Weight-Aware Core Extraction in SAT-Based MaxSAT Solving

Jeremias Berg(✉) and Matti Järvisalo

HIIT, Department of Computer Science, University of Helsinki, Helsinki, Finland
{jeremias.berg,matti.jarvisalo}@helsinki.fi

Abstract. Maximum satisfiability (MaxSAT) is today a competitive approach to tackling NP-hard optimization problems in a variety of AI and industrial domains. A great majority of the modern state-of-the-art MaxSAT solvers are core-guided, relying on a SAT solver to iteratively extract unsatisfiable cores of the soft clauses in the working formula and ruling out the found cores via adding cardinality constraints into the working formula until a solution is found. In this work we propose weight-aware core extraction (WCE) as a refinement to the current common approach of core-guided solvers. WCE integrates knowledge of soft clause weights into the core extraction process, and allows for delaying the addition of cardinality constraints into the working formula. We show that WCE noticeably improves in practice the performance of PMRES, one of the recent core-guided MaxSAT algorithms using soft cardinality constraints, and explain how the approach can be integrated into other core-guided algorithms.

1 Introduction

Several recent breakthroughs in algorithmic techniques for the constraint optimization paradigm of maximum satisfiability (MaxSAT) are making MaxSAT today a competitive approach to tackling NP-hard optimization problems in a variety of AI and industrial domains, from planning, debugging, and diagnosis to machine learning and systems biology, see e.g. [8,11,13,14,17,25,34].

A great majority of the most successful MaxSAT solvers today are based on the so-called core-guided MaxSAT solving paradigm, see e.g. [4,12,16,29–31]. Such solvers iteratively use Boolean satisfiability (SAT) solvers for finding unsatisfiable cores, i.e., sets of soft clauses that together with the hard clauses are unsatisfiable, of the input MaxSAT instance. After finding a new core, the core is essentially compiled into the MaxSAT instance via adding a cardinality constraint enforcing that one of the soft clauses in the core cannot be satisfied. An in-built property of core-guided solvers is hence that the MaxSAT instance grows

Work supported by Academy of Finland (grants 251170 COIN, 276412, 284591); and DoCS Doctoral School in Computer Science and Research Funds of the University of Helsinki.

J.C. Beck (Ed.): CP 2017, LNCS 10416, pp. 652–670, 2017.
DOI: 10.1007/978-3-319-66158-2_42

at each iteration due to compiling a new core into the instance. This can lead to the instance becoming bloated, as many, possibly large, cores are compiled, intuitively making the job of the SAT solver increasingly difficult. One way of improving core-guided solvers is to develop more efficient ways of compiling the cores, decreasing the blow-up of the instance. Most recently, progress in core-guided solvers has been made by developing new ways of compiling the cores via *soft cardinality constraints* [12,30,31].

In this work we propose *weight-aware core extraction* (WCE) as a technique that refines the process of how cores are extracted and when they are compiled into the working formula during core-guided MaxSAT search. WCE integrates knowledge of soft clause weights into the core extraction process, and allows for delaying the addition of cardinality constraints into the working formula by enabling the extraction of more cores between compilation steps, thereby also intuitively making the job of the core extractor (SAT solver) easier. In this paper we explain in detail how a specific implementation of clause cloning allows integrating WCE into PMRES, the first algorithm making use of soft cardinality constraints [31]. We also show empirically that WCE noticeably improves the performance of PMRES in practice on standard weighted partial MaxSAT benchmarks from the most recent MaxSAT solver evaluation. Going beyond PMRES, we also explain how ideas behind WCE can be integrated into other core-guided algorithms employing soft cardinality constraints, and to what extent the presented ideas can be used in some of the other MaxSAT approaches utilizing SAT solvers for core extraction.

In terms of related work, ideas underlying WCE have been previously applied for computing lower bounds for MaxSAT instances [18,20–23]. Specifically, the lower bounds are applied in the context of core-guided MaxSAT solving before the actual search in [18]. Furthermore, WCE also bears some resemblance with the (weaker) approaches to obtaining bounds during branch-and-bound search for MaxSAT based on detecting unsatisfiable cores by e.g. unit propagation [20–23].

The rest of the paper is organized as follows. After necessary background on MaxSAT (Sect. 2) and a detailed description of the PMRES algorithm (Sect. 3), we present our main contributions, weight-aware core extraction in the context of PMRES (Sect. 4). We then present empirical results on the speed-ups obtained via WCE on PMRES (Sect. 5), and further, explain how and to what extent the presented technique can be integrated into other SAT-based MaxSAT algorithms (Sect. 6).

2 Maximum Satisfiability

For background on weighted partial maximum satisfiability (MaxSAT in short), recall that for a Boolean variable x, there are two literals, the positive x and the negative $\neg x$. A clause is a disjunction ($\vee$) of literals, and a CNF formula a conjunction ($\wedge$) of clauses. When convenient, we treat a clause as a set of literals and a CNF formula as a set of clauses. We assume familiarity with other logical

connectives and denote by $\text{CNF}(\phi)$ a set of clauses logically equivalent to the formula ϕ; we can assume without loss of generality that the size of $\text{CNF}(\phi)$ is linear in the size of ϕ [33].

A MaxSAT instance consists of a set of hard clauses F_h, a set of soft clauses F_s, and a function $w\colon F_s \to \mathbb{N}$ that associates a positive integral cost to each of the soft clauses. We extend w to a set $S \subseteq F_s$ of soft clauses by $w(S) = \sum_{C \in S} w(C)$. Further, let $w_S^{\min} = \min_{C \in S}\{w(C)\}$, i.e., the smallest weight among the clauses in S. If $w(C) = 1$ for all $C \in F_s$, the instance is unweighted.

A truth assignment τ is a function from Boolean variables to true (1) and false (0). A clause C is satisfied by τ if $\tau(l) = 1$ for a positive or $\tau(l) = 0$ for a negative literal $l \in C$. A CNF formula is satisfied by τ if τ satisfies all clauses in the formula. If some τ satisfies a CNF formula, the formula is satisfiable, and otherwise unsatisfiable. An assignment τ is a solution to a MaxSAT instance $F = (F_h, F_s, w)$ if τ satisfies F_h. We denote the set of soft clauses not satisfied by τ by $F_{\bar{\tau}}$, i.e., $F_{\bar{\tau}} = \{C \in F_s \mid \tau(C) = 0\}$. The cost of τ is $w(F_{\bar{\tau}})$. A solution τ is optimal (for F) if $w(F_{\bar{\tau}}) \leq w(F_{\bar{\tau}'})$ for every solution τ' to F. We denote the cost of optimal solutions to F by $\textsc{Cost}(F)$. Without loss of generality, we will assume that a MaxSAT instance always has a solution, i.e., that F_h is satisfiable.

