	56.4 %	83.7 %	34.0 %	51.6 %	22.4 %	22.4 %	22.4 %	**87.5 %**	0.0 %	0.0 %	0.0 %	0.0 %
PMS															
aes	7	0.0(0)	524(1)	163(1)	112(1)	0.0(0)	2721(1)	0.0(0)	**480(3)**	493(3)	0.0(0)	2123(1)	1661(1)	3.2(1)	2.9(1)
bcp-fir	50	52.6(49)	19.8(49)	10.5(49)	176(48)	39.6(47)	16.9(49)	276(48)	659(37)	**12.2(50)**	77.3(48)	809(35)	746(36)	170(17)	32.5(7)
bcp-hysi	17	120(16)	156(16)	230(16)	148(16)	**300(17)**	80.8(16)	139(15)	360(10)	1.8(5)	123(11)	135(6)	48.1(6)	623(2)	134(7)
bcp-hysu	38	511(34)	256(34)	251(32)	482(31)	**141(35)**	287(28)	435(29)	1114(26)	0.0(0)	113(15)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
bcp-msp	50	591(29)	804(26)	216(30)	514(15)	299(20)	693(8)	219(18)	**317(33)**	572(28)	2.8(3)	982(22)	567(19)	1625(14)	372(5)
bcp-mtg	40	746(36)	441(39)	1.2(40)	0.6(40)	**0.2(40)**	6.0(40)	0.9(40)	8.1(40)	592(28)	251(29)	418(15)	10.1(8)	140(10)	0.0(0)
bcp-syn	50	121(28)	121(24)	57.2(27)	51.2(25)	94.9(19)	154(28)	286(23)	18.0(48)	**14.7(48)**	640(21)	420(18)	319(18)	196(29)	139(14)
cir-tc	4	90.3(4)	94.4(4)	84.3(4)	58.2(4)	**37.5(4)**	145(4)	190(4)	375(1)	473(1)	376(1)	552(1)	0.0(0)	0.0(0)	0.0(0)
clo-s	50	629(35)	1310(17)	256(29)	139(35)	**741(43)**	566(35)	410(44)	968(22)	175(30)	56.5(32)	0.0(0)	188(2)	0.0(0)	0.0(0)
des	50	1188(29)	843(26)	497(33)	733(28)	**429(48)**	580(22)	378(19)	2005(17)	818(18)	402(24)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
hap-a	6	11.7(5)	51.3(5)	44.2(4)	**1.1(5)**	7.2(5)	3.1(5)	13.9(5)	58.9(5)	1209(5)	2.4(5)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
pac-pms	40	51.0(40)	287(36)	254(40)	17.8(40)	7.3(40)	6.2(40)	67.7(40)	18.4(34)	**1.0(40)**	11.4(37)	2833(1)	0.0(0)	0.0(0)	0.0(0)
pbo-mne	50	51.2(50)	279(50)	287(50)	494(50)	188(50)	780(35)	199(50)	**107(50)**	3148(2)	1224(17)	1665(29)	3509(7)	4951(4)	0.0(0)
pbo-mnl	50	15.7(50)	153(50)	85.4(50)	138(50)	79.0(50)	278(35)	129(50)	**15.2(50)**	370(2)	403(24)	1046(37)	2353(21)	3863(12)	0.0(0)
pbo-rou	15	4.3(15)	6.3(15)	1.1(15)	0.6(15)	2.7(15)	4.4(15)	13.6(15)	229(11)	33.6(15)	**0.6(15)**	2.9(5)	8.9(5)	252(1)	0.0(0)

