Core-Boosted Linear Search for Incomplete MaxSAT*

Jeremias Berg[1], Emir Demirović[2], and Peter J. Stuckey[3,4]

[1] HIIT, Department of Computer Science, University of Helsinki, Finland, jeremias.berg@cs.helsinki.fi
[2] University of Melbourne, Australia, emir.demirovic@unimelb.edu.au
[3] Monash University, Australia, peter.stuckey@monash.edu
[4] Data61, CSIRO, Australia

Abstract. Maximum Satisfiability (MaxSAT), the optimisation extension of the well-known Boolean Satisfiability (SAT) problem, is a competitive approach for solving NP-hard problems encountered in various artificial intelligence and industrial domains. Due to its computational complexity, there is an inherent tradeoff between scalability and guarantee on solution quality in MaxSAT solving. Limitations on available computational resources in many practical applications motivate the development of *complete any-time* MaxSAT solvers, i.e. algorithms that compute optimal solutions while providing intermediate results. In this work, we propose *core-boosted linear search*, a generic search-strategy that combines two central approaches in modern MaxSAT solving, namely linear and core-guided algorithms. Our experimental evaluation on a prototype combining reimplementations of two state-of-the-art MaxSAT solvers, PMRES as the core-guided approach and LinSBPS as the linear algorithm, demonstrates that our core-boosted linear algorithm often outperforms its individual components and shows competitive and, on many domains, superior results when compared to other state-of-the-art solvers for incomplete MaxSAT solving.

Keywords: Maximum Satisfiability · MaxSAT · SAT-based MaxSAT · incomplete solving · linear algorithm · core-guided MaxSAT

1 Introduction

Discrete optimisation problems are ubiquitous throughout society. When solving a discrete optimisation problem, the goal is to find the best solution according to a given objective function among a finite, but potentially large set of possibilities. Examples of such problems include scheduling, routing, timetabling, and other forms of management decision problems. The solution approaches to discrete

* The first author is financially supported by the University of Helsinki Doctoral Program in Computer Science and the Academy of Finland (grant 312662). We thank the University of Melbourne and the Melbourne School of Engineering Visiting Fellows scheme for supporting the visit of Jeremias Berg.

optimisation can be divided into *complete* and *incomplete* methods. The aim of complete methods is to find the best possible solution and prove its optimality. However, in many real world applications complete solving is a difficult, and in many cases, a practically infeasible task. Hence, in practice, one might resort to incomplete solving, i.e. computing the best possible solution within a *limited time*, rather than exclusively searching for an optimal solution.

There is a wide range of technologies available for discrete optimisation. The focus of this work is on the Boolean optimisation paradigm of Maximum (Boolean) Satisfiability (MaxSAT), the optimization extension of the well-known Boolean satisfiability (SAT) problem. MaxSAT can be used to solve any NP-hard discrete optimisation problem that can be formulated as minimising a linear objective over Boolean variables subject to a set of clausal constraints. Modern MaxSAT solving technology builds on the exceptional performance improvements of SAT solvers, starting in the late 90s [39, 49]. Most MaxSAT solvers used in real-world applications are SAT-based, i.e. reduce the discrete optimisation problem into a sequence of satisfiability queries of Boolean formulas conjunctive normal form (CNF), and tackle the queries with SAT solvers. In the last decade, MaxSAT solving technology has matured significantly, leading to successful applications of MaxSAT in a wide range of AI and industrial domains, such as timetabling, planning, debugging, diagnosis, machine learning, and systems biology [22, 3, 10, 20, 51, 24, 19, 15, 36]. See [6, 5, 42] for more details.

SAT-based MaxSAT solvers can be roughly partitioned into *linear* [28, 29], *core-guided* [4, 25, 41, 8, 44, 41], and *implicit hitting-set-based* [21, 45] algorithms. The two most relevant ones for this work are the linear and core-guided algorithms. Linear algorithms are upper bounding approaches that encode the MaxSAT instance, along with its pseudo-Boolean objective function, into *conjunctive normal form* (CNF) and iteratively query a SAT solver for a solution better than the current best one. In contrast, core-guided algorithms are lower-bounding approaches that use a SAT solver to extract a series of *unsatisfiable cores*, i.e. sets of soft constraints that cannot be simultaneously satisfied, and reformulate the underlying MaxSAT instance to rule out each core as a source of unsatisfiability. Both search strategies have shown strong performance in the annual MaxSAT evaluations, linear search is particularly effective for incomplete solving while many of the best performing complete solvers are core-guided.

As our main contribution, we propose *core-boosted linear search* for incomplete MaxSAT solving, a novel search strategy that combines linear and core-guided search with the aim of achieving the best of both worlds. A core-boosted solver initially reformulates an input instance with a core-guided solver and then solves the reformulated instance with a linear search solver. The exchange of information from the core-guided phase to the linear phase tightens the gap between the lower and upper bound, allowing the use of a simpler pseudo-Boolean encoding. As a result, the approach is often more effective than either a pure linear or a pure core-guided search.

To demonstrate the potential of core-boosted linear search we report on an experimental evaluation of a prototype solver that combines reimplementations

of two state-of-the-art MaxSAT solvers, PMRES [44] as the core-guided algorithm and LinSBPS [14] as the linear algorithm. We compare core-boosted linear search to its individual components on a standard set of benchmarks. Our results indicate that core-boosted linear search is indeed more effective that either core-guided or linear search for incomplete solving. An in-depth look at the search progression on three selected instances demonstrates the ability of core-boosted linear search to both avoid the worst-case executions of its components, and make use the information flow between them to more quickly find solutions of higher quality.

The rest of the paper is organised as follows. After the preliminaries in Section 2, we give a detailed discussion of core-guided and linear search methods for MaxSAT in Section 3. Core-boosted linear search is then presented in Section 4. We discuss related work in Section 5, after which we present our experimental evaluation in Section 6. Lastly, we give concluding remarks in Section 7.

2 Preliminaries

For a Boolean variable x there are two literals, the positive x and the negative $\neg x$. The negation $\neg l$ of a literal l satisfies $\neg\neg l = l$. A clause C is a disjunction ($\vee$) of literals (represented as a set of its literals), and a CNF formula F a conjunction ($\wedge$) of clauses (represented as a set of its clauses). The set $\textsc{Var}(F)$ of the variables of F contains all variables x s.t. $x \in C$ or $\neg x \in C$ for some $C \in F$. We assume familiarity with other logical connectives and denote by $\text{CNF}(\phi)$ a set of clauses logically equivalent to the formula ϕ. We also assume without loss of generality, that the size of $\text{CNF}(\phi)$ is linear in the size of ϕ [47].

A truth assignment τ is a function mapping Boolean variables to 1 (true) or 0 (false). A clause C is satisfied by τ (denoted by $\tau(C) = 1$) if $\tau(l) = 1$ for a positive or $\tau(l) = 0$ for a negative literal $l \in C$, otherwise C is falsified by τ (denoted $\tau(C) = 0$). A CNF formula F is satisfied by τ ($\tau(F) = 1$) if τ satisfies all clauses in the formula and falsified otherwise ($\tau(F) = 0$). If some τ satisfies a CNF formula F, then F is satisfiable, otherwise it is unsatisfiable. The NP-complete Boolean Satisfiability problem (SAT) asks to decide if a given CNF formula F is satisfiable [17].

