Refined Core Relaxation for Core-Guided MaxSAT Solving

Hannes Ihalainen
HIIT, Department of Computer Science, University of Helsinki, Finland

Jeremias Berg
HIIT, Department of Computer Science, University of Helsinki, Finland

Matti Järvisalo
HIIT, Department of Computer Science, University of Helsinki, Finland

Abstract

Maximum satisfiability (MaxSAT) is a viable approach to solving NP-hard optimization problems. In the realm of core-guided MaxSAT solving—one of the most effective MaxSAT solving paradigms today—algorithmic variants employing so-called soft cardinality constraints have proven very effective. In this work, propose to combine weight-aware core extraction (WCE)—a recently proposed approach that enables relaxing multiple cores instead of a single one during iterations of core-guided search—with a novel form of structure sharing in the cardinality-based core relaxation steps performed in core-guided MaxSAT solvers. In particular, the proposed form of structure sharing is enabled by WCE, which has so-far not been widely integrated to MaxSAT solvers, and allows for introducing fewer variables and clauses during the MaxSAT solving process. Our results show that the proposed techniques allow for avoiding potential overheads in the context of soft cardinality constraint based core-guided MaxSAT solving both in theory and in practice. In particular, the combination of WCE and structure sharing improves the runtime performance of a state-of-the-art core-guided MaxSAT solver implementing the central OLL algorithm.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Constraint and logic programming

Keywords and phrases maximum satisfiability, MaxSAT, core-guided MaxSAT solving

Digital Object Identifier 10.4230/LIPIcs.CP.2021.35

Supplementary Material Source code and experiment data are available at `https://bitbucket.org/coreo-group/cgss/`

Funding Work financially supported by Academy of Finland under grants 322869 and 342145.

1 Introduction

Maximum satisfiability (MaxSAT) [8, 19] has in recent years developed into a noteworthy declarative Boolean optimization paradigm, with successful applications in various NP-hard industrial problem domains and artificial intelligence applications (see e.g. [8] and references therein).

So-called core-guided MaxSAT algorithms form one of the most central MaxSAT solving paradigms today [15, 26, 25, 28, 12, 18, 5, 27, 4]. The core-guided approach consists of iteratively extracting unsatisfiable cores, i.e., inconsistent subsets of soft constraints, using a Boolean satisfiability (SAT) solver, and at each iteration transforming the MaxSAT instance to include knowledge of the new unsatisfiable cores. This transformation involves compiling the cores into the instance via additional cardinality constraints. A key aspect in which the various core-guided algorithm differ is how exactly the transformations are performed. The use of so-called soft cardinality constraints [1, 28, 26] has recently proven to be a particularly effective approach to core-guided MaxSAT solving [7, 6].

In this work, we focus on improving and understanding core-guided MaxSAT algorithms through fine-grained changes to the way the transformations at each iteration are performed and realized. Here we focus in particular on realizing the proposed improvements and understanding their effects in the context of OLL [26, 1], which gives one of the currently most successful MaxSAT solving approaches through its implementation in the RC2 MaxSAT solver [18], and arguably represents the current state of the art in complete core-guided MaxSAT solving. That said, the techniques proposed in this work are also applicable in the context of other current and foreseeable variants of the core-guided approach using soft cardinality constraints; we will more shortly provide evidence on this with an implementation of the PMRES [28] core-guided algorithm, in addition to OLL.

Our main contributions are centered around and build on the recently proposed approach of weight-aware core extraction (WCE) [11]. In short, WCE enables relaxing multiple cores instead of a single one during iterations of core-guided search by exploiting soft clause weights in weighted MaxSAT instances, and can be viewed as an extension of work on computing lower bounds for MaxSAT [16, 20].

