Core Boosting in SAT-Based Multi-Objective Optimization

Christoph Jabs, Jeremias Berg, and Matti Järvisalo

HIIT, Department of Computer Science, University of Helsinki, Finland
christoph.jabs@helsinki.fi

Abstract. Maximum satisfiability (MaxSAT) constitutes today a successful approach to solving various real-world optimization problems through propositional encodings. Building on this success, approaches have recently been proposed for finding Pareto-optimal solutions to multi-objective MaxSAT (MO-MaxSAT) instances, i.e., propositional encodings under multiple objective functions. In this work, we propose *core boosting* as a reformulation/preprocessing technique for improving the runtime performance of MO-MaxSAT solvers. Core boosting in the multi-objective setting allows for shrinking the ranges of the multiple objectives at hand, which can be particularly beneficial for MO-MaxSAT relying on search that requires enforcing increasingly tighter objective bounds through propositional encodings. We show that core boosting is effective in improving the runtime performance of SAT-based MO-MaxSAT solvers typically with little overhead.

Keywords: Multi-objective optimization · maximum satisfiability · core boosting · preprocessing.

1 Introduction

Maximum satisfiability (MaxSAT) [5], the optimization extension of Boolean satisfiability (SAT) [12], has developed from a theoretical tool into a competitive practical constraint optimization paradigm. This is in particular due to noticeable algorithmic advances in practical MaxSAT algorithms developed in recent years based on the iterative use of SAT solvers. Today, MaxSAT solvers are successfully employed to efficiently solve large instances of various types of real-world NP-hard combinatorial optimization problems via propositional encodings under a single objective function.

Building on advances in (single-objective) MaxSAT solving, algorithmic advances have been recently made towards developing increasingly effective solvers for the more general and challenging realm of MaxSAT under multiple objectives, i.e., multi-objective MaxSAT (MO-MaxSAT) [44,45,28,14,23,15]. Motivated through practical applications that give rise in a natural way to propositional encodings of optimization problems under multiple objectives, the goal in MO-MaxSAT solving is to efficiently enumerate all Pareto-optimal—as a standard notion of optimality in the multi-objective setting—solutions (or, more precisely, a representative Pareto-optimal solution for each point in the so-called non-dominated set within the search space of all solutions). Pareto-optimal solutions are solution with respect to which no objective can be improved without making the solution worse in terms of another objective, hence intuitively constituting the best possible solutions in general terms under multiple objectives.

A complementary approach to improving constraint solvers by developing more effective algorithms is that of developing preprocessing (or reformulation) techniques to be applied before calling a solver. The aim of (effective) preprocessing is to improve solver runtimes to the extent that the additional time spent in preprocessing is worthwhile in terms of the combined overall time spent in preprocessing and solving, compared to the time required to directly solve the original problem instance. Preprocessing has been highly influential in SAT solving [13], and motivated by this, extensions of SAT preprocessing have been proposed for MaxSAT [7,10,25,42] and most recently for MO-MaxSAT [27]. However, so-far liftings of (Max)SAT preprocessing techniques to the realm of MO-MaxSAT have turned out to provide relatively small runtime improvements for current state-of-the-art MO-MaxSAT solvers. This suggests that more research is called for towards harnessing the full potential of preprocessing for speeding up MO-MaxSAT solving.

In this work, we propose core boosting as an approach to automatically reformulating MO-MaxSAT instances. Core boosting was earlier proposed in the context of single-objective MaxSAT solving [8] with later applications in single-objective core-guided constraint programming [22] and pseudo-Boolean optimization [16]. In the previous works, core boosting was proposed as an anytime algorithm that combines so-called core-guided and upper-bounding search for finding good solutions to single-objective constraint optimization instances within a stringent runtime limit. In contrast, we develop core boosting for the multi-objective setting as a pre-solving phase technique that allows for reformulating an MO-MaxSAT instance by tightening its search space, in particular via detecting inconsistent parts of the search space which can be subsequently ignored by MO-MaxSAT solvers. This is achieved—in short, as we will later on explain in more detail—by increasing objective offsets in a given MO-MaxSAT instance to match the so-called ideal point of the multi-objective search space without removing any Pareto-optimal solutions. As such core boosting can be viewed as a preprocessing technique which significantly differs from more typical (Max)SAT-based preprocessing techniques so-far studied for MO-MaxSAT. Core boosting intuitively leads to enabling more optimized propositional encodings of pseudo-Boolean constraints used within state-of-the-art MO-MaxSAT solvers. We explain in detail how core boosting can be tightly integrated into MO-MaxSAT solvers, and provide an open-source implementation in conjunction with three recently-proposed algorithms for MO-MaxSAT. Empirically, it turns out that core boosting can be highly effective in speeding up overall runtimes of MO-MaxSAT algorithms, having a noticeably greater positive impact on runtime performance than presently available MaxSAT-based preprocessing techniques for MO-MaxSAT.

2 Multi-Objective MaxSAT

For Boolean variable x, there are two literals: the positive x and the negative $\neg x$. A clause is a disjunction of literals, and a (CNF) formula a conjunction of clauses. When convenient, we view a clause as the set of literals in the clause, and a formula as the set of its clauses. An assignment τ maps variables to $\{0,1\}$, i.e., $\tau(x) = 1$ (true) or $\tau(x) = 0$ (false). Assignments extend to literals, clauses, and formulas by $\tau(\neg x) = 1 - \tau(x)$ for negative literal $\neg x$, $\tau(C) = \max\{\tau(l) \mid l \in C\}$ for clause C, and $\tau(F) = \min\{\tau(l) \mid C \in F\}$ for formula F. An assignment for which $\tau(F) = 1$ is a *solution* to F. If a formula has a solution, the formula is satisfiable, otherwise the formula is unsatisfiable.

A pseudo-Boolean (PB) expression $O = (\sum_i c_i \cdot l_i) + o$ is a sum of terms—each consisting of a literal l_i and a positive integer constant c_i—and a non-negative integer constant o referred to as *offset* of O. We denote the set of literals appearing in O by $\textsc{lits}(O)$. The value under O of an assignment τ over $\textsc{lits}(O)$ is $O(\tau) = (\sum_i c_i \tau(l_i)) + o$. For an integer B, a pseudo-Boolean constraint $O \leq B$ is satisfied by an assignment τ if $O(\tau) \leq B$. Our work makes extensive use of CNF encodings that encode values of PB constraints into literals [6,18,30]. More precisely, $\textsc{Cnf}(O \leq B)$ is a CNF formula that defines a literal $\langle O \leq B \rangle$ such that any solution τ of $\textsc{Cnf}(O \leq B)$ sets $\tau(\langle O \leq B \rangle) = 1$ if and only if τ satisfies $O \leq B$[1]. When clear from context, $\langle O \leq B \rangle$ should be understood as $\textsc{Cnf}(O \leq B) \wedge \langle O \leq B \rangle$. For example, the formula $F \wedge \langle O \leq B \rangle$ stands for the formula $F \wedge \textsc{Cnf}(O \leq B) \wedge \langle O \leq B \rangle$, the solutions of which are the solutions τ of F that satisfy $O \leq B$. We also use $\langle O < B \rangle$ as a shorthand for $\langle O \leq B - 1 \rangle$, and $\langle O \geq B \rangle$ as a shorthand for $\neg \langle O < B - 1 \rangle$.