Table 13 continued

Family	#	$wpm2_{shucgo}$	$wpm2$-13∗	$msunc$	$optim$-ni	$qms2$-$g2$	$wpm1$-13	$pwbo2.33$	$maxhs13$	$ilp13$	$w/pmifu$	$wmsz09$	$msz13f$	ckm-s	$ahms$
pro-ins	12	230(12)	351(12)	350(3)	350(12)	**121(12)**	2027(3)	2398(8)	1536(3)	0.3(1)	0.9(1)	3082(5)	480(7)	4.4(1)	0.0(0)
tpr-Mp	48	636(46)	**189(48)**	551(39)	853(38)	621(48)	224(16)	2424(24)	772(46)	2675(28)	1226(4)	0.0(0)	0.0(0)	498(3)	0.0(0)
tpr-Op	50	94.4(50)	**9.1(50)**	41.5(50)	220(50)	153(50)	1082(42)	5214(13)	18.0(50)	36(50)	0.0(0)	395(25)	1143(22)	363(25)	0.0(0)
Total	627	528	502	512	503	**543**	422	445	486	354	287	200	152	119	34
		82.9 %	80.7 %	77.9 %	80.8 %	**85.0 %**	68.4 %	71.9 %	70.6 %	54.1 %	46.2 %	29.0 %	22.8 %	15.2 %	6.0 %
WPMS															
hap-ped	100	196(98)	**142(98)**	267(93)	437(72)	1391(35)	126(93)	264(77)	857(33)	1495(22)	81.7(85)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
pac-wpms	99	401(69)	347(62)	544(12)	116(22)	3065(17)	36.0(91)	31.3(43)	13.1(85)	**0.5(99)**	9.2(45)	2249(23)	1032(27)	0.0(0)	209(2)
pre-pla	29	14.6(29)	**13.2(29)**	218(29)	63.9(29)	19.0(29)	28.0(28)	154(29)	84.6(28)	1013(12)	2.5(11)	1649(7)	6.3(5)	414(5)	49.0(1)
time	26	552(9)	1045(9)	191(8)	393(10)	789(8)	**900(11)**	1078(7)	79.0(1)	0.0(0)	90.0(8)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
upg-pro	100	105(100)	297(100)	70.0(100)	15.9(8)	0.0(0)	4.0(100)	56.3(100)	18.6(100)	**1.0(100)**	26.5(100)	311(47)	2231(90)	0.0(0)	0.0(0)
wcsp-s5d	21	17.4(14)	122(14)	13.9(14)	84.9(13)	550(15)	73.2(14)	110(7)	2.4(6)	**19.4(17)**	17.8(6)	777(5)	1278(6)	0.5(3)	293(4)
wcsp-s5l	21	11.7(14)	17.0(14)	23.7(14)	488(11)	**693(15)**	291(14)	1.0(6)	2.6(6)	899(9)	0.4(6)	1.0(3)	0.3(4)	0.9(4)	0.5(3)
Total	396	333	326	270	165	119	**351**	269	259	259	261	85	132	12	10
		76.5 %	75.5 %	67.0 %	50.7 %	46.5 %	**79.6 %**	58.5 %	53.8 %	55.3 %	50.9 %	18.9 %	26.0 %	7.2 %	5.5 %
Total Ind.	1078	**881**	844	808	706	681	794	721	752	620	590	285	284	131	44
		78.9 %	76.8 %	73.5 %	73.2 %	71.2 %	70.1 %	64.8 %	62.7 %	52.0 %	50.5 %	24.2 %	22.0 %	12.0 %	5.4 %

Best results are in bold

Table 14 $wpm2_{shucgo}$ compared to MSE13 best complete solvers on the crafted instances of MSE13 (number and mean ratio of solved instances)