A central concept in modern SAT-based MaxSAT algorithms is that of *(unsatisfiable) cores*. For a MaxSAT instance $F = (F_h, F_s, w)$, a subset $S \subseteq F_s$ of soft clauses is an unsatisfiable core of F iff $F_h \cup S$ is unsatisfiable. An unsatisfiable core S is minimal (an MUS) of F iff $F_h \cup S'$ is satisfiable for all $S' \subset S$.

3 The PMRES Algorithm

In order to explain weight-aware core extraction, we will use the PMRES algorithm [31]. Figure 1 gives PMRES in pseudo-code. When invoked on a MaxSAT

```
1  PMRES(F_h, F_s, w):
2  (F_h^w, F_s^w) ← (F_h, F_s)
3  while true do
4      (result, κ, τ) ← SATSOLVE(F_h^w ∪ F_s^w)
5      if result="satisfiable" then return τ;
6      else
7          R ← ∅
8          w_κ^min ← min{w(C) | C ∈ κ}
9          for C_i ∈ κ do
10             F_s^w.remove(C_i)
11             if w(C_i) > w_κ^min then
12                 F_s^w.add(CL(C_i))
13                 w(CL(C_i)) ← w(C_i) − w_κ^min
14             F_h^w ← F_h^w.add((C_i ∨ r_i))
15             R.add(r_i)
16         RELAX(w_κ^min, R)
```

```
1  RELAX(w_κ^min, R):
2  n ← |R|
3  F_h^w.add((r_1 ∨ ... ∨ r_n))
4  for i=1...n-1 do
5      F_h^w.add(CNF(d_i ↔ (r_{i+1} ∨ d_{i+1})))
6      F_s^w.add((¬r_i ∨ ¬d_i))
7      w((¬r_i ∨ ¬d_i)) ← w_κ^min
```

Fig. 1. The PMRES algorithm.

instance (F_h, F_s, w) PMRES works by iteratively calling a SAT solver (line 4) on a working formula, initialized to $F_h \cup F_s$, i.e., considering all hard and soft clauses of the input formula as a SAT instance (line 2). If the working formula is satisfiable (line 5), PMRES returns the satisfying assignment reported by the SAT solver, which is guaranteed to be an optimal solution to the MaxSAT instance [31]. Otherwise the SAT solver returns an unsatisfiable core κ of the working formula. PMRES then proceeds by removing all of the soft clauses in the core from the working formula and *cloning* a subset of them; clause cloning is a common way of extending MaxSAT algorithms from unweighted to weighted MaxSAT [3,6,12,30,31], and works as follows. First the minimum-weight $w_\kappa^{\min}$ of clauses in the core κ is determined (line 8). Then each clause in *core* is removed (line 10), and a soft clone $\mathrm{CL}(C)$ of each clause $C \in \kappa$ with $w(C) > w_\kappa^{\min}$ is introduced to the working formula and given the weight $w(C) - w_\kappa^{\min}$ (lines 11–13).

After clause cloning, PMRES extends each $C \in \kappa$ by a fresh relaxation variable r and adds the extended clause $C \vee r$ as hard to the working formula (line 14). The intuition here is that setting $r = 1$ allows for the corresponding soft clause to be left unsatisfied, while setting $r = 0$ forces the corresponding clause to be satisfied. Finally, PMRES *relaxes* the found core by adding a soft cardinality constraint over the introduced r variables via the function $\mathrm{RELAX}(w_\kappa^{\min}, \mathcal{R})$ (line 16). The added cardinality constraint is encoded as hard and soft clauses using additional new variables $d_1, \ldots, d_{|\kappa|-1}$, and essentially enforces that either exactly one of the introduced relaxation variables is set to true, or some soft clause corresponding to $(r_i \rightarrow \neg d_i)$ is falsified (lines 2–7 of RELAX). In order to see this, notice first that the hard clause $(r_1 \vee \ldots \vee r_{|\kappa|})$ forces at least one relaxation variable to be set to true, Assume then that two variables r_k and r_t for some $k < t$ are both set to true. Then the hard clauses of form $d_i \leftrightarrow (r_{i+1} \vee d_{i+1})$ imply that d_j is set to true for all $j < t$. Specifically the variable d_k is set to true, and the soft clause encoding $(r_k \rightarrow \neg d_k)$ will become falsified.

We end this section by discussing two improvements that have been proposed for PMRES and other similar core-guided MaxSAT algorithms; the so-called stratification and hardening rules [4,5,26]. Assume that PMRES in invoked on a MaxSAT instance $F = (F_h, F_s, w)$. The stratification rule aims at prioritizing the extraction of cores κ for which $w_\kappa^{\min}$ is large. Since the sum of the minimum weights of the extracted cores is a lower bound on the optimal cost of the MaxSAT instance, the goal in extracting cores with large minimum weights is to decrease the total number of iterations required for termination. More precisely, PMRES extended with stratification maintains a bound $w_{\max}$, initialized by a heuristic. During solving, PMRES does not invoke the SAT solver on $F_h^w \cup F_s^w$, i.e., all of the clauses of the working formula, but rather, only on a subset of them consisting of all hard clauses and the soft clauses with weight greater than $w_{\max}$. Whenever this subset of the working formula is satisfiable, the algorithm checks if the SAT solver was invoked on the whole working formula, i.e., whether $w_{\max} = 1$. If that is the case, the algorithm terminates. Otherwise the value of $w_{\max}$ is decreased heuristically, and the search continues.

Several different strategies for updating $w_{\max}$ have been proposed [4,5]. A fairly simple one is to initialize $w_{\max}$ to the maximum weight of the soft clauses, i.e., $w_{\max} = \max\{w(C) \mid C \in F_s\}$ and update it by decreasing it to the highest weight of soft clauses that is lower than the current value of $w_{\max}$.[1]

The *hardening rule* attempts to further exploit information that can be obtained from the satisfying assignments obtained during solving in conjunction with stratification. For some intuition, notice that all subsets of the working formula that PMRES with stratification invokes the SAT solver on, always include F_h. Hence whenever the SAT solver returns satisfiable, the returned assignment τ is a solution to the MaxSAT instance, and as such an upper bound on the optimal cost of the instance. The hardening rule exploits this fact by noting that any solution τ_2 that does not satisfy a clause $C \in F_s^w$ with $w(C) > w(F_{\bar{\tau}})$ will have $w(F_{\bar{\tau}_2}) > w(F_{\bar{\tau}})$ and as such can not be an optimal solution to F. Hence all such soft clauses have to be satisfied by any optimal solution to F and can therefore be hardened, i.e., turned into hard clauses.

Even though the presentation here is specific to PMRES, the stratification and hardening rules can be used in conjunction with several different core-guided MaxSAT algorithms. This is also the case for weight-aware core extraction presented next.

4 Weight-Aware Core Extraction for PMRES

We now describe weight-aware core extraction (WCE), a generic technique designed to improve performance of PMRES and other similar MaxSAT algorithms. WCE delays the addition of cardinality constraints to the working formula with the aim of extracting more valid cores (or "core mining") from the working instance before adding more constraints to the formula.