We focus on the following natural extension of maximum satisfiability to the multi-objective setting. An instance $\mathcal{I} = (F, \mathcal{O})$ of multi-objective maximum satisfiability (MO-MaxSAT) consists of a formula F and p linear objective functions $\mathcal{O} = (O_1, \ldots, O_p)$ represented as pseudo-Boolean expressions. Note that this definition covers single-objective MaxSAT with $p = 1$. Any solution τ to F is a solution to $\mathcal{I}$. A solution τ has cost $\mathcal{O}(\tau) = (O_1(\tau), \ldots, O_p(\tau))$ with respect to $\mathcal{I}$, and cost $O_i(\tau)$ with respect to objective O_i. The ideal point [19] $(\gamma_1, \ldots, \gamma_p)$ of $\mathcal{I}$ consists of the smallest value for each objective over the solutions to $\mathcal{I}$, i.e., $\gamma_i = \min\{O_i(\tau) \mid \tau(F) = 1\}$. We focus on the task of computing Pareto-optimal solutions to MO-MaxSAT instances. A solution τ dominates another solution τ' if $O_i(\tau) \leq O_i(\tau')$ for all $i = 1, \ldots, p$ and $O_i(\tau) < O_i(\tau')$ for some i. A solution τ is *Pareto-optimal* if it is not dominated by any solution to $\mathcal{I}$. The costs of Pareto-optimal solutions form the non-dominated set of $\mathcal{I}$.

Example 1. Consider the bi-objective instance on the left in Fig. 1. The infeasible region with respect to the objectives, i.e., the objective values for which no solutions to F exist, is illustrated on the right. The non-dominated set of the instance is $\{(4,8), (5,7), (6,3)\}$ and its ideal point $(4,3)$. □

---

[1] In practice, the algorithms considered in this work often employ an implication relationship—$\tau(\langle O \leq B \rangle) = 1$ *if* τ satisfies $O \leq B$—rather than the mentioned equivalence. We assume equivalences for simplicity without loss of generality.

$$
\begin{aligned}
F =& \{(x_1 \vee x_2 \vee x_3), (x_1 \vee x_2 \vee x_4), \\
& (x_2 \vee x_3 \vee x_5), (x_3 \vee x_4 \vee x_5)\} \\
O_1 =& 3x_2 + 4x_3 + 2x_4 + 5x_5 \\
O_2 =& 7x_1 + 4x_2 + 1x_3 + 2x_4
\end{aligned}
$$

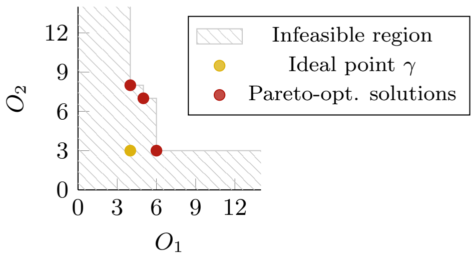

Fig. 1. A bi-objective MaxSAT instance and its infeasible region with respect to the objectives.

While we will present core boosting in the context of computing a single representative solution to each element in the non-dominated set, we note that the technique is also applicable when computing *every* Pareto-optimal solution. In particular, unlike other specific preprocessing techniques [27] core boosting maintains all solutions for each element in the non-dominated set.

2.1 SAT-based MO-MaxSAT Algorithms

Similarly as single-objective MaxSAT algorithms, multi-objective MaxSAT algorithms [44,45,28,14,23,15] make extensive use of *SAT solvers* [36], i.e., decision procedures that either compute a solution of a CNF formula, or determine that the formula is unsatisfiable, i.e., that no such solutions exist. For computing Pareto-optimal solutions, SAT solvers are used to iteratively compute solutions of the instance. When a new solution is found new constraints that rule out dominated solutions from consideration are added until no more solutions remain, at which point the entire non-dominated set has been discovered.

A close analogy in the single-objective case is the Sat-Unsat (LSU) [5] algorithm for minimizing a single objective O subject to a CNF formula F. This algorithm is implemented by various single-objective MaxSAT solvers [18,11,32,46]. Starting from some upper bound UB, LSU minimizes O by incrementally invoking a SAT solver on the formula $F \wedge \langle O < \text{UB}\rangle$. If the SAT solver finds a new solution τ, we have $O(\tau) < \text{UB}$ and hence the upper bound is improved. If the solver reports unsatisfiability, the latest solution found is optimal and the algorithm terminates. With this intuition, we next detail three state-of-the-art algorithms for computing Pareto-optimal solutions: P-Minimal, BiOptSat, and LowerBound.

P-Minimal [44,31] can be seen as multi-objective LSU. When solving an instance of MO-MaxSAT $(F, (O_1, \ldots, O_p))$, P-Minimal iteratively invokes a SAT solver on a working formula consisting of F and additional constraints added in previous iterations. When the SAT solver returns a solution τ, P-Minimal (i) blocks all solutions of worse quality, i.e., the ones dominated by τ, and (ii) restricts search in subsequent iterations to solutions dominating τ. Part (i) is achieved by adding the constraints $\bigvee_{i=1}^{p} \langle O_i < O_i(\tau)\rangle$ that enforce subsequent

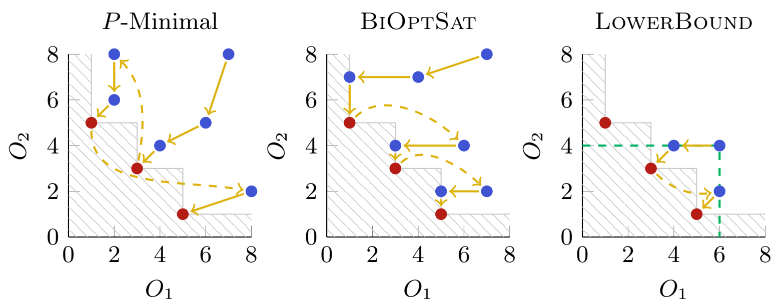

Fig. 2. Search trajectories of P-Minimal (left), BiOptSat (middle), and LowerBound (right) in terms of objective values.

solutions to improve in at least one objective. Part (ii) is achieved by adding the constraints $\bigwedge_{i=1}^{p} \langle O_i \leq O_i(\tau)\rangle$ that enforce subsequent solutions to not worsen in any objective. When the SAT solver reports unsatisfiability, all constraints of type (ii) are removed and search continues. When no more solutions can be found without any constraints of type (ii), all Pareto-optimal solutions have been found. Fig. 2 (left) illustrates one possible search trajectory of P-Minimal (with respect to objective values) on a bi-objective instance. Assume search starts at a solution with objective values $(7, 8)$. By iteratively steering search to regions that dominate the current solution, P-Minimal moves through intermediate solutions (marked in blue) until it discovers the (red) non-dominated point at $(3, 3)$, after which the SAT solver reports unsatisfiability. Then, all constraints of type (ii) are removed and P-Minimal starts the minimization procedure again (illustrated by the dashed line) while retaining the blocking constraints (i). After three minimization procedures, the entire non-dominated set in Fig. 2 is discovered and the algorithm terminates since all solutions are blocked.