As our main contribution, we propose a novel form of structure sharing in the cardinality-based core transformation steps performed in core-guided MaxSAT solvers. The proposed form of structure sharing is enabled by WCE. While WCE has so-far not been widely integrated to MaxSAT solvers, its combination with the here proposed structure sharing approach provides performance improvements to RC2, as we will empirically show. Syntactically, the structure sharing approach allows for introducing fewer variables and clauses during the MaxSAT solving process, which alleviates to an extent issues resulting from iteratively performing core transformation steps which in turn results in the working formulas of the algorithm increasing in size beyond the capabilities of modern SAT solvers. Here we note that, while so-called incremental cardinality constraints have been proposed and are applied in core-guided MaxSAT solvers [25, 24, 23] with the goal of keeping the working formulas smaller by more carefully introducing necessary parts of cardinality constraint encodings, the structure sharing approach we propose is a conceptually different technique. Structure sharing focuses of sharing substructures of *multiple* cores extracted via WCE during a *single* iteration of core-guided search. Furthermore, as we will show, shared substructures obtained via structure sharing allow for a more careful introduction of equivalence in the core transformations steps.

By a careful implementation on top of the state-of-the-art RC2 MaxSAT solver, we demonstrate that structure sharing and refined equivalences, combined with WCE, improves the runtime performance of RC2 on standard weighted MaxSAT benchmarks from MaxSAT Evaluations. Complementing the main practical contributions, we also provide a theoretical analysis of on the impact of integrating WCE into the OLL algorithm, more specifically on the effect that WCE has on the number of core extractions needed for termination and the sizes of cores extracted (as well as its potential drawbacks).

2 Maximum Satisfiability

A literal l is a Boolean variable x or its negation $\neg x$, the negation of a variable satisfies $\neg\neg x = x$. A clause C is disjunction (or set) of literals and a CNF formula F is a conjunction (or set) of clauses. A truth assignment τ maps Boolean variables to 0 (false) and 1 (true). τ is extended to literals l, clauses C and formulas F, respectively, by $\tau(\neg l) = 1 - \tau(l)$, $\tau(C) = \max\{\tau(l) \mid l \in C\}$ and $\tau(F) = \min\{\tau(C) \mid C \in F\}$. An assignment τ is a model of F if $\tau(F) = 1$. A formula is satisfiable if it has model, and otherwise unsatisfiable.

A MaxSAT instance $\mathcal{F}$ consists of two CNF formulas, the set of hard clauses $\textsc{hard}(\mathcal{F})$ and the set of soft clauses $\textsc{soft}(\mathcal{F})$, and a weight function $w\colon \textsc{soft}(\mathcal{F}) \to \mathbb{N}$ that assigns a positive weight to each soft clause. An assignment τ is a solution to $\mathcal{F}$ if it satisfies the hard clauses. The cost $\mathrm{COST}(\mathcal{F}, \tau)$ of τ is the sum of the weights of the soft clauses it falsifies, i.e. $\mathrm{COST}(\mathcal{F}, \tau) = \sum_{C \in \textsc{soft}(\mathcal{F})} (1 - \tau(C))w(C)$. A solution τ is optimal if $\mathrm{COST}(\mathcal{F}, \tau) \leq \mathrm{COST}(\mathcal{F}, \tau')$ for all solutions τ' to $\mathcal{F}$. The cost $\mathrm{COST}(\mathcal{F})$ of an instance $\mathcal{F}$ is the cost of its optimal solutions. In the rest of this paper, we assume all MaxSAT instances have solutions, i.e. that $\textsc{hard}(\mathcal{F})$ is satisfiable. In practice, the implementations of core-guided MaxSAT algorithms that we are aware off check this assumption by invoking a SAT solver on the hard clauses prior to search.