Clause Cloning Through Assumptions (Without Cloning). WCE requires clause cloning to be implemented in a specific way through *assumptions* which essentially avoid actual clause cloning (copying) altogether. A similar approach to clause cloning is taken in [1]. For more details, we first need to overview how core extraction is usually implemented in SAT-based MaxSAT solving. Several modern SAT solvers allow querying for the satisfiability of a CNF formula under a set of assumptions, represented as a partial assignment of the variables in the formula. Whenever the formula is unsatisfiable under those assumptions, the SAT solver returns some subset of the assumptions that are required in the proof of unsatisfiability. Notice that not only are unsatisfiable formulas unsatisfiable under all assumptions, but a satisfiable formula may also be unsatisfiable under some assumptions. For example, consider the CNF formula $F = \{(x \vee y), (\neg x)\}$. Although the formula is satisfiable, it is unsatisfiable when assuming $x = 1$ or $y = 0$.

[1] In our implementation used in the experiments of this work, we use the slightly more sophisticated *diversity heuristic* [5] which attempts to balance the number of new soft clauses introduced and the amount that $w_{\max}$ is decreased.

Core-guided MaxSAT solvers make use of the assumptions interface in SAT solvers by extending each soft clause $C \in F_s$ with a fresh assumption variable $\text{A}(C)$ and sending the extended clause $C \vee \text{A}(C)$ to the SAT solver. During each SAT solver call, all assumption variables are assumed false, thus reducing all extended clauses $C \vee \text{A}(C)$ to C. Whenever the working formula is unsatisfiable, the SAT solver will return the extracted core κ in terms of the subset of assumption variables corresponding to the clauses in κ. Importantly for the PMRES algorithm, this means that each clause $C_i \in \kappa$ is already extended with the variable $\text{A}(C_i)$ and that variable can be reused as the relaxation variable r_i that would otherwise be introduced on lines 15–16 in the algorithm described in Fig. 1. Notice that whenever the SAT solver is invoked without assuming the value of $\text{A}(C_i)$ and the variable only appears in the extended clause $C_i \vee \text{A}(C_i)$, it can be set to true by the SAT solver, thereby satisfying the extended clause and effectively removing the clause from the formula. The same argument does not hold as soon as other constraints involving $\text{A}(C_i)$ are added to the formula.

With this, clause cloning through assumption variables is implemented as follows. Assume that a clause $C \in \kappa$ extended with the assumption variable $\text{A}(C)$ needs to be cloned, i.e., it is a member of some extracted core κ and $w(C) > w_\kappa^{\min}$. A simple way of improving on the naive description of clause cloning in Sect. 3 is to introduce a new soft clause $C' = (\neg\text{A}(C))$ with weight $w(C') = w(C_i) - w_\kappa^{\min}$. The correctness of this follows by noting that the extended clause $(C \vee \text{A}(C))$ will be hard in all subsequent SAT solver calls. Thus satisfying C' forces the clause C to be satisfied as well, achieving the same effect as cloning the whole C. To further improve on this, notice that as C' would be added as a soft clause, it would also be extended with an assumption variable and the extended clause $C' \vee \text{A}(C')$ would be sent to the SAT solver. This creates the logical chain $\neg\text{A}(C') \rightarrow \neg\text{A}(C) \rightarrow C$. The basic form of clause cloning would then assume the variable $\text{A}(C')$ to false in subsequent SAT solver calls, thus forcing C as well. To refine this, note that the same affect is achieved by simply assuming the value of $\text{A}(C)$ to false instead, thus removing the need of introducing the clause C' at all. In more detail, when a core κ is extracted, the minimum weight $w_\kappa^{\min}$ is computed. Then the weight of each clause $C \in \kappa$ is decreased by $w_\kappa^{\min}$. In subsequent SAT calls, the assumption variable of each clause C with weight $w(C) > 0$ is assumed false, essentially treating that clause as soft. All other clauses are treated as hard. Refining clause cloning in this way blurs the line between hard and soft clauses. When discussing PMRES with clause cloning implemented through assumptions we say that a clause C is soft as long as the internal SAT solver is invoked assuming $\text{A}(C) = 0$, i.e., as long as $w(C) > 0$. When $w(C)$ drops to 0, the extended clause $(C \vee \text{A}(C))$ becomes hard. Notice that in order for $w(C)$ to become 0, the clause C has appeared in at least one core. Hence we have added a cardinality constraint over $\text{A}(C)$ so not assuming the value of it does not remove the clause C from the formula.

Except for removing the need of introducing clones to the formula, implementing clause cloning through assumptions also results in tighter cardinality constraints, as illustrated by the following example.

Example 1. Let $F = (F_h, F_s, w)$ be a MaxSAT instance $F_h = \{(x \vee y), (y \vee z)\}$, $F_s = \{C_1 = (\neg x), C_2 = (\neg y), C_3 = (\neg z)\}$, and $w(C_1) = 1$ and $w(C_2) = w(C_3) = 2$. Assume that we invoke the basic version of PMRES, i.e., the algorithm in Fig. 1, on F and that it first extracts the core $\{C_1, C_2\}$. After relaxing the core the working instance (F_h^w, F_s^w, w) consists of $F_h^w = F_h \wedge (C_1 \vee r_1) \wedge (C_2 \vee r_2) \wedge \mathrm{CNF}(r_1 + r_2 = 1)_h$, $F_s^w = \mathrm{CL}(C_2) \wedge C_3 \wedge \mathrm{CNF}(r_1 + r_2 = 1)_s$ with $w(\mathrm{CL}(C_2)) = 1, w(C_3) = 2$. Here we use $\mathrm{CNF}(r_1 + r_2 = 1)_h$ and $\mathrm{CNF}(r_1 + r_2 = 1)_s$ to denote the hard and soft clauses, respectively, introduced in the RELAX subroutine. If PMRES next extracts and relaxes the core $\{\mathrm{CL}(C_2), C_3\}$, the final working formula will have the hard clauses $F_h^w = F_h \wedge (C_1 \vee r_1) \wedge (C_2 \vee r_2) \wedge (\mathrm{CL}(C_2) \vee r_3) \wedge (C_3 \vee r_4) \wedge \mathrm{CNF}(r_1 + r_2 = 1)_h \wedge \mathrm{CNF}(r_3 + r_4 = 1)_h$ and the soft clauses $F_s^w = \mathrm{CL}(C_3) \wedge \mathrm{CNF}(r_1 + r_2 = 1)_s \wedge \mathrm{CNF}(r_3 + r_4 = 1)_s$. This instance is satisfiable by setting $r_2 = r_3 = 1$ and $r_1 = r_4 = 0$. In total there are 4 different ways of satisfying the added cardinality constraints. A similar argument holds even if we use the assumption variables of soft clauses in cores when encoding the cardinality constraints and introduce the negations of those variables as soft clauses when performing clause cloning.