BiOptSat in its Sat-Unsat variant [28] computes the non-dominated set of a bi-objective MaxSAT instance $(F, (O_1, O_2))$ via the so-called lexicographic method [34]. Assuming the same initial solution with objective values $(7, 8)$, BiOptSat starts by employing single-objective LSU to find a solution τ minimizing O_1, as illustrated in Fig. 2 (middle) by arrows going leftward until the blue solution on the infeasibility boundary is reached. Next, O_2 is minimized while restricting O_1 to at most $O_1(\tau)$, i.e., subject to $F \wedge \langle O_1 \leq O_1(\tau)\rangle$, using LSU, illustrated in the figure by the downward arrows, until the red Pareto-optimal solution τ^p is found. To find the next Pareto-optimal solution, BiOptSat adds $\langle O_2 < O_2(\tau^p)\rangle$ to F and reiterates. This is illustrated by the dashed arrows in Fig. 2. The algorithm terminates when no solutions remain, indicated by the SAT solver reporting unsatisfiability after removing the constraints on O_1.

LowerBound [15], in contrast to P-Minimal and BiOptSat, mainly performs lower-bounding search to compute the non-dominated set. It maintains a *fence*, i.e., a tuple $(\lambda_0, \ldots, \lambda_p)$ of values, initialized to $(0, \ldots, 0)$, that represents the greatest objective values currently considered. During search LowerBound alternates between iteratively loosening the fence until the region bounded by the constraints $\bigwedge_{i=1}^{p} \langle O_i \leq \lambda_i \rangle$ contains feasible solutions, and then employing P-Minimal to find all elements of the non-dominated set "inside" the current fence. The search of P-Minimal inside a fence is illustrated in Fig. 2 (right) for the fence shown in green. After P-Minimal finds all Pareto-optimal solutions within the fence, the fence is loosened further. The algorithm terminates once all solutions have been blocked by P-Minimal.

3 Core Boosting for MO-MaxSAT

We now detail core boosting for MO-MaxSAT as our main contribution.

3.1 Effects of Core Boosting

Before describing how core boosting is realized, we explain how core boosting allows for reducing the search space of MO-MaxSAT instances and detail how core boosting reformulates MO-MaxSAT instances.

Core boosting is a technique that through reformulating an MO-MaxSAT instance increases the offsets of the objectives of the instance to match the ideal point without removing any Pareto-optimal solutions. As such core boosting can be viewed as a preprocessing technique which significantly differs from more typical (Max)SAT-based preprocessing techniques recently proposed for MO-MaxSAT [27]. The intuition for the potential usefulness of the core boosting reformulation stems from the fact that MO-MaxSAT algorithms such as P-Minimal, BiOptSat, and LowerBound search only over the non-constant parts of the objectives in the instance: the range of possible solution costs that the algorithms consider during search is bounded "from below" by the point consisting of the offsets of each objective, and "from above" by the point consisting of the maximum value of each objective. As such, increasing the offsets of the objectives conceptually leads to a smaller search space.

Example 2. Recall the bi-objective MaxSAT instance from Example 1 and Fig. 1. For this instance, the range of solution costs that P-Minimal, BiOptSat, and LowerBound consider during search is 0 to 12 for both O_1 and O_2, as illustrated on the left in Fig. 3. Applying core boosting on this instance results in a reformulation with the same Pareto-optimal solutions and objectives O_1^{cb}, O_2^{cb} with offsets $o_1^{\text{cb}} = 4$ and $o_2^{\text{cb}} = 3$, respectively. When solving the reformulated instance, MO-MaxSAT algorithms are effectively searching over the costs in the range $4 \ldots 12$ for O_1 and $3 \ldots 12$ for O_2. This search space (depicted on the right in Fig. 3) is smaller than the one that would be considered without core boosting. In particular, after core boosting, the cross-hatched area shown in Fig. 3 does not need to be considered during search. □

Formally, core boosting transforms an instance $\mathcal{I} = (F, \mathcal{O})$ with an ideal point $(\gamma_1, \ldots, \gamma_p)$ into a reformulated ("core-boosted") instance $\mathcal{I}^{\text{cb}} = (F^{\text{cb}}, \mathcal{O}^{\text{cb}})$ for which the following hold.

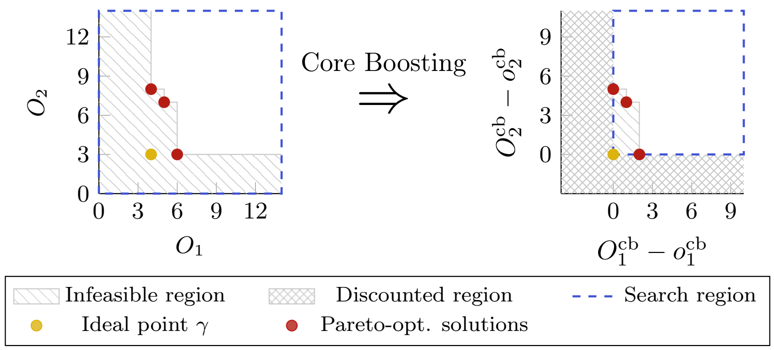

Fig. 3. Illustration on how core boosting shifts the point where search is anchored to the ideal point and reduces the search space.

(i) All solutions of F^{cb} are solutions to F, and any solution to F can be uniquely extended into a solution to F^{cb}.
(ii) $\mathcal{O}(\tau) = \mathcal{O}^{\text{cb}}(\tau)$ for all solutions τ to F^{cb}.
(iii) The offset of objective O_i^{cb} is γ_i.

In other words, core boosting reformulates a given MO-MaxSAT instance in a way that all solutions and their costs are preserved, and the offset of each objective O_i is increased to γ_i, the ith coordinate of its ideal point.

When viewed as a lower-bounding method, the offsets that core boosting derives for each objective are as high as possible while guaranteeing that all Pareto-optimal solutions and the non-dominated set are preserved. More precisely, consider an MO-MaxSAT instance $\mathcal{I} = (F, (O_1, \dots, O_p))$, its ideal point $(\gamma_1, \dots, \gamma_p)$ and fix an index i. By definition, there is a Pareto-optimal solution τ for which $O_i(\tau) = \gamma_i$. Since the coefficients of objectives are positive, any reformulation $\mathcal{I}^{\text{ref}} = (F^{\text{ref}}, (O_1^{\text{ref}}, \dots O_p^{\text{ref}}))$ of $\mathcal{I}$ in which the offset of O_i^{ref} is strictly greater than γ_i will have a different non-dominated set, and specifically the cost of τ will be different. In the context of algorithms computing the entire non-dominated set, core boosting therefore derives the tightest lower bound given by a single point.[2]

3.2 Core Boosting via Single-Objective Core-Guided Search

We now detail how the reformulation performed by core boosting can be realized in practice via single-objective lower-bounding search based on so-called unsatisfiable cores, i.e., using core-guided MaxSAT search [37,2,1,40,41]. We detail core boosting in pseudocode as Algorithm 1. Invoked on an MO-MaxSAT instance $(F, \mathcal{O})$, core boosting iteratively invokes single-objective core-guided lower-bounding search (represented in pseudocode by the CoreGuided subprocedure) on single-objective MaxSAT instances. In the ith iteration, Core

[2] Exploring similar ideas from the perspective of so-called lower bound sets [20] constitutes interesting future work beyond the scope of this paper.