To simplify the discussion we will assume that every $C_i \in \textsc{soft}(\mathcal{F})$ is of form $(\neg b_i)$ for a variable b_i. The assumption can be made without loss of generality since one can always transform any soft clause $C \in \textsc{soft}(\mathcal{F})$ into the hard clause $C \vee b$ and the soft clause $(\neg b)$ with $w((\neg b)) = w(C)$. A variable b for which $(\neg b) \in \textsc{soft}(\mathcal{F})$ is a *blocking variable*. Let $\mathcal{B}(\mathcal{F})$ be the set of all blocking variables of $\mathcal{F}$. As setting $b = 1$ falsifies the soft clause $(\neg b)$, we can refer to soft clauses and blocking variables interchangeably, and extend the weight function w to blocking variables by $w(b) = w((\neg b))$. The cost of a solution τ is then $\mathrm{COST}(\mathcal{F}, \tau) = \sum_{b \in \mathcal{B}(\mathcal{F})} \tau(b)w(b)$. For a set $B \subset \mathcal{B}(\mathcal{F})$ we let $\textsc{minw}(B) = \min\{w(b) \mid b \in B\}$ be the smallest weight of the variables in B.

A central concept in modern MaxSAT solving is that of a (an unsatisfiable) core. Given a MaxSAT instance $\mathcal{F}$, a set $\kappa \subseteq \textsc{soft}(\mathcal{F})$ is a core if the formula $\textsc{hard}(\mathcal{F}) \wedge \kappa$ is unsatisfiable. As each soft clause is a unit clause containing the negation of a blocking variable, this implies that any solution to $\mathcal{F}$ sets $b = 1$ for at least one $b \in \mathcal{B}(\mathcal{F})$ for which $(\neg b) \in \kappa$. Hence we often view a core as a set of blocking variables (or a clause) $\{b \mid (\neg b) \in \kappa\}$ that is entailed by $\textsc{hard}(\mathcal{F})$. A core κ is minimal (an MUS) if $\textsc{hard}(\mathcal{F}) \wedge \kappa_s$ is satisfiable for all $\kappa_s \subset \kappa$.

▶ **Example 1.** Let n and r be two integers with $0 < r < n$, and $\mathcal{F}^{n,r}$ a MaxSAT instance such that $\textsc{hard}(\mathcal{F}^{n,r})$ contains clauses equivalent to $\sum_{i=1}^{n} b_i \geq r$ and $\mathcal{B}(\mathcal{F}^{n,r}) = \{b_1, \ldots, b_n\}$. In words, the clauses of $\mathcal{F}^{n,r}$ enforce that at least r soft clauses should be falsified (recall that assigning a blocking variable b to 1 corresponds to falsifying a soft clause). Assuming $w(b_i) = 1$ for all $i = 1 \ldots n$, any solution τ that sets exactly r variables in $\mathcal{B}(\mathcal{F}^{n,r})$ to 1 and the rest to 0 is an optimal solution to $\mathcal{F}^{n,r}$ and has $\mathrm{COST}(\mathcal{F}^{n,r}, \tau) = \mathrm{COST}(\mathcal{F}^{n,r}) = r$. Any $\kappa \subset \mathcal{B}(\mathcal{F}^{n,r})$ that contains at least $n - r + 1$ variables is a core. In order to see this note that since r blocking variables need to be set to 1—or equivalently r soft clauses need to be falsified—by any solution, it follows that any subset of blocking variables with at least $(n - r + 1)$ variables has to contain at least 1 that is set to 1 by any solution. The MUSes of $\mathcal{F}^{n,r}$ are the subsets of $\mathcal{B}(\mathcal{F}^{n,r})$ that contain exactly $n - r + 1$ variables.

Unit propagation is the main form of constraint propagation applied in CDCL SAT solvers. Consider a clause $C = \{l_1, \ldots, l_n\}$ and assume the value of l_i has been fixed to 0 for all $i = 1, \ldots, n - 1$. Then in order for C to be satisfied, the value of the literal l_n needs to be fixed to 1. We say that $l_n = 1$ is (unit) propagated by the clause C and the assignments $l_i = 0$ for $i = 1, \ldots, n - 1$. CDCL SAT solvers perform unit propagation whenever the current partial assignment sets all but one literal from a clause to false (0).