If we instead use assumptions to implement clause cloning, the final working formula will have the hard clauses $F_h^w = F_h \wedge (C_1 \vee \mathrm{A}(C_1)) \wedge (C_2 \vee \mathrm{A}(C_2)) \wedge \mathrm{CNF}(\mathrm{A}(C_1) + \mathrm{A}(C_2) = 1)_h \wedge \mathrm{CNF}(\mathrm{A}(C_2) + \mathrm{A}(C_3) = 1)_h$ and the soft clauses $F_s^w = C_3 \wedge \mathrm{CNF}(\mathrm{A}(C_1) + \mathrm{A}(C_2) = 1)_s \wedge \mathrm{CNF}(\mathrm{A}(C_2) + \mathrm{A}(C_3) = 1)_s$ with $w(C_1) = w(C_2) = 0$ and $w(C_3) = 1$. As $w(C_3) > 0$, the final SAT call will be made assuming $\mathrm{A}(C_3) = 0$. Under the assumption, there is only a single way of satisfying the added cardinality constraints. Even disregarding the assumptions, there are only 2 ways of satisfying the added cardinality constraints with one of them resulting in the rest of the instance becoming satisfiable. ■

WCE. Having discussed clause cloning in conjunction with WCE, we now turn to describing WCE in detail. For some intuition, consider the following example.

Example 2. Consider again the MaxSAT instance F from Example 1 and assume that PMRES with clause cloning implemented through assumptions first extracts $\{C_1, C_2\}$. The working formula (F_h^w, F_s^w, w) will then become $F_h^w = F_h \wedge (C_1 \vee \mathrm{A}(C_1)) \wedge \mathrm{CNF}(\mathrm{A}(C_1) + \mathrm{A}(C_2) = 1)_h$ and $F_s^w = C_2 \wedge C_3 \wedge \mathrm{CNF}(\mathrm{A}(C_1) + \mathrm{A}(C_2) = 1)_s$ with $w(C_2) = 1, w(C_3) = 2$. The only core of the instance is $\{C_2, C_3\}$. Notice, however, that ignoring the added cardinality constraints at this stage and invoking the SAT solver on the (simpler) subset of the working formula consisting of $F_h^w = F_h \wedge (C_1 \vee \mathrm{A}(C_1))$ and $F_s^w = C_2 \wedge C_3$ with $w(C_2) = 1, w(C_3) = 2$ would result in the exact same core being extracted. ■

The pseudocode of PMRES extended with WCE is shown in Fig. 2. Before invoking its SAT solver, PMRES with WCE first adds an assumption $\mathrm{A}(C) = 0$ for all soft clauses C with $w(C) > 0$ (line 5). Then it invokes the SAT solver on the working formula with these assumptions. If a core κ is extracted, $w_\kappa^{\min}$ is computed and the weight of all clauses in the core decreased by $w_\kappa^{\min}$ (lines 14 and 16). However, instead of immediately calling $\mathrm{RELAX}(w_\kappa^{\min}, \mathcal{R})$, the tuple

```
1  PMRES+WCE(F_h, F_s, w):
2  (F_h^w, F_s^w) ← (F_h, F_s)
3  ℝ ← ∅
4  while true do
5      A ← {A(C) = 0 | C_i ∈ F_s^w, w(C) > 0}
6      (result, κ, τ) ← SATSOLVE(F_h^w ∪ F_s^w, A)
7      if result="satisfiable" AND |ℝ| = 0 then return τ;
8      else if result="satisfiable" then
9          for (R, w_κ^min) ∈ ℝ do
10             RELAX(w_κ^min, R)
11         ℝ ← ∅
12     else
13         R ← ∅
14         w_κ^min ← min{w(C) | C ∈ κ}
15         for C ∈ κ do
16             w(C) ← w(C) − w_κ^min
17             R ← R ∪ {A(C)}
18         ℝ.add((R, w_κ^min))
```

Fig. 2. PMRES+WCE, the PMRES algorithm with WCE. In the pseudocode, the assumption variable of a soft clause C used in core extraction is given by $\text{A}(C)$.

$(\mathcal{R}, w_\kappa^{\min})$ is added to the set $\mathbb{R}$ (line 18). Then the SAT solver in invoked again with a new set of assumptions. Notice that at each iteration, the weight of at least one soft clause C is dropped to 0. In subsequent SAT solver calls the value of $\text{A}(C)$ is not assumed anymore, effectively removing that clause from the formula until the cardinality constraints are added, which is why the working formula will eventually become satisfiable. The algorithm then checks if new cores have been extracted since the last time cardinality constraints were added. If so, the corresponding cardinality constraints are added to the formula and the loop iterates (lines 8–11). If there are no new cores, the algorithm terminates and returns the satisfying truth assignment as an optimal MaxSAT solution (line 7).

Similarly to the stratification rule, all working formulas of PMRES extended with WCE contain the original hard clauses F_h. As such, whenever the working formula is satisfiable, the algorithm obtains an upper bound on the cost of the optimal solutions. The bound might in some cases allow PMRES with WCE to terminate even before all cardinality constraints have been added to the working formula.

Example 3. Consider the MaxSAT instance F from Examples 1 and 2. Invoke PMRES with WCE on F and assume that the first core it extracts is again $\kappa^1 = \{C_1, C_2\}$. Now the addition of cardinality constraints is delayed and the SAT solver is invoked on $F_h^w = F_h \wedge (C_1 \vee \text{A}(C_1))$ and $F_s^w = C_2 \wedge C_3$ with $w(C_1) = 0, w(C_2) = 1$ and $w(C_3) = 2$. As $w(C_1) = 0$, the variable $\text{A}(C_1)$ is not assumed to any value and the clause $(C_1 \vee \text{A}(C_1))$ can be satisfied by setting $\text{A}(C_1) = 1$. Nevertheless, PMRES+WCE still extracts the core $\kappa^2 = \{C_2, C_3\}$. On the third

iteration the SAT solver is invoked on $F_h^w = F_h \wedge (C_1 \vee \text{A}(C_1) \wedge (C_2 \vee \text{A}(C_2))$ and $F_s^w = C_3$ with $w(C_1) = w(C_2) = 0$ and $w(C_3) = 1$ assuming $\text{A}(C_3) = 0$. This instance is satisfiable. Next the algorithm adds cardinality constraints to form the same (satisfiable) final working instance as shown in Example 1. Then it would invoke the SAT solver on that instance, find it satisfiable, and terminate.