A (weighted partial) MaxSAT instance $\mathcal{F}$ consists of two sets of clauses: the hard $\textsc{Hard}(\mathcal{F})$, the soft $\textsc{Soft}(\mathcal{F})$, and a function $w^{\mathcal{F}} \colon \textsc{Soft}(\mathcal{F}) \to \mathbb{N}$ associating a positive integral cost to each soft clause. The set $\textsc{Var}(\mathcal{F})$ of the variables of $\mathcal{F}$ is $\textsc{Var}(\textsc{Hard}(\mathcal{F})) \cup \textsc{Var}(\textsc{Soft}(\mathcal{F}))$. An assignment τ is a solution to $\mathcal{F}$ if $\tau(\textsc{Hard}(\mathcal{F})) = 1$. The cost $\text{COST}(\mathcal{F}, \tau)$ of a solution τ to $\mathcal{F}$ is the sum of the weights of the soft clauses it falsifies i.e. $\text{COST}(\mathcal{F}, \tau) = \sum_{C \in \textsc{Soft}(\mathcal{F})} w^{\mathcal{F}}(C) \times (1 - \tau(C))$. A solution τ is optimal if $\text{COST}(\mathcal{F}, \tau) \leq \text{COST}(\mathcal{F}, \tau')$ for all solutions τ' to $\mathcal{F}$. We denote the cost of the optimal solutions to $\mathcal{F}$ by $\text{COST}(\mathcal{F})$. The NP-hard (weighted partial) MaxSAT problem asks to compute an optimal solution to a given instance $\mathcal{F}$. In the rest of the paper we will assume that all MaxSAT instances have solutions, i.e. that $\textsc{Hard}(\mathcal{F})$ is satisfiable.

Algorithm 1: LIN-SEARCH

Input: A MaxSAT instance $\mathcal{F}$
Output: An optimal solution τ to $\mathcal{F}$
begin
  $n \leftarrow |\textsc{Soft}(\mathcal{F})|, \quad \tau^\star \leftarrow \textsc{InitialSolution}(\mathcal{F})$
  $\mathcal{R} \leftarrow \{r_1, \ldots, r_n\}, \quad F_s^R = \{C_i \vee r_i \mid C_i \in \textsc{Soft}(\mathcal{F}), r_i \notin \textsc{Var}(\mathcal{F})\}$
  while *true* **do**
    if *Resource-Out* **then** return $\tau^\star$
    $PB \leftarrow \sum_{i=1}^{n} w^{\mathcal{F}}(C_i) \times r_i < \text{COST}(\mathcal{F}, \tau^\star)$
    $F_w \leftarrow \textsc{Hard}(\mathcal{F}) \cup F_s^R \cup \text{CNF}(PB)$
    $(res, \tau) \leftarrow \text{SATSOLVE}(F_w)$
    if *res="satisfiable"* **then** $\tau^\star \leftarrow \tau$
    else return $\tau^\star$

A central concept in many SAT-based MaxSAT algorithms is that of an *(unsatisfiable) core*. For a MaxSAT instance $\mathcal{F}$, a subset $\kappa \subseteq \textsc{Soft}(\mathcal{F})$ of soft clauses is an unsatisfiable core of $\mathcal{F}$ iff $\textsc{Hard}(\mathcal{F}) \wedge \kappa$ is unsatisfiable.

3 Core-Guided and Linear Search for Incomplete MaxSAT

We detail two abstract MaxSAT solving algorithms, LIN-SEARCH (Algorithm 1) and CORE-GUIDED (Algorithm 2), representing linear and core-guided search, respectively. Both use SAT-solvers to reduce MaxSAT solving into a sequence of satisfiability queries. However, the manner in which the SAT solver is used differs significantly. We present both algorithms as complete *any-time* algorithms, i.e. algorithms that, given enough resources, compute the optimal solution to a MaxSAT instance while also providing intermediate solutions during search.

In the following descriptions of the MaxSAT algorithms, we abstract the use of the SAT-solver into two functions. The function SATSOLVE represents a basic SAT-solver query. Given a CNF formula F, the query $\text{SATSOLVE}(F)$ returns a tuple (res, τ), where *res* denotes whether the formula is satisfiable and τ is a satisfying assignment to F if one exists. The extended function $\textsc{Extract-Core}(\textsc{Hard}(\mathcal{F}), \textsc{Soft}(\mathcal{F}))$ takes as input the hard and soft clauses of a MaxSAT instance $\mathcal{F}$ and returns a triplet (res, κ, τ), where *res* indicates if $\textsc{Hard}(\mathcal{F}) \wedge \textsc{Soft}(\mathcal{F})$ is satisfiable, τ is a satisfying assignment for $\textsc{Hard}(\mathcal{F}) \wedge \textsc{Soft}(\mathcal{F})$ if one exists, and $\kappa \subset \textsc{Soft}(\mathcal{F})$ is a core of $\mathcal{F}$ if $\textsc{Hard}(\mathcal{F}) \wedge \textsc{Soft}(\mathcal{F})$ is unsatisfiable. Practically all SAT-solvers used in MaxSAT solving offer a so-called *assumption interface* [43] that can be used to implement SATSOLVE and EXTRACT-CORE.

The pseudocode of LIN-SEARCH, is detailed in Algorithm 1. When solving an instance $\mathcal{F}$, LIN-SEARCH refines an upper bound on $\text{COST}(\mathcal{F})$ by maintaining and iteratively improving a best known solution $\tau^\star$ to $\mathcal{F}$. Initially, $\tau^\star$ is set to

any solution of $\mathcal{F}$, for example by invoking the SAT solver on $\textsc{Hard}(\mathcal{F})$. During search, the existence of a solution τ having cost less that $\tau^\star$ is checked by querying the internal SAT solver. If no such solution is found, then $\tau^\star$ is optimal and Lin-Search terminates. Otherwise $\tau^\star$ is updated and the search continues. In more detail, the existence of a solution τ for which $\text{COST}(\mathcal{F},\tau) < \text{COST}(\mathcal{F},\tau^\star)$ is checked by querying the SAT-solver for the satisfiability of a working formula $F_w = \textsc{Hard}(\mathcal{F}) \cup F_s^R \cup \text{CNF}(PB)$ consisting of the hard clauses, the soft clauses each extended with a unique *relaxation variable* r_i and a CNF-encoding of a *pseudo-Boolean* (PB) constraint $PB = \sum_{i=1}^{n} w^{\mathcal{F}}(C_i) \times r_i < \text{COST}(\mathcal{F},\tau^\star)$ that is satisfied by an assignment τ iff $\sum_{i=1}^{n} w^{\mathcal{F}}(C_i) \times \tau(r_i) < \text{COST}(\mathcal{F},\tau^\star)$. The intuition underlying F_w is that setting a relaxation variable r_i to true allows falsification of the corresponding soft clause C_i. Thus the PB constraint essentially limits the sum of the weights of the soft clauses falsified by an assignment τ to be less than the current best known upper bound $\text{COST}(\mathcal{F},\tau^\star)$ on $\text{COST}(\mathcal{F})$. In other words, F_w is satisfied by an assignment τ iff τ is a solution to $\mathcal{F}$ for which $\text{COST}(\mathcal{F},\tau) < \text{COST}(\mathcal{F},\tau^\star)$.