Algorithm 1 Core boosting for MO-MaxSAT

Input: An MO-MaxSAT instance $\mathcal{I} = (F, (O_1, \ldots, O_p))$
Output: A reformulated MO-MaxSAT instance $\mathcal{I}^{\text{cb}}$

1: $F^{\text{cb}} \leftarrow F$
2: **for** $i \leftarrow 1$ **to** p **do**
3: $\quad (F^{\text{cb}}, O_i^{\text{cb}}) \leftarrow \textsc{CoreGuided}(F^{\text{cb}}, O_i)$
4: **return** $(F^{\text{cb}}, (O_1^{\text{cb}}, \ldots, O_p^{\text{cb}}))$

Algorithm 2 CoreGuided

Input: A single-objective MaxSAT instance $\mathcal{I} = (F, (O))$
Output: An optimal solution τ to $\mathcal{I}$ and a reformulated instance $\mathcal{I}'$

$F^{\text{ref}} \leftarrow F, \quad O^{\text{ref}} \leftarrow O$
while true **do**
$\quad (\text{res}, \kappa, \tau) \leftarrow \textsc{ExtractCore}(F^{\text{ref}}, O^{\text{ref}})$
$\quad$ **if** res = "unsatisfiable" **then**
$\quad\quad (F^{\text{ref}}, O^{\text{ref}}) \leftarrow \textsc{Reformulate}(F^{\text{ref}}, O^{\text{ref}}, \kappa)$
$\quad$ **else**
$\quad\quad$ **return** $\tau, (F^{\text{ref}}, (O^{\text{ref}}))$

Guided is invoked on F^{cb} and O_i (line 3), adding new clauses to F^{cb} and reformulating O_i to O_i^{cb}. The formula F^{cb} consists of the clauses of the original instance F and all additional constraints added by CoreGuided in previous iterations.

Algorithm 2 details a generic abstraction of core-guided search under a single objective. The algorithm works by iteratively extracting so-called (unsatisfiable) cores based on which the instance is reformulated. A core κ of a single-objective MaxSAT instance $\mathcal{I} = (F, (O))$ is a subset of objective literals $\kappa \subset \textsc{lits}(O)$ out of which at least one literal has to incur cost, i.e., has to be assigned to 1. Such a core can be obtained with a modern off-the-shelf SAT solver by employing its assumption interface [17,36]. This core extraction is done in the ExtractCore subroutine which takes a formula F and an objective O as input and returns a triple $(\text{res}, \kappa, \tau)$ where res indicates whether $F' = F \wedge \bigwedge_{l \in \textsc{lits}(O)} \neg l$ is satisfiable. If res = "unsatisfiable", κ contains a core of $(F, (O))$, otherwise τ contains a solution to F'.

When a new core is extracted, the instance is reformulated by the Reformulate subroutine. Existing core-guided algorithms differ mainly in the details of how Reformulate is instantiated. Core boosting makes very lightweight assumptions on the underlying core-guided algorithm. It can be realized with any core-guided algorithm whose instantiation of Reformulate increases the offset of the objective, decreases the sum of coefficients in the objective of the literals in the core, and adds additional clauses and variables to preserve the solutions and their costs in the instance. More specifically, the properties of Reformulate required for core boosting can be summarized as follows. Assume that Reformulate is invoked with formula F, objective O, and core κ, and that it returns a new formula F^{ref} and objective O^{ref}. Then the following must hold for core-boosting to be applicable: (i) Every solution of F^{ref} is a solution of F; (ii) $O(\tau) = O^{\text{ref}}(\tau)$ holds for all solutions of F^{ref}; (iii) the sum of coefficients in O^{ref} is smaller than in O; and (iv) the offset of O^{ref} is greater than the offset of O. It should be noted that these properties are met by practically all modern core-guided algorithms [21,37,2,1,40,41,25,43,26].

As a side-remark, each reformulation performed by all core-guided algorithms we are aware of increases the offset of the objective, and decreases the sum of coefficients, exactly by the minimum coefficient of the literals in the core. This is because at least one literal in the core has to incur cost. Thus, the smallest possible cost incurred due to a core matches the smallest coefficient among the literals in the core.

Example 3. Invoke CoreGuided on the constraints and objective O_1 of the MO-MaxSAT instance from Fig. 1. Let the cores extracted in the first two iterations of executing CoreGuided on $(F, (O_1))$ be $\kappa^1 = \{x_2, x_3, x_5\}$ and $\kappa^2 = \{x_3, x_4, x_5\}$. After reformulating these two cores, the reformulated instance is satisfiable at line 4. The smallest coefficients in the cores are $c_{\kappa^1} = 3$ and $c_{\kappa^2} = 1$, respectively. The final reformulated objective has a constant offset of $o_1^{\text{cb}} = c_{\kappa^1} + c_{\kappa^2} = 4$ with its coefficient sum reduced by the offset. □

Note that core boosting for MO-MaxSAT, as proposed here, differs from core boosting for (single-objective) MaxSAT [8]. For MaxSAT, CoreGuided is executed under a heuristically determined time limit since core-guided search is complete for MaxSAT. In contrast, in the multi-objective setting running CoreGuided without resource limits will *not* fully solve the instance (assuming that the objectives conflict with each other in that their minimum values correspond to different solutions). Instead, the search space is reduced with respect to the individual objectives.

3.3 Realizing Core Boosting

A variety of core-guided single-objective MaxSAT algorithms from the literature [37,2,1,40,41] could be employed for practical implementations of core boosting. We detail here our implementation of core boosting based on the effective core-guided algorithm OLL [40,1].

Informally speaking, OLL instantiates Reformulate by introducing PB constraints over the literals in the extracted cores in a way that systematically allows additional literals to be assigned to 1 in subsequent iterations. More precisely, after obtaining a core κ, OLL (i) decreases the coefficient of each $l \in \kappa$ by c_κ, the smallest coefficient among the literals in κ, removing l from the objective if the new coefficient is 0 (this process is called clause cloning in some references [9]); and (ii) adds new variables $\langle \sum_{l \in \kappa} l \geq k \rangle$ to the reformulated objective with the coefficient c_κ and constraints $\textsc{Cnf}(\sum_{l \in \kappa} l \leq k)$ to the formula for $k = 2, \ldots, |\kappa|$. Conceptually, step (i) relaxes the current objective by removing at least one literal from the objective and allowing at least one literal in κ

to be assigned to 1 in subsequent iterations. Step (ii) adds constraints to ensure that at most one literal can be set to 1 without new cores being discovered, thus ensuring the preservation of optimal solutions.

An important intuition for understanding the effects of core boosting is that the changes in the number of literals in the objective depend on the coefficients in the extracted cores. If the coefficients in the variables of a core κ are not equal, not all literals will be removed from the objective in step (i). Since OLL introduces $|\kappa| - 1$ new variables, in those cases the number of literals in the reformulated objective can increase. In contrast, if the coefficients of the variables are all equal, the number of variables in the objective will decrease by one after reformulation as then all variables in the core are removed and $|\kappa| - 1$ variables are introduced.