Family	#	$ilp13$	$wmsz09$	$msz13f$	$maxhs13$	$wpm2_{shucgo}$	$qms2\text{-}g2$	$ahms$	$wpm2\text{-}13*$	$pwbo2.33$	$ckm\text{-}s$	$wpm1\text{-}13$	$optim\text{-}ni$	$msunc$	$w/pmifu$
MS															
mc.7	50	5160(2)	376(50)	304(50)	0.0(0)	0.0(0)	0.0(0)	637(47)	0.0(0)	0.0(0)	**293(50)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)
mc.8	50	5132(6)	290(50)	230(50)	0.0(0)	0.0(0)	0.0(0)	469(48)	0.0(0)	0.0(0)	**206(50)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)
mc-dm	62	1014(34)	193(53)	**115(53)**	745(5)	854(12)	509(8)	58.7(47)	359(11)	856(6)	63.4(52)	962(10)	996(6)	238(7)	8.4(4)
mc-sg	5	**318(4)**	3.0(3)	10.3(3)	2.5(2)	17.7(2)	179(2)	153(4)	14.4(2)	20.1(2)	554(4)	0.7(2)	0.1(1)	0.4(1)	0.1(1)
Total	167	46	**156**	**156**	7	14	10	146	13	8	**156**	12	7	8	5
		37.7 %	86.4 %	86.4 %	12.0 %	14.8 %	13.2 %	86.5 %	14.4 %	12.4 %	**91.0 %**	14.0 %	7.4 %	7.8 %	6.6 %
PMS															
frb	25	821(14)	316(5)	163(5)	704(15)	0.0(0)	**164(25)**	310(5)	0.0(0)	1265(20)	1067(5)	0.0(0)	1577(15)	0.0(0)	111(24)
job-sh	3	0.0(0)	0.0(0)	0.0(0)	217(3)	65.9(3)	**24.9(3)**	0.0(0)	55.8(3)	164(3)	0.0(0)	31.7(2)	102(3)	89.9(3)	34.3(3)
mcq-ran	96	34.3(96)	2.3(96)	**1.7(96)**	26.2(96)	569(90)	308(79)	3.1(96)	551(90)	555(71)	9.3(96)	800(59)	336(78)	191(74)	39.7(2)
mcq-str	62	349(38)	374(38)	**131(38)**	144(38)	752(28)	778(28)	67.4(31)	816(27)	57.8(19)	176(31)	225(14)	602(26)	225(22)	310(15)
mo-3s	80	9.1(80)	0.5(80)	**0.4(80)**	2.6(80)	4.4(80)	622(78)	34.8(80)	6.7(80)	398(72)	5.4(80)	144(80)	389(75)	8.3(80)	557(12)
mo-str	60	278(59)	86.8(58)	124(57)	41.3(60)	19.2(60)	22.8(60)	657(5)	45.0(60)	**9.0(60)**	1108(50)	855(42)	143(54)	60.2(60)	0.2(1)
mine-kbt	42	**212(42)**	2985(38)	2316(38)	1065(8)	1373(6)	138(6)	1202(4)	1577(5)	2010(5)	2300(28)	2147(6)	774(6)	418(6)	2482(1)
pse-ml	4	25.3(4)	583(3)	594(3)	3.9(4)	38.1(4)	**3.0(4)**	508(3)	34.0(4)	148(4)	725(3)	158(4)	7.3(4)	49.3(4)	1.4(3)
sch	5	0.0(0)	0.0(0)	0.0(0)	0.0(0)	688(1)	777(1)	0.0(0)	**804(1)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
Total	377	**333**	318	317	304	272	284	224	270	254	293	207	261	249	61
		68.4 %	60.4 %	60.2 %	71.1 %	63.7 %	**73.2 %**	40.3 %	63.2 %	65.2 %	55.0 %	48.3 %	64.6 %	58.5 %	35.1 %