Before proceeding with core-guided search, we make two observations regarding the effectiveness of Lin-Search that are important for understanding core-boosted linear search. As the search in Lin-Search is focused on decreasing the best known upper bound, we expect it to be most effective for solving an instance $\mathcal{F}$ when the difference between $\text{COST}(\mathcal{F})$ and the cost $\text{COST}(\mathcal{F},\tau^\star)$ of the initial solution $\tau^\star$ is small. Thus, a high quality, i.e. low cost, initial solution can have a significant impact on the overall performance of Lin-Search. The second observation concerns the PB constraint $\sum_{i=1}^{n} w^{\mathcal{F}}(C_i) \times r_i < \text{COST}(\mathcal{F},\tau^\star)$. Similar constraints are encountered in many different domains, as such a lot of research has been put into developing efficient CNF encodings of them [46, 11, 27]. Even so, the PB constraint is arguably the main bottleneck of the overall performance of Lin-Search and we expect any further techniques that allow the use of simpler, and more compact (encodings) PB constraints to improve the overall performance of Lin-Search.

The pseudocode of Core-Guided, basic core-guided search extended with *stratification* [37, 7], is detailed in Algorithm 2. Stratification is a heuristic designed to steer the core extraction of Core-Guided toward cores κ for which the minimum weight of the clauses in κ is high. Stratification is a standard technique in modern core-guided solvers. Importantly for this work, stratification allows us to treat core-guided search as an any-time method for MaxSAT.

When solving an instance $\mathcal{F}$, Core-Guided maintains a working instance initialised to $\mathcal{F}$ and a stratification bound b^{STRAT} initialised to the highest weight of the soft clauses in $\mathcal{F}$. During iteration i of the main search loop, the SAT solver is queried for a core κ^i of a subset of the current working instance $\mathcal{F}^i$ containing all hard clauses and STRAT, all soft clauses with weight greater than or equal to b^{STRAT}. If no such core exists, an intermediate solution τ is obtained and compared to the best known solution $\tau^\star$. If all soft clauses were considered in the SAT call, the obtained solution is also optimal and the algorithm terminates. If not, the bound b^{STRAT} is lowered and the search con-

Algorithm 2: CORE-GUIDED

Input: A MaxSAT instance $\mathcal{F}$
Output: An optimal solution τ to $\mathcal{F}$
begin
  $\tau^\star \leftarrow \textsc{InitialSolution}(\mathcal{F}), \quad b^{\text{STRAT}} \leftarrow \max\{w^{\mathcal{F}}(C) \mid C \in \textsc{Soft}(\mathcal{F})\}$
  $\mathcal{F}^1 \leftarrow \mathcal{F}, \quad i \leftarrow 1$
  while *true* **do**
    if *Resource-Out* **then** return $\tau^\star$
    $\text{STRAT} \leftarrow \{C \mid C \in \textsc{Soft}(\mathcal{F}^i), w^{\mathcal{F}^i}(C) \geq b^{\text{STRAT}}\}$
    $(res, \kappa^i, \tau) \leftarrow \textsc{Extract-Core}(\textsc{Hard}(\mathcal{F}^i), \text{STRAT})$
    if *res="satisfiable"* **then**
      if $\text{COST}(\mathcal{F}, \tau) < \text{COST}(\mathcal{F}, \tau^\star)$ **then** $\tau^\star \leftarrow \tau$
      if $\text{STRAT} = \textsc{Soft}(\mathcal{F}^i)$ **then return** τ
      else $b^{\text{STRAT}} \leftarrow \max\{w^{\mathcal{F}^i}(C) \mid C \in \textsc{Soft}(\mathcal{F}^i), w^{\mathcal{F}^i}(C) < b^{\text{STRAT}}\}$
    else
      $\mathcal{F}^{i+1} \leftarrow \textsc{Reformulate}(\mathcal{F}^i, \kappa^i)$
      $i \leftarrow i + 1$

tinues. When a core κ^i is extracted, the working instance is updated by the function REFORMULATE. Informally speaking, REFORMULATE reformulates $\mathcal{F}^i$ in a way that rules out κ^i as a source of unsatisfiability and allows falsifying one clause in κ^i without incurring cost. Most of the core-guided MaxSAT solvers that fit the CORE-GUIDED abstraction [4, 25, 41, 8, 44, 41] differ mainly in the implementation of REFORMULATE. The correctness of such solvers is often established by showing that $\mathcal{F}^i$ is MaxSAT-reducible to $\mathcal{F}^{i+1}$ and that $\textsc{Var}(\mathcal{F}^i) \subset \textsc{Var}(\mathcal{F}^{i+1})$ [6]. While a precise treatment of MaxSAT-reducibility is outside the scope of this work, the next proposition summarises the consequences of it that are important for understanding core-boosted linear search.

Proposition 1. *Let $\mathcal{F}$ be a MaxSAT instance, κ a core of $\mathcal{F}$, $w^\kappa = \min\{w^{\mathcal{F}}(C) \mid C \in \kappa\}$ and $\mathcal{F}^R = \textsc{Reformulate}(\mathcal{F}, \kappa)$. Assume that $\mathcal{F}$ is MaxSAT reducible to $\mathcal{F}^R$ and that $\textsc{Var}(\mathcal{F}) \subset \textsc{Var}(\mathcal{F}^R)$. Then the following hold: (i) any solution τ to $\mathcal{F}$ can be extended into a solution τ^R to $\mathcal{F}^R$ s.t. $\text{COST}(\mathcal{F}, \tau) = \text{COST}(\mathcal{F}^R, \tau^R) + w^\kappa$ and (ii) any solution τ^R to $\mathcal{F}^R$ is a solution to $\mathcal{F}$ for which $\text{COST}(\mathcal{F}^R, \tau^R) = \text{COST}(\mathcal{F}, \tau^R) - w^\kappa$.*

An alternative intuition to core-guided search offered by Proposition 1 is thus a search strategy that lowers the optimal cost of its working instance by extracting cores that witness lower bounds and reformulating the instance s.t the cost of every solution to the instance is lowered exactly by the identified lower bound. Core-guided search terminates once the optimum cost of the working instance has been lowered to 0.

Example 1. Let $\mathcal{F}$ be a MaxSAT instance having $\textsc{Hard}(\mathcal{F}) = \{(x_1 \vee x_2), (x_3 \vee x_4)\}$ and $\textsc{Soft}(\mathcal{F}) = \{(\neg x_i) \mid i = 1 \ldots 4\}$ with $w^{\mathcal{F}}((\neg x_1)) = w^{\mathcal{F}}((\neg x_2)) = 1$ and