Example 4. Recall Example 3 where O_1 (see Fig. 1) was reformulated based on the cores κ^1 and κ^2. The detailed objective reformulated by OLL is $O_1^{\text{cb}} = x_4 + 4x_5 + 3\langle\kappa_1 \geq 2\rangle + 3\langle\kappa_1 \geq 3\rangle + \langle\kappa_2 \geq 2\rangle + \langle\kappa_2 \geq 3\rangle$ and the clauses $\textsc{Cnf}(\kappa_1 \leq k) \wedge \textsc{Cnf}(\kappa_2 \leq k)$ are added to the formula. □

Core boosting can be tightly integrated with SAT-based MO-MaxSAT algorithms in a way that allows for reusing the structure introduced by the core-guided algorithm employed for core boosting in the subsequent MO-MaxSAT search. More specifically, both the implementation of OLL and our implementations of P-Minimal, BiOptSat, and LowerBound realize their PB constraints using (generalized) totalizers [6,39,30]. For a PB expression O, a totalizer realizes $\textsc{Cnf}(O \leq B)$ by first partitioning O into subsets of size 1 and then iteratively merging partitions by adding extra clauses and variables that count the sum of coefficients of the terms in the partitions to be merged. The merging stops when there is a single partition left, at which point the new variables obtained in the last step correspond to $\langle O \leq k\rangle$ for all $k = 1, \ldots, B$. For an alternative view, the structure of $\textsc{Cnf}(O \leq B)$ created with a totalizer encoding can be visualized as a binary tree. The leaves of the tree correspond to the terms in O. Each internal node corresponds to new variables that count the sum of weights of the terms in the leaves of the subtree rooted at that node set to 1 by satisfying assignments.

When building a totalizer over the reformulated objective O^{ref} obtained after core boosting, our implementation makes use of the fact that some variables already correspond to the roots of other totalizers introduced by OLL. Instead of treating those as leaves in the new totalizer, we instead treat them as internal nodes, thus avoiding redundancy in the encoding which would be incurred by "recounting" the counting variables introduced by OLL in the pseudo-Boolean constraint over O^{ref} used by P-Minimal, BiOptSat, or LowerBound.

3.4 Core Boosting and MO-MaxSAT Solver Interactions

Finally, in addition to decreasing the range of objectives that algorithms need to search over, we identify two further interactions between core boosting and MO-MaxSAT algorithms.

The first relates to the number of clauses in a PB constraint $\textsc{Cnf}(O \leq B)$ built over an objective $O = \sum_i (c_i \cdot l_i) + o$ by MO-MaxSAT algorithms such as P-Minimal, BiOptSat, and LowerBound. For CNF encodings used in practice—including the totalizer we use—the number of clauses in $\textsc{Cnf}(O \leq B)$ depends on B and either on the sum of coefficients or the number of unique sums that can be obtained from the coefficients [30,18]. The effect of core boosting on these properties will depend on the specific instance being solved. Due to the properties of typical instantiations of Reformulate (recall Section 3.2), core boosting is guaranteed to decrease the sum of coefficients of each objective. In contrast (recall Section 3.3), the effect of core boosting on the number of variables in the objectives and the subset sums that can be obtained from them will depend on the coefficients of the variables in the extracted cores. Finally, another important contrast between single and multi-objective core boosting is that the intermediate solutions obtained during invocations of core-guided search on the separate objectives can not be used as global upper bound for the objectives by the MO-MaxSAT algorithm. Thus, we expect the effect of core boosting on the number of clauses introduced by the subsequent MO-MaxSAT algorithms to be more limited than in the single-objective case.

As a second interaction, core boosting can alter the structure of the PB constraints built over the reformulated objective by the MO-MaxSAT algorithms. In the case of totalizers this structure is defined by the ordering of the leaves in the totalizer tree and the structure of the tree itself. It is well-known that the structure of PB encodings employed can have a significant impact on the performance of constraint optimization algorithms [3,4]. While fundamental understanding on exactly how the structure of a totalizer affects the performance of constraint optimization algorithms is lacking, conventional wisdom based on empirical evaluation suggests that it is beneficial to place interrelated variables "close" in the tree. Core boosting achieves an approximation of this in the core-guided phase where totalizer tree substructures are built that have variables appearing together in cores as leaves.

4 Empirical Evaluation

We empirically evaluate the impact of core boosting. We integrated core boosting into three recent MO-MaxSAT algorithms, P-Minimal [44,31], BiOptSat [28], and LowerBound [15] using their implementations in the MO-MaxSAT solver Scuttle [27]. The implementation and benchmarks used in our evaluation, as well as full empirical data, are available in open source (https://bitbucket.org/coreo-group/scuttle). All experiments reported on were run on 2.40-GHz Intel Xeon Gold 6148 CPUs with 381-GB RAM in RHEL under a 1.5-hour per-instance time and 32-GB memory limit. Whenever core boosting was applied, the reported runtimes include the time spent in both core boosting and the MO-MaxSAT solver.

We use benchmark instances from seven domains from earlier evaluations of MO-MaxSAT solvers: multi-objective set covering with fixed set cardinality (set-cover-sc) and fixed element probability (set-cover-ep) [28], learning interpretable decision rules (lidr) [33], the flying tourist problem (ftp) [35], package upgradeability (packup) [29], staff shift scheduling (shiftdesign) [38], and satellite photograph scheduling (spot5) [24]. Set-cover-sc and set-cover-ep, instances with two objectives were obtained from [28] and instances with 3–5 objectives were generated similarly following [28]. The lidr instances contain 2 objectives and were also obtained from [28]. The ftp instances containing 2 objectives were obtained from [15] as instances with pseudo-Boolean constraints and encoded with the (generalized) totalizer encoding [6,30]. Package upgradeability instances were obtained from Mancoosi International Solver Competition of years 2011 and 2012 (https://www.mancoosi.org/misc/), and encoded with PackUp [29] with all combinations of 2–5 of the 5 minimization objectives. The shiftdesign and spot5 benchmarks were obtained from MaxSAT Lib (https://www.cs.toronto.edu/maxsat-lib/) and the single objective deconstructed according to [42], resulting in 3 objectives for shiftdesign and 2 for spot5. The packup, shiftdesign, and lidr families have unit coefficients in all objectives, i.e., are unweighted in MaxSAT terminology. For a balanced and meaningful benchmark set, we randomly sampled instances from each benchmark family, discarding instances that were solved in less than five seconds by P-Minimal without core boosting, until we obtained 20 instances per number of objectives and benchmark family.

4.1 Impact of Core Boosting on Solver Performance

We turn to the results of the evaluation. Since BiOptSat is specific to bi-objective problems, we report on its performance solely on the 120 instances with two objectives. Out of the 20 shiftdesign instances, all configurations of P-Minimal and LowerBound solved exactly 1 instance. We therefore exclude shiftdesign from the reported results.

Table 1 shows the number of solved instances and the cumulative runtime (divided by 10^3 seconds) over the solved instances per benchmark family for P-Minimal, BiOptSat, and LowerBound with and without core boosting. Especially on the set covering and spot5 families, all algorithms benefit greatly from core boosting. The core-boosted configurations often solve more than twice as many instances as the variant without core boosting. With core boosting, all solvers solve more set-cover-ep benchmarks in less cumulative runtime than without core boosting.