Table 14 continued

Family	#	*ilp*13	*wmsz*09	*msz*13*f*	*maxhs*13	$wpm2_{shucgo}$	*qms*2-*g*2	*ahms*	*wpm*2-13∗	*pwbo*2.33	*ckm-s*	*wpm*1-13	*optim-ni*	*msunc*	*w*/*pmifu*
WMS															
frb	34	649(22)	119(14)	58.0(14)	858(25)	125(9)	2316(19)	130(14)	182(9)	1.7(4)	356(14)	446(5)	539(25)	155(9)	**72.5(33)**
ram	15	221(3)	103(4)	**29.2(4)**	2.4(1)	117(3)	231(3)	16.9(3)	93.0(3)	220(3)	883(4)	0.6(1)	167(3)	115(3)	1.6(1)
wmc-dm	62	1279(46)	277(60)	115(60)	1523(13)	2.3(5)	2052(8)	125(55)	18.5(5)	0.1(4)	**50.0(60)**	99.8(5)	1.0(4)	693(5)	0.1(3)
wmc-sg	5	**658(5)**	32.0(4)	1082(5)	1096(4)	409(1)	0.0(0)	0.1(1)	348(1)	20.4(1)	0.5(1)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
Total	116	76	82	**83**	43	18	30	73	18	12	79	11	32	17	37
		64.7 %	61.2 %	**66.2 %**	45.3 %	18.6 %	22.2 %	42.5 %	18.6 %	14.6 %	46.2 %	7.4 %	25.0 %	13.6 %	27.1 %
WPMS															
CSG	10	172(4)	806(2)	340(1)	**8.0(10)**	45.3(10)	1189(10)	242(1)	71.2(10)	188(10)	0.0(0)	1.5(4)	15.1(6)	35.9(6)	3.3(5)
auc-pat	86	**0.1(86)**	15.1(86)	6.9(86)	**0.1(86)**	316(82)	1444(47)	325(42)	276(82)	1359(30)	33.5(10)	558(72)	657(17)	840(17)	0.1(2)
auc-sch	84	0.1(84)	365(81)	216(84)	**0.1(84)**	0.9(84)	848(84)	134(68)	1.0(84)	71.5(82)	0.0(0)	1.7(84)	240(84)	8.5(84)	0.4(84)
mine-pl	56	156(56)	335(51)	66(41)	8.1(56)	0.7(56)	34.3(56)	95.1(6)	0.8(56)	**0.7(56)**	0.5(10)	4.5(53)	6.3(53)	14.7(53)	145.1(25)
mine-wa	18	**0.1(18)**	1945(9)	2354(9)	0.8(18)	2304(4)	0.3(1)	0.1(17)	2802(2)	4.0(17)	0.2(1)	19.1(14)	0.5(1)	2.8(1)	0.0(0)
pse-ml	12	142(3)	205(4)	490(3)	158(3)	260(4)	**548(5)**	636(1)	756(5)	729(4)	335(2)	316(4)	1954(4)	242(3)	0.1(1)
ran-net	74	225(60)	4267(1)	4240(8)	**11.3(73)**	2593(48)	4678(4)	389(44)	3275(16)	49.7(5)	0.0(0)	6.2(70)	0.0(0)	0.0(0)	65.8(8)
Total	340	311	234	232	**330**	288	207	179	255	204	23	301	165	164	125
		78.0 %	56.1 %	52.7 %	**89.1 %**	73.7 %	58.2 %	44.7 %	67.1 %	66.7 %	7.4 %	74.9 %	44.8 %	43.6 %	30.9 %
Total Cra.	1000	766	**790**	788	684	592	531	622	556	478	451	531	465	438	228
		65.5 %	63.6 %	63.4 %	62.2 %	51.0 %	50.3 %	49.6 %	48.8 %	48.4 %	45.6 %	43.5 %	42.7 %	38.2 %	27.8 %

Best results are in bold

Table 15 $wpm2_{shucgo}$ incomplete compared to MSE13 incomplete solvers on the industrial instances of MSE13 (number and mean ratio of solved instances)

Family	#	$wpm2_{shucgo}$	$ccls$	$ira\text{-}nov$	$optim$
MS					
cir-dp	3	6.80(1)	**4.26(3)**	0.00(0)	0.00(0)
sean-s	52	11.77(17)	**34.56(43)**	9.62(3)	9.97(2)
Total	55	18	**46**	3	2
		32.0 %	**91.0 %**	2.0 %	1.0 %
PMS					
aes	7	15.26(1)	33.09(1)	**35.81(6)**	18.54(2)
bcp-fir	50	**11.22(47)**	17.51(47)	35.25(33)	10.06(11)
bcp-hysi	17	**18.07(17)**	3.09(16)	2.86(7)	5.60(9)
bcp-hysu	38	39.74(32)	**10.34(32)**	0.00(0)	4.88(1)
bcp-msp	50	**26.50(39)**	9.02(14)	16.48(7)	10.79(6)
bcp-mtg	40	2.92(40)	**0.23(40)**	0.00(0)	11.62(2)
bcp-syn	50	5.10(25)	12.48(28)	**13.56(30)**	24.07(28)
cir-tc	4	105.67(4)	**79.21(4)**	0.00(0)	0.00(0)
clo-s	50	43.21(19)	**20.61(46)**	7.14(9)	0.00(0)
des	50	59.75(32)	**31.56(38)**	0.00(0)	0.00(0)
hap-a	6	11.09(5)	**0.41(6)**	0.00(0)	0.00(0)
pac-pms	40	34.45(40)	**1.43(40)**	0.00(0)	0.00(0)
pbo-mne	50	**57.27(47)**	17.81(41)	0.00(0)	0.00(0)
pbo-mnl	50	**20.04(49)**	16.68(48)	0.00(0)	0.00(0)
pbo-rou	15	4.14(15)	**0.33(15)**	5.45(1)	0.00(0)
pro-ins	12	96.55(9)	60.30(7)	**17.61(10)**	0.24(1)
tpr-Mp	48	**64.36(40)**	38.53(36)	0.00(0)	0.00(0)
tpr-Op	50	**6.90(50)**	47.91(42)	0.00(0)	0.00(0)
Total	627	**511**	501	103	60
		80.0 %	79.0 %	20.0 %	10.0 %
WPMS					
hap-ped	100	**38.26(89)**	28.18(64)	13.65(23)	0.00(0)
pac-wpms	99	**104.53(99)**	23.52(23)	0.00(0)	0.00(0)
pre-pla	29	**8.16(24)**	5.83(23)	13.78(10)	1.92(2)
time	26	37.22(10)	**45.85(18)**	22.34(3)	0.00(0)
upg-pro	100	**67.99(82)**	51.55(54)	0.00(0)	0.00(0)
wcsp-s5d	21	**21.20(18)**	1.98(10)	14.10(11)	1.08(5)
wcsp-s5l	21	4.66(14)	0.02(6)	**37.07(16)**	0.86(6)
Total	396	**336**	198	63	13
		77.0 %	52.0 %	28.0 %	8.0 %
Total Ind.	1078	**865**	745	169	75
		76.0 %	73.0 %	21.0 %	9.0 %