$w^{\mathcal{F}}((\neg x_3)) = w^{\mathcal{F}}((\neg x_4)) = 2$. We sketch one possible execution of the PMRES algorithm [44], an instantiation of CORE-GUIDED, when invoked on $\mathcal{F}$. First, the initial working formula $\mathcal{F}^1$ is set to $\mathcal{F}$ and the stratification bound b^{STRAT} is set to the highest weight of soft clauses, i.e. 2. Thus $\text{STRAT} = \{(\neg x_3), (\neg x_4)\}$ in the first iteration. The formula $\text{HARD}(\mathcal{F}^1) \wedge \text{STRAT}$ is unsatisfiable, the only core obtainable at this point is $\kappa^1 = \{(\neg x_3), (\neg x_4)\}$. Using the PMRES algorithm, the next working instance $\mathcal{F}^2 = \text{REFORMULATE}(\mathcal{F}^1, \kappa^1)$ has $\text{HARD}(\mathcal{F}^2) = \text{HARD}(\mathcal{F}^1) \cup \{(\neg x_3 \vee \neg r_1), \text{CNF}(d_1 \leftrightarrow \neg x_4)\}$, $\text{SOFT}(\mathcal{F}^2) = \{(\neg x_1), (\neg x_2), (\neg r_1 \vee \neg d_1)\}$ with $w^{\mathcal{F}^2}(x_1) = w^{\mathcal{F}^2}(x_2) = 1$ and $w^{\mathcal{F}^2}((\neg r_1 \vee \neg d_1)) = 2$. The stratification bound is not altered so $\text{STRAT} = \{(\neg r_1 \vee \neg d_1)\}$ during the next iteration. Now $\text{HARD}(\mathcal{F}^2) \wedge \text{STRAT}$ is satisfiable so b^{STRAT} is lowered to 1. In the next iteration $\text{STRAT} = \text{SOFT}(\mathcal{F}^2)$ and the SAT solver obtains the core $\kappa^2 = \{(\neg x_1), (\neg x_2)\}$. The instance is again reformulated and the next working instance $\mathcal{F}^3 = \text{REFORMULATE}(\mathcal{F}^2, \kappa^2)$ has $\text{HARD}(\mathcal{F}^3) = \text{HARD}(\mathcal{F}^2) \cup \{(\neg x_1 \vee \neg r_2), \text{CNF}(d_2 \leftrightarrow \neg x_2))\}$ and $\text{SOFT}(\mathcal{F}^3) = \{(\neg r_2 \vee \neg d_2), (\neg r_1 \vee \neg d_1)\}$ with $w^{\mathcal{F}^3}((\neg r_2 \vee \neg d_2)) = 1$ and $w^{\mathcal{F}^3}((\neg r_1 \vee \neg d_1)) = 2$. In the final iteration $\text{STRAT} = \text{SOFT}(\mathcal{F}^3)$ and since $\text{HARD}(\mathcal{F}^3) \wedge \text{SOFT}(\mathcal{F}^3)$ is satisfiable, CORE-GUIDED terminates.

We conclude this section with a few observations regarding CORE-GUIDED that are important for understanding core-boosted linear search. When solving an instance $\mathcal{F}$, CORE-GUIDED focuses its search on the lower bound of $\text{COST}(\mathcal{F})$. Thus, we expect CORE-GUIDED to be effective if $\text{COST}(\mathcal{F})$ is low and, in particular, to not be significantly affected by the quality of the initial solution. The main bottleneck of CORE-GUIDED is instead the increased complexity of the core-extraction steps. Note that the core κ^i extracted during the i:th iteration of CORE-GUIDED is a core of the i:th working instance $\mathcal{F}^i$ and not necessarily of the original instance $\mathcal{F}$. While the effects of reformulation on the complexity of the EXTRACT-CORE calls are not fully understood, it has been shown that extracting a core of $\mathcal{F}^i$ can be exponentially harder than extracting a core of $\mathcal{F}$ [13].

4 Core-Boosted Linear Search for incomplete MaxSAT

In this section, we propose and discuss *core-boosted linear search*, the main contribution of our work. The execution of a core-boosted (linear search) algorithm is split into two phases. On input $\mathcal{F}$, the algorithm begins search in a core-guided phase by invoking CORE-GUIDED on $\mathcal{F}$. If CORE-GUIDED is able to find an optimal solution within the resources allocated to it, then the core-boosted algorithm terminates. Otherwise CORE-GUIDED returns its final working instance $\mathcal{F}_w$ along with the best solution $\tau^\star$ it found. The core-boosted algorithm then moves on to its linear phase by invoking LIN-SEARCH on $\mathcal{F}_w$ using $\tau^\star$ as the initial solution. The linear phase runs until either finding the optimal solution to $\mathcal{F}_w$, or running out of computational resources. By Proposition 1, the best solution $\tau^\star$ to $\mathcal{F}_w$ found by LIN-SEARCH is also a solution to $\mathcal{F}$. Specifically, an

optimal solution of $\mathcal{F}_w$ is also an optimal solution to $\mathcal{F}$ implying the completeness of core-boosted linear search for MaxSAT. We emphasize that the linear component Lin-Search of a core-boosted algorithm is invoked on $\mathcal{F}_w$, the final working instance of Core-Guided, and not on $\mathcal{F}$, the initial input instance. As we discuss next and demonstrate in our experiments, this allows the linear phase of core-boosted linear search to benefit from the core-guided phase in a non-trivial manner.

The discussion on Lin-Search and Core-Guided in Section 3 serves as useful basis for understanding the potential benefits of core-boosted linear search. Since core-boosted linear search makes use of both core-guided and linear search, we expect it to be effective both on the same instances as linear search, and as core-guided search, or at least not significantly worse. For example, if the instance $\mathcal{F}$ being solved has low optimal cost, then we expect a core-boosted algorithm to be able to solve the instance effectively during its initial core-guided phase. Similarly, if $\text{COST}(\mathcal{F})$ is close to the cost $\text{COST}(\mathcal{F}, \tau^\star)$ of the initial solution $\tau^\star$, then $\text{COST}(\mathcal{F}_w)$ is also close to $\text{COST}(\mathcal{F}_w, \tau^\star)$. Hence we expect a core-boosted algorithm to be effective during its linear phase, even factoring in the reformulations done during the core-guided phase.

The potential benefits of core-boosted linear search go beyond merely averaging out the performance of core-guided and linear search. As discussed in the previous section, one of the main drawbacks of core-guided search is the increased complexity of core extraction over time. Thus stopping the core-guided phase and solving the working instance by linear search should be beneficial. Further, the linear phase can also benefit from the reformulation steps performed by the core-guided phase. Specifically, such reformulations can decrease the size of the PB constraint $PB = \sum_{i=1}^{n} w^{\mathcal{F}}(C_i) \times r_i < \text{COST}(\mathcal{F}, \tau^\star)$ that needs to be encoded during linear search. Depending on the specific encoding used, the number of clauses resulting from encoding PB into CNF depends either on the magnitudes of the weights of the soft clauses and the right-hand side [23] or on the number of unique sums that can be created from those weights [27]. The reformulation steps performed during the core-guided phase of a core-boosted algorithm can affect both of these factors. By Proposition 1 $\text{COST}(\mathcal{F}_w, \tau^\star) \leq \text{COST}(\mathcal{F}, \tau^\star)$ which implies that both the magnitude of the weights in $\mathcal{F}_w$ and the initial right hand side $\text{COST}(\mathcal{F}_w, \tau^\star)$ of PB are smaller in the reformulated $\mathcal{F}_w$ than in the original $\mathcal{F}$. Additionally, the core-guided phase can also decrease the number of soft clauses in the instance; the second working instance of Example 1 has one less soft clause than the first one. Finally, the so-called *hardening rule* [7] commonly used in conjunction with core-guided search, can also decrease the number of soft clauses of the instance, and thus allow the linear phase of a core-boosted algorithm to use a more compact PB constraint