Fig. 4 shows a per-instance runtime comparison of P-Minimal, BiOptSat, and LowerBound, respectively, with and without core boosting. It can be seen that many of the spot5 instances that could not be solver without core boosting within the 1.5-hour time limit become trivial to solve after core boosting: 5 spot5 instances that P-Minimal does not solve without core boosting are solved in under 5 seconds after core boosting. On the other benchmark families, core boosting both allows for solving more instances and also drastically reduces solving times.

The time spent in core boosting is for a great majority of the benchmark instances negligible compared to time spent in the MO-MaxSAT solvers: only for

Table 1. Number of solved instances (#) and cumulative runtime over solved instances in 10^3 seconds for each algorithm with and without core boosting (CB).

Algorithm	CB	set-cover-sc #	set-cover-sc $\sum t$	set-cover-ep #	set-cover-ep $\sum t$	packup #	packup $\sum t$	lidr #	lidr $\sum t$	ftp #	ftp $\sum t$	spot5 #	spot5 $\sum t$
P-MINIMAL	no	14	22.74	34	26.82	13	10.10	**7**	7.97	**6**	3.53	2	0.33
	yes	**34**	37.62	**39**	20.53	13	**7.14**	6	4.01	5	1.35	**13**	1.27
BIOPTSAT	no	8	4.84	18	8.09	8	0.65	6	4.61	**7**	8.56	2	0.35
	yes	**16**	8.37	**19**	3.76	8	**0.24**	6	**2.44**	5	1.32	**13**	1.07
LOWERBOUND	no	5	1.97	15	15.01	8	0.44	5	6.61	**6**	3.16	2	0.69
	yes	**13**	3.55	**18**	6.86	8	**0.22**	5	**5.04**	5	1.39	**13**	6.79

10 (resp. 9) instances more than 5% of the runtime was spent in core boosting in conjunction with *P*-MINIMAL (resp., BIOPTSAT or LOWERBOUND). Core boosting timed out on only a single benchmark instance that was solved without core boosting.

As core boosting can be viewed as preprocessing, we also compare its impact on solver runtimes to the impact of the recently-proposed MO-MaxSAT preprocessor MaxPre 2.1 [27] implementing liftings of SAT and MaxSAT preprocessing techniques to MO-MaxSAT. Table 2 shows the effect of core boosting and MaxPre on the number of instances solved by *P*-MINIMAL, BIOPTSAT, and LOWERBOUND in terms of the change in number of solved instances. Overall, the positive impact of core boosting is more significant than that of MaxPre. However, as MaxPre has a somewhat more positive impact on ftp and lidr families, an interesting direction for further work would be to study how to interleave core boosting and the various MaxPre preprocessing techniques for maximal positive overall impact on runtimes.

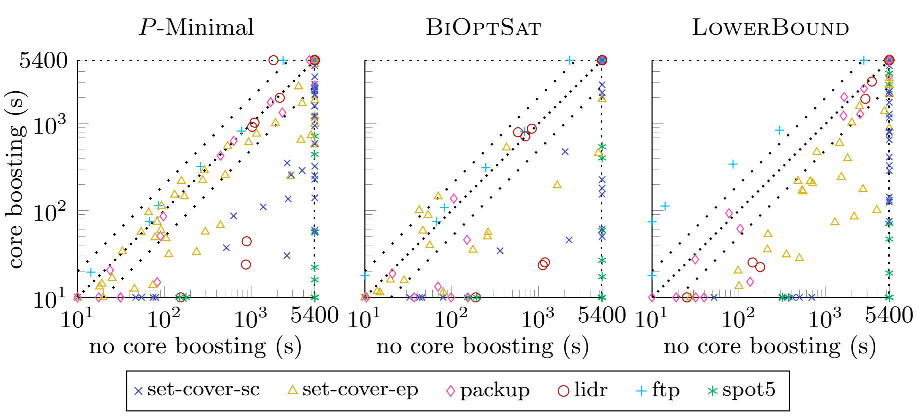

Fig. 4. Comparison of the per-instance CPU time of the *P*-MINIMAL (left), BIOPTSAT (middle), and LOWERBOUND (right) algorithms with and without core boosting.

Table 2. Change in number of solved instances ($\Delta\#$) through core boosting (CB) and preprocessing with MaxPre.

Algorithm	Prepro.	set-cover-sc $\Delta\#$	set-cover-ep $\Delta\#$	packup $\Delta\#$	lidr $\Delta\#$	ftp $\Delta\#$	spot5 $\Delta\#$
P-Minimal	CB	**+20**	**+5**	**±0**	−1	−1	**+11**
	MaxPre	+1	−1	−1	**±0**	**+3**	+1
BiOptSat	CB	**+8**	**+1**	**±0**	**±0**	−2	**+11**
	MaxPre	±0	±0	**±0**	**±0**	**+2**	+1
LowerBound	CB	**+16**	**+6**	**+1**	±0	−1	**+11**
	MaxPre	+1	±0	−1	**+1**	**+1**	±0

4.2 Impact of Core Boosting on Search Space and Instance Size

Finally, we analyze the effects of core boosting on the search space and constraint encodings during search, and how these relate to changes in solving time. Due to space constraints, we focus on presenting results for P-Minimal; the data for BiOptSat and LowerBound shows the same trends.

Fig. 5 (left) relates the impact of core boosting on solver performance with reduction of search space achieved by core boosting. The change in search space (on the x-axis) is measured as the (hyper)volume of the search space after core boosting relative to the original volume, i.e., a value of 50% represents that the search space volume was halved with 100% representing that core boosting has no effect. In detail, the measure is $\frac{V(\mathcal{O}^{\mathrm{cb}})}{V(\mathcal{O})} \cdot 100$, where $V(\mathcal{O}) = \prod_{O \in \mathcal{O}} \sum_{l_i \in \textsc{lits}(O)} c_i$ is the search space volume of a given set of objectives, i.e., the product of the objective coefficient sums. The impact of core boosting on solver performance is measured as $\frac{t_{\mathrm{no\ cb}} - t_{\mathrm{cb}}}{t_{\mathrm{no\ cb}} + t_{\mathrm{cb}}}$, with $t_{(\mathrm{no})\ \mathrm{cb}}$ denoting solving time of P-Minimal with and without core boosting. We additionally assign value 1 (-1) for instances only solved with (without) core boosting. Positive (negative) values therefore express a positive (negative) impact of core boosting in terms of decreased solving time, with 0 representing no impact. We observe that core boosting has the strongest positive impact on solver performance on those instances that it significantly reduces the search space of.

Fig. 5 (right) shows the combined number of clauses in the PB constraint encodings in the MO-MaxSAT solver with and without core boosting. Here clauses were counted at beginning of search based on the same initial solution to eliminate differences due to diverging search trajectories. For most benchmark families—esp. ones with unit coefficients in the objectives—the size of the encodings decreases due to core boosting. However, on the set covering instances core boosting results in larger PB encodings. As core boosting nevertheless decreases overall solving time of also these set covering instances, there appears to be no clear correlation between the changes in encoding sizes and solving times in general.