3 MaxSAT by Soft Cardinality Constraints

Our main focus is on the OLL algorithm, which is one of the most successful core-guided MaxSAT solving approaches using so-called soft cardinality constraints. Algorithm 1 represents a generic abstraction of the core-guided approach using soft cardinality constraints to

Algorithm 1 CG, a generic view on core-guided MaxSAT solving with soft cardinality constraints.

```
Input: A MaxSAT instance $\mathcal{F}$
Output: An optimal solution $\tau$ to $\mathcal{F}$
1  begin
2      $\mathcal{F}^1 \leftarrow \mathcal{F}$
3      for $i = 1, \ldots$ do
4          $(res, \kappa, \tau) \leftarrow \textsc{Extract-Core}(\mathcal{F}^i, \mathcal{B}(\mathcal{F}^i))$
5          if $res$="satisfiable" then return $\tau$
6          $w^\kappa \leftarrow \textsc{minw}(\kappa)$
7          for $b \in \kappa$ do
8              $w(b) \leftarrow w(b) - w^\kappa$
9          $\mathcal{F}^{i+1} \leftarrow \mathcal{F}^i \cup \textsc{Relax}(\kappa, w^\kappa)$
```

MaxSAT solving. When invoked on an instance $\mathcal{F}$, the algorithm begins by initializing a working instance $\mathcal{F}^1$ to $\mathcal{F}$ on Line 2. Then the main search loop (Lines 3-9) is started. In each iteration of the loop, a core of the current working instance $\mathcal{F}^i$ is extracted on Line 4 using the assumption interface of the underlying SAT solver [14]. In Algorithm 1 the use of the SAT solver is abstracted into the function Extract-Core that takes as input the current instance and its blocking variables (to use as assumptions). The function returns a triple (res, κ, τ). If res = "satisfiable" then τ is an assignment satisfying $\textsc{hard}(\mathcal{F}^i)$ that sets $\tau(b) = 0$ for each $b \in \mathcal{B}(\mathcal{F}^i)$. Such τ has $\text{COST}(\mathcal{F}^i, \tau) = 0$ and will be optimal for both $\mathcal{F}^i$ and $\mathcal{F}$, so Algorithm 1 terminates and returns it (Line 5).

If the SAT solver instead reports unsatisfiable, it will return a core κ of $\mathcal{F}^i$ expressed as a subset of $\mathcal{B}(\mathcal{F}^i)$ (i.e. as a clause over blocking variables). Next, the variables in κ are refined (Lines 6-8). During refinement, the weight of each blocking variable $b \in \kappa$ is lowered by $\textsc{minw}(\kappa)$. An important intuition here is that at least one of the variables in κ will have its weight set to 0, and will thus not be treated as a blocking variable in subsequent iterations. Refining the blocking variables essentially corresponds to *clause cloning* via assumptions (see, e.g., [11, 3, 21]), a common way for core-guided MaxSAT solvers to take soft clause weights into account.

After refining the blocking variables, the core κ is relaxed on Line 9 via the function Relax. Core-guided MaxSAT algorithms differ mainly in the specifics of the Relax function. For each κ of $\mathcal{F}^i$, at least one variable in κ needs to be assigned to 1 by any optimal solution to $\mathcal{F}^i$. Essentially all instantiations of Relax we are aware of relax the instance by adding new clauses and blocking variables to $\mathcal{F}$ that allow, in a controlled way, a single blocking variable in κ to be set to 1 in subsequent iterations. In this work we focus especially on the transformation used by the OLL core-guided MaxSAT algorithm [26, 1], defined as follows.

▶ **Definition 2.** *Given $(\kappa, w) = (\{b_1, \ldots, b_n\}, w)$, OLL implements Relax by adding $\mathcal{O}(n \log n)$ new variables and $\mathcal{O}(n^2)$ new clauses corresponding to $(\sum_{k=1}^{n} b_k \geq (i+1)) \rightarrow b_i^\kappa$ for $i = 1, \ldots, n-1$. Each b_i^κ is a new blocking variable of weight w.*