5 Related Work

We begin by detailing the instantiations of Lin-Search and Core-Guided that we use in the prototype core-boosted linear search algorithm experimented with

in the next section. As the linear search component we use the basic Lin-Search extended with varying resolution and solution-based phase-saving in the style of LinSBPS, the best performing solver of the incomplete 300s track of the 2018 MaxSAT evaluation [14]. Solution-based phase-saving is a heuristic designed to steer the search towards the currently best known solution by modifying the branching heuristic of the internal SAT solver to always prefer setting the polarity of a literal it branches on to equal its polarity in the currently best known solution. Varying resolution is a heuristic designed to alleviate the issues that Lin-Search has with large PB constraints. When invoked on an instance $\mathcal{F}$ a linear search algorithm using varying resolution starts its search by creating a lower resolution version of $\mathcal{F}$ by dividing all weights of soft clauses by some constant d and removing all clauses $C \in \textsc{Soft}(\mathcal{F})$ for which $\lfloor w^{\mathcal{F}}(C)/d \rfloor = 0$. The low resolution version is then solved by standard linear search. When an optimal solution is found, the value of d is decreased and the search continued in higher resolution. Following LinSBPS, we used the generalized totalizer encoding (GTE) [27] to convert the PB constraints to CNF. Given a set of input literals $L = \{l_1, \ldots, l_n\}$ and their corresponding weights $W = \{w_1, \ldots w_n\}$ the GTE creates a set of output literals $o_1, \ldots o_k$ s.t. each o_i corresponds to a sum s_i formable with the weights in W for which $s_i < s_j$ if $i < j$. The sum of weights of the literals in L set to true is then restricted to be less than s_i with the unit clause $(\neg o_i)$.

As the instantiation of Core-Guided we use the PMRES algorithm [44] extended with weight aware core extraction (WCE) [16] and the hardening rule. Weight aware core extraction is a heuristic designed to allow Core-Guided to extract multiple cores before reformulating the instance and increasing its complexity. When extracting a new core κ PMRES with WCE first computes $c^{\kappa} = \min\{w^{\mathcal{F}}(C) \mid C \in \kappa\}$, then lowers the weight of all clauses in κ by c^{κ} (removing all clauses with weight 0). When no new cores can be extracted, the Reformulate function is invoked on all of the found cores and the stratification bound is reset. The search continues until no new cores can be found after a reformulation step. This strategy corresponds to the S/to/WCE strategy of [16]. While an alternative strategy that prefers reformulating to lowering the stratification bound was deemed more effective for complete MaxSAT solving in [16], we found that S/to/WCE is more effective for incomplete solving. For lowering the stratification bound, we use the *diversity heuristic* [7] that balances the amount that b^{STRAT} is lowered with the number of new soft clauses introduced.

In the next section, we report on a comparison of core-boosted linear search and all of the solvers that participated in the incomplete track of the 2018 MaxSAT Evaluation: LinSBPS, maxroster, SATLike, Open-WBO and Open-WBO-Inc and their variations. Most of them implement variations of an approach where: i) a heuristic of some kind if used to find a good initial solution to the instance being solved and ii) that solution is used to initialise a complete any-time algorithm. In most cases, the complete algorithm is some variant of Lin-Search. The solver SATLike [30] deviates from this description and instead uses local-search techniques in order to quickly traverse the search space

and look for solutions of increasing quality. A more detailed description of the solvers can be found on the evaluation homepage [14].

For related work from the field of complete MaxSAT solving, the Primal-Dual MaxSAT algorithm [18] extends PMRES with a second instance reformulation used to rule out solutions that falsify the same clauses as an intermediate solution obtained during search. The main two differences between Primal-Dual and core-boosted linear search are that Primal-Dual reformulates the instance on each iteration, thus increasing the complexity of core extraction steps, and that the reformulation only rules out solutions that falsify a particular set of clauses. In contrast, lowering the bound on the PB constraint in Lin-Search rules out all solutions that have higher cost than the best known solution. The WMSU3 [38] algorithm maintains a cardinality constraint over soft clauses similar to Lin-Search but only relaxes a soft clause C after extracting a core κ for which $C \in \kappa$. The similar WPM3 [9] uses linear-search as a subroutine within core-guided search in order to obtain tighter bounds on the cardinality constraints.

In addition to core-guided and linear search, a third central approach to SAT-based MaxSAT solving is based on *implicit-hitting sets* [21, 45]. When solving, such solvers maintain a set of cores of the input instance. During each iteration, a minimum-cost hitting set over the set of cores is computed. The clauses in the hitting set are then removed from the instance and the SAT solver invoked on the remaining clauses. If the SAT solver reports satisfiable, the obtained solution is optimal. Otherwise, a new core is obtained and the search continues. Finally, MaxSAT solvers based on branch and bound have been shown to be effective on random MaxSAT instances as well as challenging instances of smaller size. Such instances are encountered for example in combinatorics [31, 33, 1, 35, 2, 32, 48, 34].

6 Experimental Evaluation

Next we present the results on a experimental evaluation of a prototype core-boosted linear search algorithm that combines the instantiations of Lin-Search and Core-Guided discussed in Section 5. We refer to our implementation of Lin-Search extended with varying resolution and solution-guided phase saving by Linear-Search. Similarly, we use Core-Guided to refer to our implementation of PMRES extended with WCE and hardening. Finally, Core-Boosted-XXs is the core-boosted algorithm that first runs Core-Guided until either XX seconds have passed or no more cores can be found with the stratification bound at 1, then reformulates the instance and solves the reformulated instance with Linear-Search. The state of the internal SAT solver of Core-Boosted-XXs is kept throughout the core-guided phase, but reset (that is learned clauses are eliminated and activities of all variables reset to 0) when execution is switched to the linear search phase and whenever resolution is increased during the linear phase.

All three algorithms were implemented on top of the publicly available Open-WBO system [40] using Glucose 4.1 [12] as the back-end SAT solver. The initial

solution of all three algorithms is obtained by invoking the SAT solver on the hard clauses of the instance being solved. We emphasise that core-boosted linear search is a general idea applicable with all implementations and extensions of Lin-Search and Core-Guided that we are aware off. The goal of these experiments is to show that core-boosting can be used to improve performance of modern core-guided and linear search solvers, not to evaluate different instantiations and extensions of Core-Guided and Lin-Search.

Our experimental setup is similar to the 300s weighted incomplete track of the 2018 MaxSAT evaluation [14]. In most of the experiments, we use the 172 benchmarks from the weighted incomplete track of the evaluation, available from https://maxsat-evaluations.github.io/2018/benchmarks.html. We enforce a per-instance time limit of 300 seconds and memory limit of 32GB. All of the experiments were run on the StarExec cluster (https://www.starexec.org) that has 2.4-GHz Intel(R) Xeon(R) E5-2609 0 quad-core machines with 128-GB RAM. As the metric for comparing solvers we use the same incomplete score as the evaluation. For an instance $\mathcal{F}$ let BEST-COST($\mathcal{F}$) denote the lowest cost found in 300 seconds by any of (the variants of) the solvers Linear-Search, Core-Guided, Core-Boosted-XXs or the solvers that participated in the evaluation. The score a solver S on $\mathcal{F}$ is defined as the ratio between BEST-COST($\mathcal{F}$) and the cost of the best solution τ^S to $\mathcal{F}$ found by S, i.e. $\textsc{Score}(S, \mathcal{F}) = \frac{\text{BEST-COST}(\mathcal{F})+1}{\text{COST}(\mathcal{F},\tau^S)+1}$. In other words, the score of S is the ratio between the cost of the solution of the virtual-best-strategy (VBS) among our methods and the MaxSAT Evaluation 2018 solvers, and the cost obtained by S. Hence the score difference between two solvers shows the percentage points by which the better solver is closer to the VBS.