However, by investigating the second to last SAT solver call we see that PMRES+WCE might be able to terminate without adding any cardinality constraints at all. First note that after extracting the cores κ^1 and κ^2 we know that $\text{Cost}(F) \geq w_{\kappa^1}^{\min} + w_{\kappa^2}^{\min} = 1 + 1 = 2$. Now, the second to last SAT solver call is performed on the clauses $(x \vee y), (y \vee z), (\neg x \vee \text{A}(C_1)), (\neg y \vee \text{A}(C_2)), (\neg z \vee \text{A}(C_3))$ assuming $\text{A}(C_3) = 0$. The assumption propagates $z = 0$ which in turn propagates $y = 1$ and $\text{A}(C_2) = 1$. At this point, all clauses except for $(\neg x \vee \text{A}(C_1))$ are already satisfied. If the internal SAT solver now satisfies the clause by setting $x = 0$, the cost of the assignment it returns will be 2, thus proving that $\text{Cost}(F) \leq 2$ and allowing the algorithm to terminate early. Although we in general can not guarantee early termination, empirically we found that it does happen. ■

The correctness of WCE is based on the correctness of PMRES. For more intuition, note that all cores that are extractable by PMRES with WCE are a subset of the cores that could be extracted by PMRES with clause cloning implemented through assumptions, and that the final working instance of both algorithms is the same.

Related Work. The method presented in [18] for computing MaxSAT lower bounds is equivalent to running Algorithm 2 until the working instance becomes satisfiable for the first time, returning the sum $\sum w_\kappa^{\min}$ over all of the cores extracted; already this lower bounding step is shown in [18] to improve the performance of specific MaxSAT algorithms compared to starting search with the trivial bound of 0. Alternatively, WCE can be seen as a more thorough integration of the bound computation and the MaxSAT algorithm itself by performing the lower bound computation *in-between* each core compilation step. Computation of lower bounds has also received significant interest in the context of branch-and-bound MaxSAT solvers [20–23], which rely heavily on good lower bounds in order to prune the search tree. For example, in [21] the authors propose a technique in which unit propagation is used to extract several cores of the working instance in order to compute a lower bound. The main difference to WCE is that WCE is not limited to cores detectable by unit propagation.

Integrating Stratification and Hardening. We end this section by discussing how the commonly used stratification and hardening rules can be integrated with WCE. There are two obvious ways of integrating stratification in conjunction with WCE. The first one is to prefer WCE to stratification: initialize the bound $w_{\max}$ heuristically [5] and assume the assumption variable $\text{A}(C)$ to false only for each of the clauses C with $w(C) \geq w_{\max}$. Then iteratively extract

cores over the subset of soft clauses under consideration, delaying the addition of cardinality constraints until the instance becomes satisfiable. At that point, add the cardinality constraints and continue. Whenever the instance remains satisfiable after the addition of cardinality constraints, harden any possible clauses and decrease the bound $w_{\max}$. The algorithm can terminate when no new cores can be extracted and the SAT solver has been invoked on all soft clauses.

Another, dual way of integrating the two is to prefer stratification to WCE, i.e., to delay the addition of cardinality constraints until $w_{\max}$ has been decreased to 1, at which point all cardinality constraints are added and the bound $w_{\max}$ is reinitialized. However, the choice between preferring stratification or WCE to the other influences the applicability of the hardening rule and the quality of the satisfiable assignments produced by WCE and stratification. Preferring WCE to stratification we know that, whenever the bound $w_{\max}$ needs to be lowered, all clauses C with $w(C) \geq w_{\max}$ are satisfied, thus allowing for the hardening of several soft clauses. In contrast, when preferring stratification to WCE, sound use of the hardening rule requires considering the delayed soft cardinality constraints, of which the SAT solver has had no information during search. The empirical results presented next support this intuition, as preferring WCE to stratification leads to performance boosts within PMRES, while preferring stratification to WCE actually degrades performance compared to PMRES without using WCE.

5 Experiments

We investigate how WCE affects the performance of the PMRES algorithm in practice. Since the implementation of PMRES by the original authors—coined Eva500a as it participated in MaxSAT Evaluation 2014 [9]—is not available in open source, we re-implemented PMRES on top of the open-source core-guided MaxSAT solver Open-WBO [29], following the description in the paper introducing the algorithm [31] using Glucose [10] as the underlying incremental SAT solver.

In the experiments we compare the following MaxSAT algorithms.

- PMRES: our re-implementation of the PMRES algorithm using stratification, implemented using assumption variables on soft clauses, hardening, and clause cloning implemented through assumptions.
- PMRES+WCE: PMRES extended with WCE, preferring WCE to stratification.
- PMRES+WCE (S/to/WCE): PMRES extended with WCE, preferring stratification to WCE.
- Eva500a [31]: the closed-source implementation of PMRES that participated, and won the industrial category of the 2014 MaxSAT Evaluation.

For reference, we also provide a comparison with MSCG15b [30], a closed-source as the best-performing core-guided MaxSAT solver using soft cardinality constraints in 2016 MaxSAT evaluation. As we will explain later in Sect. 6, WCE

could also be integrated into MSCG.[2] As benchmarks we used the weighted partial industrial (630) and crafted (331) instances from the 2016 MaxSAT Evaluation [9]. The experiments were run on 2.83-GHz Intel Xeon E5440 quad-core machines with 32-GB RAM and Debian GNU/Linux 8 using a per-instance timeout of 3600 s.

An overview of the results, comparing the performance of Eva500a and the variants PMRES, PMRES+WCE, and PMRES+WCE (S/to/WCE) of our implementation, is provided through Figs. 3, 4, and Table 1. The "cactus" plot of Fig. 3 gives the number of instances solved (x-axis) by the individual solvers under different per-instance time limits (y-axis) over all benchmarks. More detailed results are provided in Table 1, with the industrial and crafted benchmarks separated by domain, showing the number of instances from each domain, and the number of solved instances and the cumulative running time used for solving the solved instances for each solver. First, note that our PMRES re-implementation is competitive in terms of overall performance with Eva500a; on the industrial instances PMRES solves three more instances overall and uses cumulatively only 55% of the running time that Eva500a uses on the respectively solved instances. On the crafted instances, PMRES solves two instances less than Eva500a but still uses noticeably less time over all solved instances.

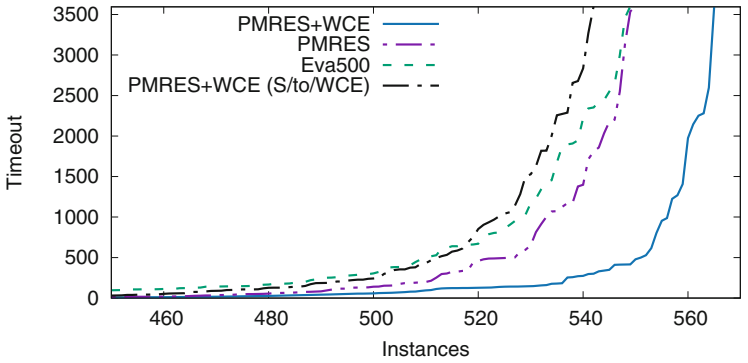

Fig. 3. Solver comparisons: number of instances solved (x-axis) by the individual solvers under different per-instance time limits (y-axis).