Best results are in bold

Table 16 $wpm2_{shucgo}$ incomplete compared to MSE13 incomplete solvers on the crafted instances of MSE13 (number and mean ratio of solved instances)

Family	#	*optim*	$wpm2_{shucgo}$	*ccls*	*ira-nov*
MS					
bip-mc-.7	50	**1.92(50)**	0.00(0)	40.25(31)	0.00(0)
bip-mc-.8	50	**2.61(50)**	0.00(0)	39.30(33)	0.00(0)
mc-dm	62	**1.14(62)**	2.98(13)	2.79(61)	0.61(8)
mc-sg	5	**1.63(5)**	1.00(2)	26.86(4)	0.01(1)
Total	167	**167**	15	129	9
		100.0 %	15.0 %	76.0 %	8.0 %
PMS					
frb	25	**14.77(25)**	0.00(0)	5.51(1)	7.14(2)
job-sh	3	0.00(0)	**37.52(3)**	0.00(0)	53.01(3)
mcq-ran	96	**2.10(96)**	27.26(82)	35.60(74)	22.59(71)
mcq-str	62	**13.82(55)**	7.36(28)	5.52(23)	5.79(20)
mo-3s	80	2.80(77)	**4.98(80)**	6.55(77)	31.66(62)
mo-str	60	0.87(3)	**4.04(59)**	17.74(18)	2.30(56)
mine-kbt	42	**7.24(42)**	9.27(3)	19.97(9)	6.88(4)
pse-ml	4	2.31(4)	6.03(4)	0.62(3)	**1.09(4)**
sch	5	14.51(1)	45.15(1)	0.00(0)	**19.60(3)**
Total	377	**303**	260	205	225
		67.0 %	61.0 %	37.0 %	61.0 %
WMS					
frb	34	**8.11(33)**	10.70(10)	14.46(10)	26.68(26)
ram	15	**44.77(15)**	13.33(3)	22.87(8)	0.00(0)
wmc-dm	62	**1.09(62)**	0.69(6)	2.20(62)	0.04(4)
wmc-sg	5	**52.95(5)**	5.16(1)	8.07(3)	0.00(0)
Total	116	**115**	20	83	30
		99.0 %	19.0 %	60.0 %	20.0 %
WPMS					s
CSG	10	0.00(0)	**54.41(10)**	0.00(0)	1.73(6)
auc-pat	86	**2.17(86)**	35.55(63)	30.08(46)	18.33(14)
auc-sch	84	2.72(84)	**0.98(84)**	0.71(35)	39.42(76)
mine-pl	56	1.12(7)	**0.62(56)**	0.79(10)	0.52(53)
mine-wa	18	**155.70(18)**	0.00(1)	0.04(1)	0.02(1)
pse-ml	12	32.71(4)	**26.21(7)**	1.18(1)	2.13(4)
ran-net	74	**115.17(58)**	8.02(3)	0.00(0)	27.33(13)
Total	340	**257**	224	93	167
		60.0 %	62.0 %	17.0 %	45.0 %
Total Cra.	1000	**842**	519	510	431
		76.0 %	47.0 %	42.0 %	40.0 %