The first experiment we report on evaluates effect of different time limits on the core-guided phase of Core-Boosted-XXs. As limits we chose 30 seconds (10% of the total time), 75s (25%), 150s (50%), 225s (75%) and 300s (100%), respectively. An important fact to keep in mind is that the core-guided phase can end earlier than the limit. For example, the solver Core-Boosted-150s runs its core-guided phase until no more cores can be found with the stratification bound at 1 *or* 150 seconds have elapsed.

Table 1 lists the average score obtained by the Core-Boosted-XXs (CB-XXs in the table) solver for different values of XX. Overall we observe a decrease in the average score when the time limit is increased, even if the effect is small in most domains. A possible explanation for this behavior is offered by Figure 1 showing the duration of the core-guided phase of the Core-Boosted-300s solver on all benchmarks. On 107 out of the 172 benchmarks, the core-guided phase ended within 30 seconds and on 38 benchmarks Core-Boosted-300s did not enter its linear search phase at all. In other words, on a clear majority of the benchmarks, the duration of core-guided phase was either very short or very long, which explains the good performance of Core-Boosted-30s. For the rest of the experiments, we fix the time limit for the core-guided phase to 30 seconds. Table 1 also lists the average score obtained by the two components of Core-Boosted individually. The scores clearly demonstrate the potential of core-boosted linear

Table 1: Average score obtained by Core-Boosted-XXs with different maximum times for the core-guided phase as well as its core-guided and linear search components. In the table CB-XXs refers to the Core-Boosted-XXs solver, Lin to the Linear-Search solver and CG to the Core-Guided solver.

Domain (#benchmarks)	**CB-30s**	**CB-75s**	**CB-150s**	**CB-225s**	**CB-300s**	**CG**	**Lin**
BTBNSL (16)	**0.996**	0.995	**0.996**	0.995	0.965	0.956	0.959
abstraction-refinement (2)	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.517
af-synthesis (19)	0.990	0.990	0.990	0.990	0.990	0.944	**0.991**
causal-discovery (14)	0.776	0.776	0.799	**0.803**	0.795	0.563	0.454
cluster-expansion (20)	**0.941**	**0.941**	**0.941**	**0.941**	**0.941**	**0.941**	**0.941**
correlation-clustering (12)	0.953	**0.956**	0.953	0.953	0.953	0.736	0.675
hs-timetabling (13)	0.701	0.655	0.566	0.459	0.144	0.076	**0.717**
lisbon-wedding (12)	**0.582**	**0.582**	**0.582**	**0.582**	**0.582**	0.544	**0.582**
maxcut (11)	**0.892**	**0.892**	**0.892**	**0.892**	**0.892**	0.594	0.884
min-width (16)	0.961	**0.965**	0.962	0.956	0.962	0.825	0.898
miplib (5)	**0.587**	**0.587**	0.584	0.584	0.444	0.309	0.571
power-distribution (2)	**0.704**	**0.704**	**0.704**	**0.704**	**0.704**	0.497	0.484
railway-transport (4)	0.927	0.923	0.916	0.920	**0.935**	0.708	0.906
relational-inference (2)	0.041	0.041	0.041	0.041	**0.429**	0.414	0.041
robot-nagivation (3)	**0.943**	**0.943**	**0.943**	**0.943**	0.000	0.000	**0.943**
spot5 (3)	0.990	0.990	0.990	0.990	0.990	0.914	**0.999**
staff-scheduling (10)	**0.895**	**0.895**	0.863	0.840	0.493	0.385	0.877
tcp (7)	**1.000**	0.998	0.998	**1.000**	**1.000**	0.864	0.988
timetabling (1)	0.667	0.148	0.130	0.131	0.131	0.026	**0.941**
Total (172)	**0.870**	0.864	0.857	0.847	0.785	0.680	0.807

search. The average score of Core-Boosted-30s is higher than either Core-Guided (CG in the table) or Linear-Search (Lin in the table) on 10 out of 19 domains and equal to its better component on 3 more.

Figure 2 shows a detailed analysis on the behaviour of core-boosted linear search in the form of plots showing the evolution of the gap between the upper and lower bound (in logscale) of Core-Boosted-30s, Linear-Search and Core-Guided on three hand-picked benchmarks. The benchmark on the left shows a case where core-guided search is effective. During the first 30 seconds, both Core-Boosted-30s and Core-Guided rapidly decrease the gap. After 30 seconds, Core-Boosted-30s switches to its linear search phase, which on this benchmark slows its search progression. Core-Guided continues with the same search strategy, finding (and proving optimality of) a solution of cost 76250 in just under 190 seconds. Even if the gap of Core-Boosted-30s is larger due to a smaller lower bound, it still finds an "almost optimal" solution having cost 76251. On this benchmark Linear-Search is unable to improve on its initial solution at all and returns a solution with cost 226338. An important observation to make is that, in contrast to Linear-Search, Core-Boosted-30s did manage to improve its solution also in the linear phase. This indicates that the linear search phase of core-boosted search can indeed benefit from the reformulation steps performed and the best solution obtained during the core-guided phase.

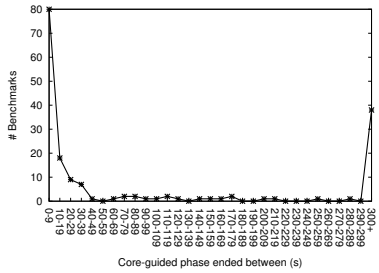

Fig. 1: Time spent in core-guided phase by Core-Boosted-300s.

The benchmark in the middle of Figure 2 demonstrates the opposite behaviour to the one on the left. On this benchmark Core-Guided is unable to improve on its initial solution having cost 651, while Linear-Search continuously improves it and ends up finding one that has cost 17. Core-Boosted-30s is initially unable to make progress, but starts decreasing its gap when switching to the linear phase after 30s and ends up finding a solution of cost 23. Finally, the benchmark on the right demonstrates a best-case scenario for core-boosted search. On this benchmark Linear-Search is unable to improve at all on its initial solution that has cost 311544. Core-Guided is able to decrease the gap by increasing the lower bound to 104585, but is unable to find a single better solution and returns the initial solution of cost 311544 as well. Core-Boosted-30s is able to use the best of both worlds by first increasing the lower bound during the core-guided phase and then switching to the linear phase in order to find a solution of cost 171437, significantly better than either of its components. Notice that the initial solution given to the linear phase of Core-Boosted-30s is the same as the one found by Linear-Search, so the performance difference between the two is only due to the reformulation steps done during core-guided search.

The results shown in Figure 2 suggest, that a more sophisticated strategy for deciding when to switch from the core-guided to the linear phase could be used to further improve the empirical performance of core-boosted linear search. Even though the instances in Figure 2 are hand-picked, the average scores over all benchmarks in the corresponding domains listed in Table 1 support the observations. For example, the instances in the hs-timetabling domain (Figure 2, middle) tend to contain only a few very large cores that are difficult to extract, making them well suited for approaches that compute solutions. On the other hand, instances in the causal-discovery domain (Figure 2, right) contain very many small cores that make finding good intermediate solutions to them difficult without first ruling out some of the cores with core guided search.