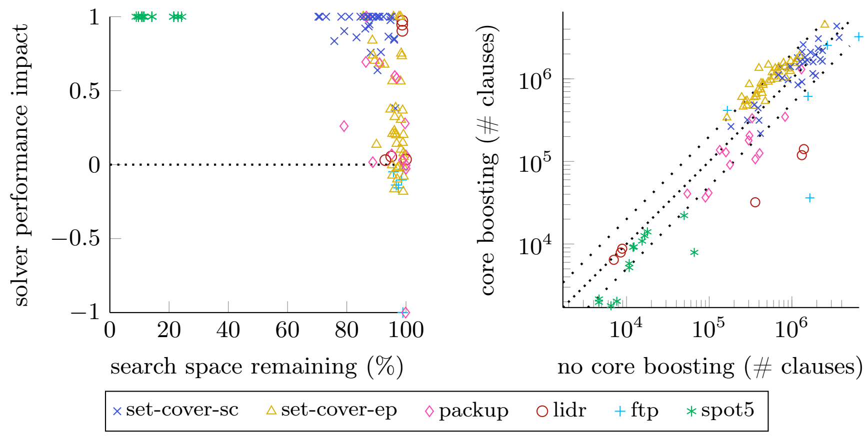

Fig. 5. Left: relating the impact of core boosting on solver performance with reduction of search space achieved (P-MINIMAL). Right: number of clauses in all objective encodings with and without core boosting.

As a further remark, we also experimented with resetting the internal SAT solver between core boosting and invocation of P-MINIMAL to check whether keeping the SAT solver state (learned clauses, variable activities, and polarities) can have an effect on overall runtimes. We observed no meaningful difference between resetting the SAT solver and keeping it alive throughout.

5 Conclusions

We proposed core boosting as an MO-MaxSAT reformulation technique that maintains all Pareto-optimal solutions. Core boosting increases the objective offsets of an MO-MaxSAT instance to match the ideal point without removing any Pareto-optimal solutions through reformulating the MO-MaxSAT instance. This results in a more restricted search space for a subsequently called MO-MaxSAT solver. Through tight integration into SAT-based MO-MaxSAT solvers, our empirical evaluation suggests that core boosting often has a significant positive impact on the runtimes of recently-proposed MO-MaxSAT algorithms, allows for solving more instances, and is more impactful than present MaxSAT-based MO-MaxSAT preprocessing techniques. The adaptation of core boosting to multi-objective generalizations of core-guided optimization algorithms proposed beyond MaxSAT, including CP and pseudo-Boolean optimization, is an interesting direction for further work.

Acknowledgments. Work financially supported by Research Council of Finland (grants 342145, 356046). The authors thank the Finnish Computing Competence Infrastructure for computational and data storage resources.

Disclosure of Interests. The authors have no competing interests to declare.

References

1. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: Dovier, A., Costa, V.S. (eds.) Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4–8, 2012, Budapest, Hungary. LIPIcs, vol. 17, pp. 211–221. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2012). https://doi.org/10.4230/LIPICS.ICLP.2012.211
2. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial MaxSAT through satisfiability testing. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing—SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 – July 3, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5584, pp. 427–440. Springer (2009). https://doi.org/10.1007/978-3-642-02777-2_39
3. Ansótegui, C., Didier, F., Gabàs, J.: Exploiting the structure of unsatisfiable cores in MaxSAT. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015. pp. 283–289. AAAI Press (2015), http://ijcai.org/Abstract/15/046
4. Ansótegui, C., Gabàs, J.: WPM3: an (in)complete algorithm for weighted partial MaxSAT. Artif. Intell. **250**, 37–57 (2017). https://doi.org/10.1016/J.ARTINT.2017.05.003
5. Bacchus, F., Järvisalo, M., Martins, R.: Maximum satisfiability. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability—Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 929–991. IOS Press (2021). https://doi.org/10.3233/FAIA201008
6. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of Boolean cardinality constraints. In: Rossi, F. (ed.) Principles and Practice of Constraint Programming—CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 – October 3, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2833, pp. 108–122. Springer (2003). https://doi.org/10.1007/978-3-540-45193-8_8
7. Belov, A., Morgado, A., Marques-Silva, J.: SAT-based preprocessing for MaxSAT. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning—19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14–19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8312, pp. 96–111. Springer (2013). https://doi.org/10.1007/978-3-642-45221-5_7, https://doi.org/10.1007/978-3-642-45221-5_7
8. Berg, J., Demirovic, E., Stuckey, P.J.: Core-boosted linear search for incomplete MaxSAT. In: Rousseau, L., Stergiou, K. (eds.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research—16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4–7, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11494, pp. 39–56. Springer (2019). https://doi.org/10.1007/978-3-030-19212-9_3
9. Berg, J., Järvisalo, M.: Weight-aware core extraction in SAT-based MaxSAT solving. In: Beck, J.C. (ed.) Principles and Practice of Constraint Programming—23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10416, pp. 652–670. Springer (2017). https://doi.org/10.1007/978-3-319-66158-2_42
10. Berg, J., Saikko, P., Järvisalo, M.: Subsumed label elimination for maximum satisfiability. In: Kaminka, G.A., Fox, M., Bouquet, P., Hüllermeier, E., Dignum,

V., Dignum, F., van Harmelen, F. (eds.) ECAI 2016—22nd European Conference on Artificial Intelligence, 29 August–2 September 2016, The Hague, The Netherlands—Including Prestigious Applications of Artificial Intelligence (PAIS 2016). Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 630–638. IOS Press (2016). https://doi.org/10.3233/978-1-61499-672-9-630

11. Berre, D.L., Parrain, A.: The Sat4j library, release 2.2. J. Satisf. Boolean Model. Comput. **7**(2-3), 59–6 (2010). https://doi.org/10.3233/SAT190075, https://doi.org/10.3233/sat190075
12. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability—Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press (2021). https://doi.org/10.3233/FAIA336
13. Biere, A., Järvisalo, M., Kiesl, B.: Preprocessing in SAT solving. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability—Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 391–435. IOS Press (2021). https://doi.org/10.3233/FAIA200992
14. Cabral, M., Janota, M., Manquinho, V.M.: SAT-based leximax optimisation algorithms. In: Meel, K.S., Strichman, O. (eds.) 25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2–5, 2022, Haifa, Israel. LIPIcs, vol. 236, pp. 29:1–29:19. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPICS.SAT.2022.29
15. Cortes, J., Lynce, I., Manquinho, V.M.: New core-guided and hitting set algorithms for multi-objective combinatorial optimization. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems—29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22–27, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13994, pp. 55–73. Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_7
16. Devriendt, J., Gocht, S., Demirovic, E., Nordström, J., Stuckey, P.J.: Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. pp. 3750–3758. AAAI Press (2021). https://doi.org/10.1609/AAAI.V35I5.16492
17. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electronic Notes in Theoretical Computer Science **89**(4), 543–560 (2003). https://doi.org/10.1016/S1571-0661(05)82542-3
18. Eén, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. J. Satisf. Boolean Model. Comput. **2**(1-4), 1–26 (2006). https://doi.org/10.3233/SAT190014
19. Ehrgott, M.: Multicriteria Optimization (2. ed.). Springer (2005). https://doi.org/10.1007/3-540-27659-9
20. Ehrgott, M., Gandibleux, X.: Bound sets for biobjective combinatorial optimization problems. Comput. Oper. Res. **34**(9), 2674–2694 (2007). https://doi.org/10.1016/J.COR.2005.10.003
21. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) Theory and Applications of Satisfiability Testing—SAT 2006, 9th International Conference, Seattle, WA, USA, August 12–15, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4121, pp. 252–265. Springer (2006). https://doi.org/10.1007/11814948_25