Best results are in bold

References

Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: ICLP (Technical Communications), pp. 211–221 (2012)

Ansotegui, C.: Maxsat Latest Developments. Invited Tutorial at CP 2013 (2013a)

Ansotegui, C.: Tutorial: Maxsat Latest Developments (2013b)

Ansótegui, C., Gabàs, J.: Solving (weighted) partial maxsat with ilp. In: CPAIOR, pp. 403–409 (2013)

Ansótegui, C., Bonet, M.L., Levy, J.: On solving MaxSAT through SAT. In: Proceedings of the 12th International Conference of the Catalan Association for Artificial Intelligence (CCIA'09), pp. 284–292 (2009)

Ansotegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial maxsat through satisfiability testing. In: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09), pp. 427–440 (2009)

Ansotegui, C., Bonet, M.L., Levy, J.: A new algorithm for weighted partial maxsat. In: Proceedings the 24th National Conference on Artificial Intelligence (AAAI'10), pp. 867–872 (2010)

Ansótegui, C., Bofill, M., Palahí, M., Suy, J., Villaret, M.: A proposal for solving weighted CSPs with SMT. In: Proceedings of the 10th International Workshop on Constraint Modelling and Reformulation (ModRef 2011), pp. 5–19 (2011)

Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving sat-based weighted maxsat solvers. In: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12), pp. 86–101 (2012)

Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving wpm2 for (weighted) partial maxsat. In: CP, pp. 117–132 (2013a)

Ansótegui, C., Bonet, M.L., Levy, J.: Sat-based maxsat algorithms. Artif. Intell. **196**, 77–105 (2013b)

Argelich, J., Li, C.M., Manyà, F., Planes, J.: Maxsat evaluation. http://www.maxsat.udl.cat (2006–2014)

Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-boolean constraints into cnf. In: SAT, pp. 181–194 (2009)

Barrett, C., Stump, A., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). http://www.SMT-LIB.org (2010)

Berre, D.L.: Sat4j, a satisfiability library for java. www.sat4j.org (2006)

Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)

Bofill, M., Palahí, M., Suy, J., Villaret, M.: Boosting weighted csp resolution with shared bdds. Proceedings of the 12th International Workshop on Constraint Modelling and Reformulation (ModRef 2013), pp. 57–73. Uppsala, Sweden (2013)

Bonet, M.L., Levy, J., Manyà, F.: A complete calculus for Max-SAT. In: SAT, pp. 240–251 (2006)

Borchers, B., Furman, J.: A two-phase exact algorithm for max-sat and weighted max-sat problems. J. Comb. Optim. **2**(4), 299–306 (1998)

Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability modulo the theory of costs: foundations and applications. In: TACAS, pp. 99–113 (2010)

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge, MA (2009)

Davies, J., Bacchus, F.: Solving maxsat by solving a sequence of simpler sat instances. In: Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11), pp. 225–239 (2011)

Davies, J., Bacchus, F.: Exploiting the power of mip solvers in maxsat. In: SAT, pp. 166–181 (2013)

Dutertre, B., de Moura, L.: The Yices SMT Solver. http://yices.csl.sri.com (2014)

Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. JSAT **2**(1–4), 1–26 (2006)

Fu, Z., Malik, S.: On solving the partial max-sat problem. In: Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT'06), pp. 252–265 (2006)

Heras, F., Larrosa, J., Oliveras, A.: MiniMaxSat: A new weighted Max-SAT solver. In: Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07), pp. 41–55 (2007)

Heras, F., Larrosa, J., Oliveras, A.: Minimaxsat: an efficient weighted max-sat solver. J. Artif. Intell. Res. (JAIR) **31**, 1–32 (2008)

Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: Proceedings of the 25th National Conference on Artificial Intelligence (AAAI'11), pp. 36–41 (2011)

Honjyo, K., Tanjo, T.: Shinmaxsat. A Weighted Partial Max-SAT Solver Inspired by MiniSat+. Information Science and Technology Center, Kobe University, Kobe (2012)

Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: Qmaxsat: a partial max-sat solver. JSAT **8**(1/2), 95–100 (2012)

Kügel, A.: Improved exact solver for the weighted MAX-SAT problem. In: POS-10. Pragmatics of SAT, Edinburgh, UK, July 10, 2010, pp. 15–27 (2010)

Larrosa, J., Heras, F.: Resolution in max-sat and its relation to local consistency in weighted csps. In: IJCAI, pp. 193–198 (2005)

Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-sat solving. Artif. Intell. **172**(2–3), 204–233 (2008)

Li, C.M., Manyà, F.: Maxsat, hard and soft constraints. In: Biere, A., van Maaren, H., Walsh, H. (eds.) Handbook of Satisfiability. IOS Press, Amsterdam (2009)

Li, C.M., Manyà, F., Planes, J.: Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In: AAAI, pp.86–91 (2006)

Li, C.M., Manyà, F., Planes, J.: New inference rules for Max-SAT. J. Artif. Intell. Res. (JAIR) **30**, 321–359 (2007)

Li, C.M., Manyà, F., Mohamedou, N.O., Planes, J.: Exploiting cycle structures in Max-SAT. In: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09), pp. 467–480 (2009)

Lin, H., Su, K.: Exploiting inference rules to compute lower bounds for Max-SAT solving. In: IJCAI'07, pp. 2334–2339 (2007)

Lin, H., Su, K., Li, C.M.: Within-problem learning for efficient lower bound computation in Max-SAT solving. In: Proceedings of the 23th National Conference on Artificial Intelligence (AAAI'08), pp. 351–356 (2008)

Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09), p.p 495–508 (2009)

Manquinho, V.M., Martins, R., Lynce, I.: Improving unsatisfiability-based algorithms for boolean optimization. In: Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT'10), Lecture Notes in Computer Science, vol. 6175, pp. 181–193. Springer, (2010)

Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms and applications. Ann. Math. Artif. Intell. **62**(3–4), 317–343 (2011)

Martins, R., Manquinho, V.M., Lynce, I.: Exploiting cardinality encodings in parallel maximum satisfiability. In: ICTAI, pp. 313–320 (2011)

Martins, R., Manquinho, V.M., Lynce, I.: Clause sharing in parallel maxsat. In: LION, pp. 455–460 (2012)

Martins, R., Joshi, S., Manquinho, V.M., Lynce, I.: ncremental cardinality constraints for maxsat. In: Principles and Practice of Constraint Programming—20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings, pp. 531–548 (2014)

Morgado, A., Heras, F., Marques-Silva, J.: Improvements to core-guided binary search for maxsat. In: Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12), pp. 284–297 (2012)

Morgado, A., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J.: Iterative and core-guided maxsat solving: a survey and assessment. Constraints **18**(4), 478–534 (2013a)

Morgado, A., Heras, F., Marques-Silva, J.: Model-guided approaches for maxsat solving. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4–6, 2013, pp. 931–938 (2013b)

Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided maxsat with soft cardinality constraints. In: Proceedings of the Principles and Practice of Constraint Programming—20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. pp. 564–573 (2014)

Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided maxsat resolution. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27–31, 2014, pp. 2717–2723. Québec City, Canada. (2014)

Nieuwenhuis, R., Oliveras, A.: On sat modulo theories and optimization problems. In: SAT, pp. 156–169 (2006)

Pipatsrisawat, K., Darwiche, A.: Clone: Solving weighted Max-SAT in a reduced search space. In: Australian Conference on Artificial Intelligence, pp. 223–233 (2007)

Raz, R.: Resolution lower bounds for the weak pigeonhole principle. In: Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Canada, May 21–24, 2002, p 3 (2002)

Razborov, A.A.: Improved resolution lower bounds for the weak pigeonhole principle. Electron. Colloquium Comput. Complex. (ECCC) **8**(55), (2001)

Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier, Amsterdam (2006)

Sebastiani, R.: Lazy satisfiability modulo theories. J. Satisf. Boolean Model. Comput. **3**(3–4), 141–224 (2007)

Sörensen, K.: Metaheuristics the metaphor exposed. Int. Trans. Oper. Res. **22**(1), 3–18 (2015). doi:10.1111/itor.12001