Figure 3 shows a per-instance comparison of the score obtained by Core-Boosted-30s and four variants of it: 1) Core-Boosted-30s-no-reformulation that ignores the reformulated instance and invokes the linear phase on the original instance, 2) Core-Boosted-30s-no-solution that ignores the best solution obtained during the core-guided phase in the linear phase and instead initialises a new

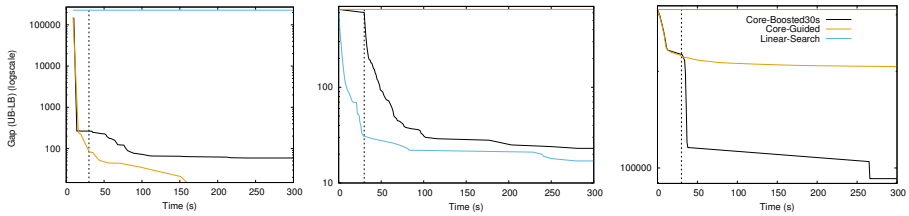

Fig. 2: Evolution of the gap between the upper and lower bound during search. The specific benchmarks shown are abstraction-refinement-downcast-antlr (left) [50], hs-timetabling-BrazilInstance5.xml (middle) [22], causal-discovery-causal_carpo_8_100 (right) [26].

solution by invoking the SAT-solver on the hard clauses of the reformulated instance, 3) Core-Boosted-30s-keep-SAT-solver that keeps the state of the internal SAT solver throughout the entire search and 4) Core-Boosted-30s-wce-to-strat that uses of the original search strategy proposed in [16] during the core-guided phase. In all plots Core-Boosted-30s is on the y-axis, so any data points in the upper left triangle correspond to benchmarks on which the baseline performed better than the variant. We observe that the baseline solver performs better than all of its variants, justifying our design choices. The results suggest that using the reformulated instance and initialising the Linear Search with the best solution obtained during core-guided search are especially important for the overall performance.

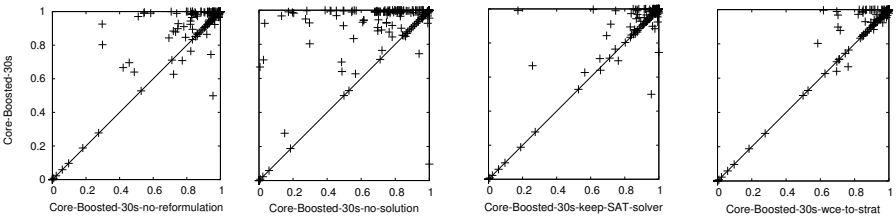

Fig. 3: The effect of different factors of Core-Boosted-30s on the overall performance.

Finally, we compare Core-Boosted-30s and its components to the other solvers that participated in the 2018 evaluation. Due to running our experiments in the same environment as the evaluation, we did not rerun the other solvers but instead compared our solvers directly to the results of the evaluation. Figure 4 demonstrates the performance of our solvers on the 300s weighted (left) and unweighted (right) tracks[5]. We observe that Core-Boosted-30s performs very well in the weighted track, improving the previous state-of-the-art (LinSBPS) by ap-

[5] A consequence of the metric we use is that the scores of the other solvers we report are lower than in the evaluation. Their relative ranking is however the same.

proximately 2% while also finishing 3rd in the unweighted category. In more detail, out of the 172 weighted instances, Core-Boosted-30s and LinSBPS are equal on 63 instances (36%), Core-Boosted-30s finds a solution of strictly lower cost on 65 (37%), and LinSBPS on 44 (25%). We also evaluated our solvers in the 60s track of the evaluation, i.e. with the time out set to 60 seconds. In the weighted track, Core-Boosted-30s gets the average score 0.814 which is again highest of all solvers followed by Open-WBO-Inc-BMO (0.793). In the unweighted track, the average score of Core-Boosted-30s is 0.696 which is second highest after SATLike-c (0.699).

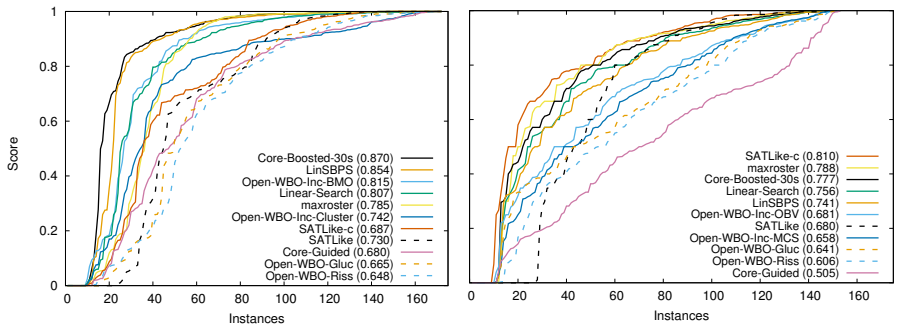

Fig. 4: Performance of Core-Boosted-30s, Linear-Search and Core-Guided compared to the results of the 300s weighted (left) and unweighted (right) track of the 2018 MaxSAT Evaluation.

7 Conclusions

We proposed core-boosted linear search, a novel search strategy for incomplete MaxSAT solving, that combines the strengths of core-guided and linear search and is, to the best of our knowledge, the first effective application of core-guided reformulation techniques in incomplete MaxSAT solving. Our experimental evaluation on a prototype implementation indicates that the information flow between the two phases of a core-boosted linear search solver often allows it to perform better than either of its individual components, while very rarely performing significantly worse. Furthermore, our comparison to other incomplete solvers shows that core-boosted linear search can be used to obtain state-of-the-art performance in weighted incomplete MaxSAT solving. As future work we plan to develop more dynamic ways of deciding when to switch between the core-guided and the linear search phase. Another interesting research directions to consider is the inclusion of MaxSAT preprocessing before, or even in-between, the core-guided and linear phases. Finally we also plan to look into extensions of core-boosted linear search to other constraint optimization paradigms.