An informal argument for the correctness of OLL, i.e., the fact that the final assignment returned by it will be an optimal solution to $\mathcal{F}$, is as follows (a formal argument can be found in [26]). Since κ is a core of $\mathcal{F}^i$, any optimal solution of $\mathcal{F}^i$ will assign at least one $b \in \kappa$ to 1 (i.e., falsify at least one $(\neg b) \in \kappa$ on the clause-level). During refinement, at least one blocking variable will have its weight lowered to 0 and will not be considered a blocking variable in subsequent iterations. This means that the SAT solver is free to assign it to 1 in subsequent iterations. At the same time, the clauses added by Relax (detailed in Definition 2) enforce that for any $k > 1$ number of blocking variables in κ assigned to 1, $k - 1$ of the new blocking variables will also be unit propagated to 1, thus incurring more cost. In other words, the OLL relaxation allows one—but only one—of the blocking variables in κ to incur cost "for free" in subsequent iterations.

In the rest of the paper we use the name OLL to refer to Algorithm 1 with Relax instantiated as in Definition 2.

4 Structure Sharing for Improving Core-Guided MaxSAT Solvers

In this section we present *structure sharing*, the main contribution of this work. Structure sharing is a technique that enables more compact core relaxation steps within the OLL algorithm. In addition to requiring fewer clauses, we will also demonstrate that core relaxation with structure sharing enables more propagation in the underlying SAT solver during subsequent iterations.

We begin by discussing the totalizer encoding for cardinality constraints [9], a common way of realizing the OLL algorithm, and weight-aware core extraction [11], a refinement to the standard way in which OLL extracts cores. Both of these are central for understanding structure sharing.

4.1 The Totalizer Encoding of Cardinality Constraints

For our purposes, a totalizer $\mathcal{T}(V)$ over a set $V = \{v_1, \ldots, v_n\}$ of variables is a satisfiable CNF formula that defines a set $O(V) = \{o_0, \ldots, o_{n-1}\}$ of output variables. Informally speaking, the output variables count the number of input variables set to true by assignments satisfying $\mathcal{T}(V)$. More precisely, if τ satisfies $\mathcal{T}(V)$, then $\tau(o_k) = 1$ if and only if $\sum_{v \in V} \tau(v) \geq k + 1$. A common way of realizing the OLL algorithm—used for example in the state-of-the-art solver RC2—is to relax a core κ by adding a totalizer $\mathcal{T}(\kappa)$ to the instance and treating the output variables of $\mathcal{T}(\kappa)$ as blocking variables in subsequent iterations, i.e., setting $b_i^\kappa = o_i$.

$\mathcal{T}(\kappa)$ can be viewed as a tree with $|\kappa|$ leaves each associated with a distinct variable of κ. As an example, Figure 1 (left) gives the tree representation of the totalizer $\mathcal{T}(\kappa)$ over $\kappa = \{b_1, b_2, b_3, b_4\}$. The root of the tree is associated with the output variables of the totalizer, i.e., the new blocking variables added when relaxing κ. Notice that since that the definition of a core implies that at least one variable in κ is set to true by any solution, the output variable b_0^κ of $\mathcal{T}(\kappa)$ is often omitted in realizations of OLL, including RC2.

The non-leaf internal nodes are associated with the so-called linking variables of $\mathcal{T}(\kappa)$ used in the encoding of the semantics of the output variables. For an internal node D of a totalizer $\mathcal{T}(\kappa)$, let $\mathcal{S}(D) \subset \kappa$ be the set of variables associated with the leaves of the subtree rooted at D. The linking variables $\{d_0^{\mathcal{S}(D)}, \ldots, d_{|\mathcal{S}(D)|-1}^{\mathcal{S}(D)}\}$ associated with D count the number of variables of $\mathcal{S}(D)$ set to true by a satisfying assignment τ of $\mathcal{T}(\kappa)$. More precisely, if τ satisfies $\mathcal{T}(\kappa)$, then $\tau(d_k^{\mathcal{S}(D)}) = 1$ if and only if $\sum_{b \in \mathcal{S}(D)} \tau(b) \geq k+1$. Essentially, the linking variables of a totalizer can be viewed as the output variables of a sub-totalizer built over $\mathcal{S}(D)$.