Turning to the influence of WCE on the performance of PMRES, we observe that PMRES+WCE (preferring WCE to stratification) has noticeably improved performance wrt PMRES (and thus also Eva500a), solving 11 more industrial and 3 more crafted instances (14 and 1 more than Eva500a). Notice that of all three solvers, PMRES+WCE is the best performing on both industrial and crafted benchmarks. Most interestingly, PMRES+WCE uses at the same time much less time on the solved instances; on the industrial instances

[2] Unfortunately, we do not have access to the source code of MSCG.

Table 1. Comparison of Eva500a, PMRES, and PMRES+WCE: number of solved instances (#) and the cumulative running time used for solving the instances (Σ) for the individual solvers, divided into the industrial (top) and crafted (bottom) benchmarks according to the individual domains with the number of instances from each domain given in parentheses.

	Eva500a		PMRES		PMRES+WCE	
	Solved	Time (s)	Solved	Time (s)	Solved	Time (s)
	#	Σ	#	Σ	#	Σ
Industrial domain (*#instances*)						
abstraction refinement (11)	6	3670	9	2842	**10**	**2147**
BTBNSL (60)	9	403	16	6142	**19**	679
correlation clustering (129)	18	17630	11	4559	**19**	**4499**
haplotyping pedigrees (100)	100	7321	100	3409	100	**1374**
hs-timetabling (14)	1	**477**	1	1858	1	2596
packup-wpms (99)	**99**	2981	95	969	94	191
preference planning (29)	29	1416	29	311	29	**264**
railway transport (11)	2	126	3	603	3	**340**
relational inference (9)	5	8391	8	1431	8	**1360**
timetabling (26)	12	1818	12	1846	12	**878**
upgradeability (100)	100	2996	100	**54**	100	59
wcsp_spot5_dir (21)	14	30	14	**13**	14	24
wcsp_spot5_log (21)	14	61	14	1975	14	**17**
Total industrial (630)	409	47319	412	26012	**423**	**14426**
Crafted domain (*#instances*)						
auctions (40)	**40**	6111	39	2691	38	1759
causal discovery (35)	**10**	9325	6	1267	6	1357
CSG (10)	7	610	8	**1056**	8	1370
frb (34)	**20**	**3310**	12	4461	17	4041
min-enc (48)	32	198	36	758	36	**455**
miplib (12)	5	1558	5	**247**	5	437
ramsey (15)	1	1	3	2282	3	**2073**
random-net (32)	13	4001	13	1196	13	**5**
set-covering (45)	9	2069	**10**	1520	9	155
staff-scheduling (12)	1	0	2	**1069**	2	1406
wmaxcut (48)	1	44	3	3208	3	**2960**
Total crafted (331)	139	27226	137	19756	**140**	**16018**

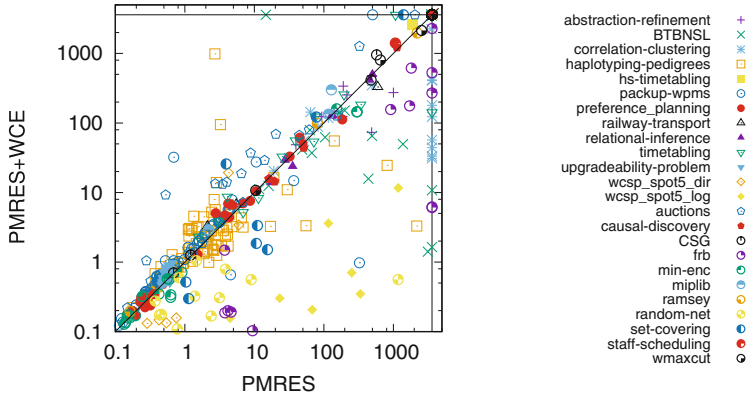

Fig. 4. Solver comparisons: per-instance running time comparison of PMRES (x-axis) and PMRES+WCE (y-axis), with the ticks below the $y = x$ line representing instances on which PMRES+WCE is faster than PMRES.

PMRES+WCE uses in total 55% of the time PMRES uses and 30% of the time Eva500a uses, even though PMRES+WCE solves more instances than PMRES and Eva500a individually. A similar observation can be made of the crafted instances; PMRES+WCE uses 81% of the time used by PMRES and 59% of the time used by Eva500a, again solving more instances than either one. The scatter plot of Fig. 4 gives a per-instance running time comparison on a log-log scale of PMRES+WCE and PMRES, with the ticks below the $y = x$ line representing instances on which PMRES+WCE is faster that PMRES. The colors of the ticks distinguish between the benchmark domains listed in Table 1.

Next, we consider the question of the relative influence of preferring stratification or WCE within PMRES. Here we observe that PMRES+WCE (S/to/WCE)—preferring stratification over WCE—actually harms the overall performance of PMRES noticeably, making it perform worse than Eva500a overall (see Fig. 3). This supports the earlier discussed intuition that preferring WCE to stratification assures that whenever the bound $w_{\max}$ needs to be lowered, all clauses C with $w(C) \geq w_{\max}$ are satisfied, thus allowing for hardening several soft clauses. In contrast, when preferring stratification to WCE, sound use of the hardening rule requires considering the delayed soft cardinality constraints, of which the SAT solver has had no information during search.

Finally, we consider the relative performance of PMRES+WCE and MSCG15b. Here we note that this is not a direct comparison of the influence of WCE in the sense that MSCG15b does not implement the PMRES algorithm of Eva500a, but rather a different core-guided algorithm using soft cardinality constraints, OLL [2,30]. As we will explain later in Sect. 6, WCE can also be integrated into the OLL algorithm. However, we could not implement WCE directly to MSCG as MSCG is not available in open source. Nevertheless, a comparison of the performance of PMRES+WCE and MSCG15b is provided in Table 2. Overall MSCG15b solves 12 more industrial instances than PMRES+WCE. How-

Table 2. Comparison of MSCG15b and PMRES+WCE: percentage (%) and number (#) of solved instances and the cumulative running time used for solving the instances (Σ) for the individual solvers, divided into industrial (top) and crafted (bottom) benchmarks according to the individual domains with the number of instances from each domain given in parentheses.