22. Gange, G., Berg, J., Demirovic, E., Stuckey, P.J.: Core-guided and core-boosted search for CP. In: Hebrard, E., Musliu, N. (eds.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research—17th International Conference, CPAIOR 2020, Vienna, Austria, September 21-24, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12296, pp. 205–221. Springer (2020). https://doi.org/10.1007/978-3-030-58942-4_14
23. Guerreiro, A.P., Cortes, J., Vanderpooten, D., Bazgan, C., Lynce, I., Manquinho, V.M., Figueira, J.R.: Exact and approximate determination of the Pareto front using minimal correction subsets. Comput. Oper. Res. **153**, 106153 (2023). https://doi.org/10.1016/J.COR.2023.106153
24. Heras, F., Larrosa, J., de Givry, S., Schiex, T.: 2006 and 2007 Max-SAT Evaluations: Contributed instances. J. Satisf. Boolean Model. Comput. **4**(2-4), 239–250 (2008). https://doi.org/10.3233/SAT190046
25. Ignatiev, A., Morgado, A., Marques-Silva, J.: RC2: an efficient MaxSAT solver. J. Satisf. Boolean Model. Comput. **11**(1), 53–64 (2019). https://doi.org/10.3233/SAT190116
26. Ihalainen, H., Berg, J., Järvisalo, M.: Refined core relaxation for core-guided MaxSAT solving. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25–29, 2021. LIPIcs, vol. 210, pp. 28:1–28:19. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPICS.CP.2021.28
27. Jabs, C., Berg, J., Ihalainen, H., Järvisalo, M.: Preprocessing in SAT-based multi-objective combinatorial optimization. In: Yap, R.H.C. (ed.) 29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27–31, 2023, Toronto, Canada. LIPIcs, vol. 280, pp. 18:1–18:20. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.CP.2023.18
28. Jabs, C., Berg, J., Niskanen, A., Järvisalo, M.: MaxSAT-based bi-objective Boolean optimization. In: Meel, K.S., Strichman, O. (eds.) 25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2–5, 2022, Haifa, Israel. LIPIcs, vol. 236, pp. 12:1–12:23. Schloss Dagstuhl—Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPICS.SAT.2022.12
29. Janota, M., Lynce, I., Manquinho, V.M., Marques-Silva, J.: PackUp: Tools for package upgradability solving. J. Satisf. Boolean Model. Comput. **8**(1/2), 89–94 (2012). https://doi.org/10.3233/SAT190090
30. Joshi, S., Martins, R., Manquinho, V.M.: Generalized totalizer encoding for pseudo-Boolean constraints. In: Pesant, G. (ed.) Principles and Practice of Constraint Programming—21st International Conference, CP 2015, Cork, Ireland, August 31 – September 4, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9255, pp. 200–209. Springer (2015). https://doi.org/10.1007/978-3-319-23219-5_15
31. Koshimura, M., Nabeshima, H., Fujita, H., Hasegawa, R.: Minimal model generation with respect to an atom set. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Proceedings of the 7th International Workshop on First-Order Theorem Proving, FTP 2009, Oslo, Norway, July 6–7, 2009. CEUR Workshop Proceedings, vol. 556. CEUR-WS.org (2009), https://ceur-ws.org/Vol-556/paper06.pdf
32. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: A partial MaxSAT solver. J. Satisf. Boolean Model. Comput. **8**(1/2), 95–100 (2012). https://doi.org/10.3233/SAT190091, https://doi.org/10.3233/sat190091

33. Malioutov, D., Meel, K.S.: MLIC: A MaxSAT-based framework for learning interpretable classification rules. In: Hooker, J.N. (ed.) Principles and Practice of Constraint Programming—24th International Conference, CP 2018, Lille, France, August 27–31, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11008, pp. 312–327. Springer (2018). https://doi.org/10.1007/978-3-319-98334-9_21
34. Marler, R., Arora, J.: Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization **26**, 369–395 (04 2004). https://doi.org/10.1007/s00158-003-0368-6
35. Marques, R., Russo, L.M.S., Roma, N.: Flying tourist problem: Flight time and cost minimization in complex routes. Expert Syst. Appl. **130**, 172–187 (2019). https://doi.org/10.1016/J.ESWA.2019.04.024
36. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability—Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 133–182. IOS Press (2021). https://doi.org/10.3233/FAIA200987
37. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. Computing Research Repository **abs/0712.1097** (2007), http://arxiv.org/abs/0712.1097
38. Martins, R.: ASP to MaxSAT: Metro, ShiftDesign, TimeTabling and BioRepair. In: Ansotegui, C., Bacchus, F., Järvislo, M., Martins, R. (eds.) MaxSAT Evaluation 2017: Solver and Benchmark Descriptions. p. 27. University of Helsinki (2017), http://hdl.handle.net/10138/228949
39. Martins, R., Joshi, S., Manquinho, V.M., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: O'Sullivan, B. (ed.) Principles and Practice of Constraint Programming—20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8656, pp. 531–548. Springer (2014). https://doi.org/10.1007/978-3-319-10428-7_39
40. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: O'Sullivan, B. (ed.) Principles and Practice of Constraint Programming—20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8656, pp. 564–573. Springer (2014). https://doi.org/10.1007/978-3-319-10428-7_41
41. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27–31, 2014, Québec City, Québec, Canada. pp. 2717–2723. AAAI Press (2014). https://doi.org/10.1609/AAAI.V28I1.9124
42. Paxian, T., Raiola, P., Becker, B.: On preprocessing for weighted MaxSAT. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) Verification, Model Checking, and Abstract Interpretation—22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17–19, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12597, pp. 556–577. Springer (2021). https://doi.org/10.1007/978-3-030-67067-2_25
43. Piotrów, M.: UWrMaxSat: Efficient solver for MaxSAT and pseudo-Boolean problems. In: 32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9–11, 2020. pp. 132–136. IEEE (2020). https://doi.org/10.1109/ICTAI50040.2020.00031
44. Soh, T., Banbara, M., Tamura, N., Berre, D.L.: Solving multiobjective discrete optimization problems with propositional minimal model generation. In: Beck, J.C. (ed.) Principles and Practice of Constraint Programming—23rd International

Conference, CP 2017, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10416, pp. 596–614. Springer (2017). https://doi.org/10.1007/978-3-319-66158-2_38

45. Terra-Neves, M., Lynce, I., Manquinho, V.M.: Multi-objective optimization through Pareto minimal correction subsets. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden. pp. 5379–5383. ijcai.org (2018). https://doi.org/10.24963/IJCAI.2018/757
46. Tobias Paxian, Sven Reimer, B.B.: Pacose: An iterative SAT-based MaxSAT solver. In: Bacchus, F., Berg, J., Järvisalo, M., Martins, R. (eds.) MaxSAT Evaluation 2020: Solver and Benchmark Descriptions, Department of Computer Science Series of Publications B, vol. B-2020-2. University of Helsinki (2020), http://hdl.handle.net/10138/318451