Consider a totalizer $\mathcal{T}(\kappa)$ and one of its output variable b_k^κ. We say that the constraints encoding $\left(\sum_{b \in \kappa} b \geq k+1\right) \to b_k^\kappa$ are the *implication constraints* of b_k^κ. Analogously, we say that the constraints encoding $b_k^\kappa \to \left(\sum_{b \in \kappa} b \geq k+1\right)$ are the *equivalence constraints* of b_k^κ. The terminology is extended to linking variables $d_k^{\mathcal{S}(D)}$; the implication constraints

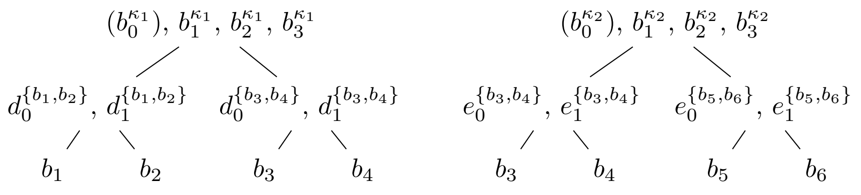
$(b_0^{\kappa_1}), b_1^{\kappa_1}, b_2^{\kappa_1}, b_3^{\kappa_1}$
$d_0^{\{b_1,b_2\}}, d_1^{\{b_1,b_2\}}$
$d_0^{\{b_3,b_4\}}, d_1^{\{b_3,b_4\}}$
b_1
b_2
b_3
b_4
$(b_0^{\kappa_2}), b_1^{\kappa_2}, b_2^{\kappa_2}, b_3^{\kappa_2}$
$e_0^{\{b_3,b_4\}}, e_1^{\{b_3,b_4\}}$
$e_0^{\{b_5,b_6\}}, e_1^{\{b_5,b_6\}}$
b_3
b_4
b_5
b_6

Figure 1 The structure of totalizers built when relaxing cores $\{b_1, b_2, b_3, b_4\}$ (left) and $\{b_3, b_4, b_5, b_6\}$ (right).

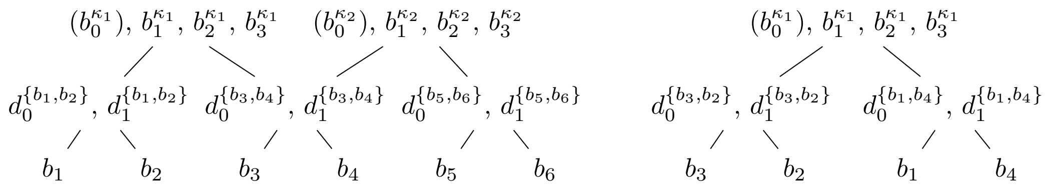
$(b_0^{\kappa_1}), b_1^{\kappa_1}, b_2^{\kappa_1}, b_3^{\kappa_1}$
$(b_0^{\kappa_2}), b_1^{\kappa_2}, b_2^{\kappa_2}, b_3^{\kappa_2}$
$d_0^{\{b_1,b_2\}}, d_1^{\{b_1,b_2\}}$
$d_0^{\{b_3,b_4\}}, d_1^{\{b_3,b_4\}}$
$d_0^{\{b_5,b_6\}}, d_1^{\{b_5,b_6\}}$
b_1
b_2
b_3
b_4
b_5
b_6
$(b_0^{\kappa_1}), b_1^{\kappa_1}, b_2^{\kappa_1}, b_3^{\kappa_1}$
$d_0^{\{b_3,b_2\}}, d_1^{\{b_3,b_2\}}$
$d_0^{\{b_1,b_4\}}, d_1^{\{b_1,b_4\}}$
b_3
b_2
b_1
b_4

Figure 2 Left: The structure of totalizers when relaxing the cores $\{b_1, b_2, b_3, b_4\}$ and $\{b_3, b_4, b_5, b_6\}$ with structure sharing. Right: An alternative totalizer for relaxing $\{b_1, b_2, b_3, b_4\}$.

of $d_k^{\mathcal{S}(D)}$ encode $\left(\sum_{b\in\mathcal{S}(D)} b \geq k+1\right) \rightarrow d_k^{\mathcal{S}(D)}$ and the equivalence constraints encode $d_k^{\mathcal{S}(D)} \rightarrow \left(\sum_{b\in\mathcal{S}(D)} b \geq k+1\right)$. The implication and equivalence constraints of the entire $\mathcal{T}(\kappa)$ are then the implication and equivalence constraints of each of its output and linking variables.