	MSCG15b			PMRES+WCE		
	Solved		Time (s)	Solved		Time (s)
	%	#	Σ	%	#	Σ
Industrial domain (#instances)						
abstraction refinement (11)	90.9	10	17096	90.9	10	**2147**
BTBNSL (60)	21.7	13	885	**31.7**	**19**	**679**
correlation clustering (129)	**25.6**	**33**	19780	14.7	19	4499
haplotyping pedigrees (100)	100.0	100	**1343**	100.0	100	1374
hs-timetabling (14)	7.1	1	**167**	7.1	1	2596
packup-wpms (99)	**100**	**99**	410	95.0	94	191
preference planning (29)	100	29	2021	100	29	**264**
railway transport (11)	27.3	3	**283**	27.3	3	340
relational inference (9)	44.4	4	3167	**88.9**	**8**	**1360**
timetabling (26)	46.2	12	**764**	46.2	12	878
upgradeability (100)	100.0	100	118	100.0	100	**59**
wcsp_spot5_dir (21)	**81.0**	**17**	2776	66.7	14	24
wcsp_spot5_log (21)	66.7	14	19	66.7	14	**17**
Total industrial (630)		435	50333		423	14426
Crafted domain (#instances)						
auctions (40)	60.0%	24	313	**95.0%**	**38**	1759
causal discovery (35)	**82.9%**	**29**	6851	17.1%	6	1357
CSG (10)	**100.0%**	**10**	825	80.0%	8	1370
frb (34)	**73.5%**	**25**	4298	50.0%	17	4041
min-enc (48)	66.7%	32	27	**75.0%**	**36**	455
miplib (12)	41.7%	5	**56**	41.7%	5	437
ramsey (15)	13.3%	2	1335	**20.0%**	**3**	2073
random-net (32)	**100.0%**	**32**	288	40.6%	13	5
set-covering (45)	**46.7%**	**21**	1530	20.0%	9	155
staff-scheduling (12)	16.7%	2	**1244**	16.7%	2	1406
wmaxcut (48)	**10.4%**	**5**	3651	6.3%	3	2960
Total crafted (331)		187	20417		140	16018

ever, at the same time MSCG used considerably more time per solved instance; this can be observed by inspecting the total cumulative running times: while PMRES+WCE uses 14416 s to solve 423 instances, MSCG15b uses a noticeable 35917 s more to solve additional 12 instances. Looking more closely at the results on a benchmark domain basis, we notice that the main advantage of MSCG15b is within the correlation clustering domain, where the solver also uses a noticeably amount of time to solve an additional 14 instances; furthermore, notice that the correlation clustering domain is over-represented among the full benchmark set with 129 instances. On the other hand, PMRES solves twice as many instances as MSCG15b within the relational inference domain, using at the same time only 43% of the cumulative running time of MSCG15b. The abstraction refinement domain provides another example where PMRES+WCE solves instances cumulatively noticeably faster than MSCG15b: here the solvers solve the same number of instances, but the cumulative running time of PMRES+WCE is less than 13% of that of MSCG15b (i.e., an 8x speed-up relative to MSCG15b). Turning to the crafted domains, we observe that MSCG15b clearly dominates on several of them. This is an interesting observation, also in that Eva500a never participated in the crafted MaxSAT evaluation categories.

6 WCE and Other SAT-Based MaxSAT Algorithms

In this section we discuss WCE in a more general setting and the question of to what extent it could be applied to other recently proposed MaxSAT algorithms.

The key to integrating WCE with a core-guided MaxSAT algorithm is whether clause cloning can be implemented through assumptions in the algorithm. As far as we understand, the reason clause cloning through assumptions can be added to PMRES is the fact that no clause ever appears in a core more than once. In more detail, let κ be a core extracted by PMRES during solving and assume $C \in \kappa$ needs to be cloned, i.e., that $w(C) > w_\kappa^{\min}$. Since the extended clause $C \vee \mathrm{A}(C)$ is added to the working formula as hard, that clause is not going to be extracted in any subsequent cores, but rather, only its clone $\mathrm{CL}(C)$ or some of the soft clauses added in $\mathrm{RELAX}(w_\kappa^{\min}, \mathcal{R})$. This simple observation is a key to implementing clause cloning through assumptions.

WPM1. As an example of an algorithm in which clause cloning seems to be difficult to implement through assumptions, consider the WPM1 algorithm [3,24]. WPM1 works similarly to PMRES in the sense that it uses a SAT solver to extract and relax unsatisfiable cores of the input instance F. Given a core κ, WPM1 clones its clauses similarly to PMRES and extends each clause $C_i \in \kappa$ (now of weight $w_\kappa^{\min}$) with a fresh relaxation variable r_i. For WCE, the key difference between PMRES and WPM1 is that WPM1 leaves all extended clauses $C_i \vee r_i$ as *soft* in the working formula and adds a cardinality constraint $\mathrm{CNF}(\sum r_i = 1)$ as *hard* clauses. Hence the extended clause might appear in subsequent cores, making it difficult if not impossible to also reuse it as its own clone.

Finally, we point out MaxSAT algorithms to which WCE can be integrated.

OLL and K. Two algorithms that closely resemble PMRES[3] are OLL [2,30] and K [1]. Both extract cores iteratively, harden and clone the clauses in cores, and compile them into the formula using soft cardinality constraints. In contrast to PMRES, OLL makes use of cardinality networks in order to dynamically modify the previously added cardinality constraints while K uses parametrized constraints for bounding their size. In both cases the clauses in the extracted cores are hardened and do not appear in subsequent cores, and as such WCE could be incorporated into both algorithms. Indeed, at least the K algorithm does implement clause cloning through assumptions [1]. The MSCG15b MaxSAT solver considered in Sect. 5 implements OLL.

WPM3 [7] maintains a set of at-most constraints, initialized to not allow any soft clauses to be falsified. During solving all clauses are treated as hard and the at-most constraints as soft, and hence all cores are subsets of these constraints. After finding a new core, WPM3 performs clause cloning and then merges the constraints to form new ones that make effective use of the global core structure. In contrast to OLL and PMRES, the cardinality constraints in the extracted cores are not hardened, but instead removed from the instance. To the best of our understanding, the version of WPM3 presented in [7] does not use the SAT solver iteratively, but instead rebuilds it on each iteration. Hence the idea of implementing clause cloning through assumptions is not applicable to this version of WPM3, even though it might be if the algorithm is extended with incremental cardinality constraints in the spirit of [28]. However, the discussed requirement of a clause in a core not appearing in any subsequent cores is satisfied by the algorithm. Thereby delaying the modification of cardinality constraints could be incorporated to the presented version of WPM3.

WMSU3 [27] maintains a single cardinality constraint $\sum_{r \in \mathcal{R}} r = \lambda$ over the set $\mathcal{R}$ of relaxation variables of clauses appearing in cores extracted so far. When a new core κ is extracted, all the assumption variables $\text{A}(C)$ of clauses $C \in \kappa$ are reused as relaxation variables, i.e., added into the set $\mathcal{R}$, after which a new bound λ is computed and the solver invoked again. Here λ is a lower bound on the optimal cost of the instance. As noted in Sect. 4, WCE can be viewed as an extension of the lower bounding technique from [18], the difference being that WMSU3 extended with WCE would perform such a core mining step in between each modification of the cardinality constraint, not only before the cardinality constraint is added.