References

1. Abramé, A., Habet, D.: AHMAXSAT: Description and evaluation of a branch and bound MaxSAT solver. Journal on Satisfiability, Boolean Modeling and Computation **9**, 89–128 (2015)
2. Abramé, A., Habet, D.: Learning nobetter clauses in MaxSAT branch and bound solvers. In: Proceedings of the 28th International Conference on Tools with Artificial Intelligence. pp. 452–459. IEEE Computer Society (2016)
3. Achá, R.J.A., Nieuwenhuis, R.: Curriculum-based course timetabling with SAT and maxSAT. Annals of Operations Research **218**(1), 71–91 (2014)
4. Alviano, M., Dodaro, C., Ricca, F.: A MaxSAT algorithm using cardinality constraints of bounded size. In: Proc. IJCAI. pp. 2677–2683. AAAI Press (2015)
5. Ansótegui, C., Bonet, M., Levy, J.: Solving (weighted) partial MaxSAT through satisfiability testing. In: Proc. SAT. Lecture Notes in Computer Science, vol. 5584, pp. 427–440. Springer (2009)
6. Ansótegui, C., Bonet, M., Levy, J.: SAT-based MaxSAT algorithms. Artificial Intelligence **196**, 77–105 (2013)
7. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving SAT-based weighted MaxSAT solvers. In: Proc. CP. Lecture Notes in Computer Science, vol. 7514, pp. 86–101. Springer (2012)
8. Ansótegui, C., Didier, F., Gabàs, J.: Exploiting the structure of unsatisfiable cores in MaxSAT. In: Proc. IJCAI. pp. 283–289. AAAI Press (2015)
9. Ansótegui, C., Gabàs, J.: Wpm3: An (in)complete algorithm for weighted partial maxsat. Artificial Intelligence **250**, 37 – 57 (2017)
10. Argelich, J., Le Berre, D., Lynce, I., Marques-Silva, J., Rapicault, P.: Solving Linux upgradeability problems using Boolean optimization. In: Proc. LoCoCo. Electronic Proceedings in Theoretical Computer Science, vol. 29, pp. 11–22 (2010)
11. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks and their applications. In: Proc SAT. pp. 167–180 (2009)
12. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern sat solvers. In: Proc IJCAI. pp. 399–404. Morgan Kaufmann Publishers Inc. (2009)
13. Bacchus, F., Narodytska, N.: Cores in core based MaxSat algorithms: An analysis. In: Proc. SAT. Lecture Notes in Computer Science, vol. 8561, pp. 7–15. Springer (2014)
14. Bacchus, F., Järvisalo, M., Martins, R., et al.: Maxsat evaluation 2018. https://maxsat-evaluations.github.io/2018/ (2018), accessed: 2018-9-05
15. Berg, J., Järvisalo, M.: Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. Artificial Intelligence **244**, 110–143 (2017)
16. Berg, J., Järvisalo, M.: Weight-aware core extraction in SAT-based MaxSAT solving. In: Proc CP. Lecture Notes in Computer Science, vol. 10416, pp. 652–670. Springer (2017)
17. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, The Netherlands, The Netherlands (2009)
18. Bjørner, N., Narodytska, N.: Maximum satisfiability using cores and correction sets. In: Proc. IJCAI. pp. 246–252. AAAI Press (2015)
19. Bunte, K., Järvisalo, M., Berg, J., Myllymäki, P., Peltonen, J., Kaski, S.: Optimal neighborhood preserving visualization by maximum satisfiability. In: Proc. AAAI. pp. 1694–1700. AAAI Press (2014)

20. Chen, Y., Safarpour, S., Marques-Silva, J., Veneris, A.: Automated design debugging with maximum satisfiability. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **29**(11), 1804–1817 (2010)
21. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MaxSAT. In: Proc. SAT. Lecture Notes in Computer Science, vol. 7962, pp. 166–181. Springer (2013)
22. Demirović, E., Musliu, N.: MaxSAT based large neighborhood search for high school timetabling. Computers & Operations Research **78**, 172–180 (2017)
23. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation **2**(1-4), 1–26 (2006)
24. Guerra, J., Lynce, I.: Reasoning over biological networks using maximum satisfiability. In: Proc. CP. Lecture Notes in Computer Science, vol. 7514, pp. 941–956. Springer (2012)
25. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: Proc. AAAI. AAAI Press (2011)
26. Hyttinen, A., Eberhardt, F., Järvisalo, M.: Constraint-based causal discovery: Conflict resolution with answer set programming. In: Proc UAI. pp. 340–349. AUAI-Press (2014)
27. Joshi, S., Martins, R., Manquinho, V.M.: Generalized totalizer encoding for pseudo-boolean constraints. In: Proc. CP. LNCS, vol. 9255, pp. 200–209 (2015)
28. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: Qmaxsat: A partial max-sat solver. Journal on Satisfiability, Boolean Modeling and Computation **8**, 95–100 (2012)
29. Le Berre, D., Parrain, A.: The Sat4j library, release 2.2 system description. Journal on Satisfiability, Boolean Modeling and Computation **7**, 59–64 (2010)
30. Lei, Z., Cai, S.: Solving (weighted) partial maxsat by dynamic local search for sat. In: Proceedings of IJCAI. pp. 1346–1352 (2018)
31. Li, C.M., Manyà, F., Planes, J.: Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In: Proc. CP. Lecture Notes in Computer Science, vol. 3709, pp. 403–414. Springer (2005)
32. Li, C.M., Manya, F., Planes, J.: New inference rules for MaxSAT. Journal of Artificial Intelligence Research **30**(1), 321–359 (2007)
33. Li, C.M., Quan, Z.: An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence. vol. 10, pp. 128–133. AAAI Press (2010)
34. Lin, H., Su, K., Li, C.M.: Within-problem learning for efficient lower bound computation in MaxSAT solving. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence. pp. 351–356. AAAI Press (2008)
35. Liu, Y.L., Li, C.M., He, K., Fan, Y.: Breaking cycle structure to improve lower bound for MaxSAT. In: Proceedings of the 10th International Workshop on Frontiers in Algorithmics. Lecture Notes in Computer Science, vol. 9711, pp. 111–124. Springer (2016)
36. Marques-Silva, J., Janota, M., Ignatiev, A., Morgado, A.: Efficient model based diagnosis with maximum satisfiability. In: Proc. IJCAI. pp. 1966–1972. AAAI Press (2015)
37. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Annals of Mathematics and Artificial Intelligence **62**(3-4), 317–343 (2011)
38. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. CoRR **abs/0712.1097** (2007)

39. Marques-Silva, J., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: Proc ICCAD. pp. 220–227. IEEE Computer Society (1996)
40. Martins, R., Manquinho, V., Lynce, I.: Open-WBO: A modular MaxSAT solver. In: Proc. SAT. Lecture Notes in Computer Science, vol. 8561, pp. 438–445. Springer (2014)
41. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: Proc. CP. Lecture Notes in Computer Science, vol. 8656, pp. 564–573. Springer (2014)
42. Morgado, A., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J.: Iterative and core-guided maxsat solving: A survey and assessment. Constraints **18**(4), 478–534 (2013)
43. Nadel, A., Ryvchin, V.: Efficient SAT solving under assumptions. In: Proc SAT. LNCS, vol. 7317, pp. 242–255. Springer (2012)
44. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: Proc. AAAI. pp. 2717–2723. AAAI Press (2014)
45. Saikko, P., Berg, J., Järvisalo, M.: LMHS: A SAT-IP hybrid MaxSAT solver. In: Proc. SAT. LNCS, vol. 9710, pp. 539–546. Springer (2016)
46. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: Proceedings of CP-05, pp. 827–831. Springer (2005)
47. 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, pp. 466–483. Springer (1983)
48. Xing, Z., Zhang, W.: MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability. Artificial intelligence **164**(1-2), 47–80 (2005)
49. Zhang, L., Madigan, C.F., Moskewicz, M.H., Malik, S.: Efficient conflict-driven learning in a Boolean satisfiability solver. In: Proc ICCAD. pp. 279–285. IEEE Computer Society (2001)
50. Zhang, X., Mangal, R., Grigore, R., Naik, M., Yang, H.: On abstraction refinement for program analyses in datalog. In: Proc PLDI. pp. 239–248. PLDI '14, ACM, New York, NY, USA (2014)
51. Zhu, C., Weissenbacher, G., Malik, S.: Post-silicon fault localisation using maximum satisfiability and backbones. In: Proc. FMCAD. pp. 63–66. FMCAD Inc. (2011)