It is fairly straight-forward to show that for a realization of OLL that makes use of totalizers, it is sufficient to add only the implication constraints of the totalizers that relax cores. In fact—to the best of our understanding—most realizations of OLL that use totalizers do not add equivalence constraints when relaxing cores at all. The motivation for leaving out the equivalence clauses is to keep the size of the working instance smaller. While redundant in the context of computing optimal solutions of MaxSAT instances, the equivalence constraints of totalizers can however enable additional propagations in the SAT solver during subsequent iterations.

▶ **Example 3.** Consider the two totalizer structures $\mathcal{T}(\kappa_1)$ and $\mathcal{T}(\kappa_2)$, depicted on the left and right sides of Figure 1, respectively. Assume that we fix $b_1^{\kappa_1} = b_5 = 1$ and $b_2 = 0$. The implication clause $(\neg b_5 \vee e_0^{\{b_5,b_6\}})$ of $\mathcal{T}(\kappa_2)$ then propagates $e_0^{\{b_5,b_6\}} = 1$. This is the the only propagation that happens due to the implication constraints. The equivalence constraints of $\mathcal{T}(\kappa_1)$ additionally propagate $d_1^{\{b_1,b_2\}} = 0$ due to the clause $(\neg d_1^{\{b_1,b_2\}} \vee b_2)$ and $d_0^{\{b_3,b_4\}} = 1$ due to the clause $(\neg b_1^{\kappa_1} \vee d_1^{\{b_1,b_2\}} \vee d_0^{\{b_3,b_4\}})$.

4.2 Weight-Aware Core Extraction

Weight-aware core extraction (WCE) is an essential techniques towards structure sharing as proposed in this work. Originally proposed for the PMRES [28] core-guided MaxSAT algorithm in [11], WCE enables the extraction of multiple cores during a single iteration of core-guided MaxSAT solving by using information on clause weights during core extraction.

Algorithm 2 details the generic abstraction CG discussed in Section 3 extended with WCE. In short, the algorithm delays the core-relaxation steps as long as possible. When a core κ is obtained, the blocking variables in κ are refined as before. However, instead of immediately invoking RELAX, CG+WCE instead stores $(\kappa, \text{MINW}(\kappa))$ in a set $\mathcal{K}$ and invokes

Algorithm 2 CG+WCE: CG with WCE.

```
Input: A MaxSAT instance $\mathcal{F}$
Output: An optimal solution $\tau$ to $\mathcal{F}$
1  begin
2      $\mathcal{F}^1 \leftarrow \mathcal{F}$
3      for $i = 1, \dots$ do
4          $\mathcal{K} \leftarrow \emptyset$
5          while true do
6              $(res, \kappa, \tau) \leftarrow$ Extract-Core$(\mathcal{F}^i, \mathcal{B}(\mathcal{F}^i))$
7              if $res=$"satisfiable" then break
8              $w^\kappa \leftarrow$ minw$(\kappa)$
9              for $b \in \kappa$ do
10                 $w(b) \leftarrow w(b) - w^\kappa$
11             $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\kappa, w^\kappa)\}$
12         if $\mathcal{K} = \emptyset$ then return $\tau$
13         $\mathcal{F}^{i+1} \leftarrow \mathcal{F}^i \cup \bigcup_{(\kappa,w^\kappa)\in\mathcal{K}}$ Relax$(\kappa, w^\kappa)$
```

the SAT solver again. Recall that the weight of at least one of the blocking variables in κ will be lowered to 0 during variable refinement so it will not be treated as a blocking variable in subsequent iterations. In other words, the next call to Extract-Core is guaranteed to either obtain a new core, or report the instance to be satisfiable. When no new cores can be found (i.e. the solver reports satisfiable) the algorithm relaxes all cores stored in $\mathcal{K}$ before reiterating. Termination occurs when no new cores can be found between two relaxation steps, that is when $\mathcal{K} = \emptyset$ on line 12.