SAT-IP Hybrids. Finally, also the SAT-IP hybrid solvers MaxHS [15] and LMHS [32], based on the implicit hitting set approach to MaxSAT, could potentially make use of specific ideas related to WCE. Specifically, WCE could be

[3] Originally in [31] PMRES was formalized as a special case of the so-called MAXRES [19] rule; the specific special case is equivalent to the formalization of PMRES used here.

incorporated into the *disjoint core extraction phase*—that is very important in terms of performance in practice [15]—in this context in a straight-forward way: instead of ruling out each clause C in a core κ from the working instance during the disjoint phase, lower the weight of all clauses in the core by $w_\kappa^{\min}$, and rule out only those clauses whose weight is lowered to 0.

7 Conclusions

We proposed weight-aware core extraction (WCE) as a refinement to the approach taken by various core-guided MaxSAT solvers for compiling the cores extracted at each iteration of search. WCE allows for extracting multiple cores of the same working formula by taking into account the residual weights of the current soft clauses, thereby postponing the compilation step and allowing the SAT solver to work on a less bloated working formula. We detailed WCE in the context of PMRES, a representative of the most recent line of core-guided MaxSAT solvers that use soft cardinality constraints in the compilation step, and showed empirically that WCE noticeably improves the performance of PMRES on standard weighted partial MaxSAT benchmarks. We also outlined how to integrate ideas behind WCE into other core-guided MaxSAT algorithms. The empirical results obtained for PMRES suggests that integrating WCE into other recent MaxSAT solvers may provide further improvements to the state of the art.

References

1. Alviano, M., Dodaro, C., Ricca, F.: A MaxSAT algorithm using cardinality constraints of bounded size. In: Proceedings of IJCAI, pp. 2677–2683. AAAI Press (2015)
2. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: Proceedings of ICLP Technical Communications, LIPIcs, vol. 17, pp. 211–221. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)
3. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial MaxSAT through satisfiability testing. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 427–440. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02777-2_39
4. Ansótegui, C., Bonet, M., Levy, J.: SAT-based MaxSAT algorithms. Artif. Intell. **196**, 77–105 (2013)
5. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving SAT-based weighted MaxSat solvers. In: Milano, M. (ed.) CP 2012. LNCS, pp. 86–101. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33558-7_9
6. Ansótegui, C., Didier, F., Gabàs, J.: Exploiting the structure of unsatisfiable cores in MaxSAT. In: Proceedings of IJCAI, pp. 283–289. AAAI Press (2015)
7. Ansótegui, C., Gabàs, J., Levy, J.: Exploiting subproblem optimization in SAT-based MaxSAT algorithms. J. Heuristics **22**(1), 1–53 (2016)
8. Argelich, J., Le Berre, D., Lynce, I., Marques-Silva, J., Rapicault, P.: Solving Linux upgradeability problems using Boolean optimization. In: Proceedings of LoCoCo. Electronic Proceedings in Theoretical Computer Science, vol. 29, pp. 11–22 (2010)

9. Argelich, J., Li, C.M., Manyà, F., Planes, J.: MaxSAT Evaluations. http://maxsat.ia.udl.cat/
10. Audemard, G., Lagniez, J.-M., Simon, L.: Improving glucose for incremental SAT solving with assumptions: application to MUS extraction. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 309–317. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39071-5_23
11. Berg, J., Järvisalo, M., Malone, B.: Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In: Proceedings of AISTATS, JMLR Workshop and Conference Proceedings, vol. 33, pp. 86–95 (2014). JMLR.org
12. Bjørner, N., Narodytska, N.: Maximum satisfiability using cores and correction sets. In: Proceedings of IJCAI, pp. 246–252. AAAI Press (2015)
13. Bunte, K., Järvisalo, M., Berg, J., Myllymäki, P., Peltonen, J., Kaski, S.: Optimal neighborhood preserving visualization by maximum satisfiability. In: Proceedings of AAAI, pp. 1694–1700. AAAI Press (2014)
14. Chen, Y., Safarpour, S., Marques-Silva, J., Veneris, A.: Automated design debugging with maximum satisfiability. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(11), 1804–1817 (2010)
15. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MaxSaT. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 166–181. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39071-5_13
16. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006). doi:10.1007/11814948_25
17. Guerra, J., Lynce, I.: Reasoning over biological networks using maximum satisfiability. In: Milano, M. (ed.) CP 2012. LNCS, pp. 941–956. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33558-7_67
18. Heras, F., Morgado, A., Marques-Silva, J.: Lower bounds and upper bounds for MaxSAT. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, pp. 402–407. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34413-8_35
19. Larrosa, J., Heras, F.: Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In: Proceedings of IJCAI, pp. 193–198. Professional Book Center (2005)
20. Li, C.M., Manyà, F., Mohamedou, N.O., Planes, J.: Resolution-based lower bounds in MaxSAT. Constraints **15**(4), 456–484 (2010)
21. Li, C.M., Manyà, F., Planes, J.: Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 403–414. Springer, Heidelberg (2005). doi:10.1007/11564751_31
22. Li, C.M., Manyà, F., Planes, J.: Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In: Proceedings of AAAI, pp. 86–91. AAAI Press (2006)
23. Lin, H., Su, K., Li, C.M.: Within-problem learning for efficient lower bound computation in Max-SAT solving. In: Proceedings of AAAI, pp. 351–356. AAAI Press (2008)
24. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted Boolean optimization. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 495–508. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02777-2_45
25. Marques-Silva, J., Janota, M., Ignatiev, A., Morgado, A.: Efficient model based diagnosis with maximum satisfiability. In: Proceedings of IJCAI, pp. 1966–1972. AAAI Press (2015)

26. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Ann. Math. Artif. Intell. **62**(3–4), 317–343 (2011)
27. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. CoRR abs/0712.1097 (2007)
28. Martins, R., Joshi, S., Manquinho, V., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 531–548. Springer, Cham (2014). doi:10.1007/978-3-319-10428-7_39
29. Martins, R., Manquinho, V., Lynce, I.: Open-WBO: a modular MaxSAT solver'. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 438–445. Springer, Cham (2014). doi:10.1007/978-3-319-09284-3_33
30. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 564–573. Springer, Cham (2014). doi:10.1007/978-3-319-10428-7_41
31. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: Proceedings of AAAI, pp. 2717–2723. AAAI Press (2014)
32. Saikko, P., Berg, J., Järvisalo, M.: LMHS: a SAT-IP hybrid MaxSAT solver. In: Creignou, N., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 539–546. Springer, Cham (2016). doi:10.1007/978-3-319-40970-2_34
33. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Siekmann, J.H., Wrightson, G. (eds.) Automation of Reasoning. 2: Classical Papers on Computational Logic 1967–1970. Symbolic Computation, pp. 466–483. Springer, Heidelberg (1983)
34. Zhu, C., Weissenbacher, G., Malik, S.: Post-silicon fault localisation using maximum satisfiability and backbones. In: Proceedings of FMCAD, pp. 63–66. FMCAD Inc. (2011)