While WCE was shown to be effective for PMRES in [11], currently we are unaware of any implementations of OLL that would make use of WCE. However, as we will discuss next, WCE is instrumental in enabling structure sharing.

4.3 Structure Sharing for OLL

With the necessary background on totalizers and WCE in place, we turn to detailing structure sharing. In order to motivate structure sharing we consider what happens when several, possibly overlapping, cores are extracted and relaxed by OLL. In particular, consider an instance $\mathcal{F}$ with $\text{HARD}(\mathcal{F}) = \{(b_1 \lor b_2 \lor b_3 \lor b_4), (b_3 \lor b_4 \lor b_5 \lor b_6)\}$, $\mathcal{B}(\mathcal{F}) = \{b_1, \dots, b_6\}$, $w(b_1) = w(b_2) = w(b_5) = w(b_6) = 1$ and $w(b_3) = w(b_4) = 2$. The two cores of $\mathcal{F}$ that are important for our discussion are $\kappa_1 = \{b_1, b_2, b_3, b_4\}$ and $\kappa_2 = \{b_3, b_4, b_5, b_6\}$. Note that since $2 = w(b_3) = w(b_4) > \text{MINW}(\kappa_1) = \text{MINW}(\kappa_2) = 1$, both cores can be extracted by OLL. Figure 1 depicts two totalizers $\mathcal{T}(\kappa_1)$ and $\mathcal{T}(\kappa_2)$ that can be built when OLL relaxes κ_1 and κ_2. A key motivation for structure sharing is that both of these trees contain an internal node D for which $\mathcal{S}(D) = \{b_3, b_4\}$. As a consequence, the working instance obtained after relaxing both cores contains separate clauses defining the linking variables $d_0^{\mathcal{S}(D)}$ and $d_1^{\mathcal{S}(D)}$ and $e_0^{\mathcal{S}(D)}$ and $e_1^{\mathcal{S}(D)}$, even though the semantics of them are exactly the same.

By structure sharing in this example, we refer to sharing the subtree with the leaves $\{b_3, b_4\}$ between both $\mathcal{T}(\kappa_1)$ and $\mathcal{T}(\kappa_2)$, resulting in the structure depicted in Figure 2 (left). Generally, correctness of structure sharing follows from the fact that it does not alter the semantics of the new blocking variables added in core relaxation. In addition to decreasing the size of the working instance (for example, the structures depicted in Figure 1 require in total 24 clauses while the equivalent single structure shown in Figure 2 (left) only requires 21) structure sharing can also both *decrease* the number of—essentially unnecessary—propagations that the SAT solver does in subsequent iterations, as well as *increase* the number of further propagations. For an example of the former, note that when the two totalizers are disjoint (as in Figure 1), fixing one of the variables in $\{b_3, b_4\}$ to 1 propagates both $d_0^{\mathcal{S}(D)} = 1$ and $e_0^{\mathcal{S}(D)} = 1$. In contrast, with structure sharing (Figure 2, left) only the variable $d_0^{\mathcal{S}(D)} = 1$ is propagated. For an example of the latter, consider the following.

▶ **Example 4.** Consider the shared totalizer structure depicted in Figure 2 and assume again the fixings $b_1^{\kappa_1} = b_5 = 1$ and $b_2 = 0$. Similarly to Example 3, these propagate $d_0^{\{b_5,b_6\}} = d_0^{\{b_3,b_4\}} = 1$. However, in this structure, the implication clause $(\neg d_0^{\{b_3,b_4\}} \vee \neg d_0^{\{b_5,b_6\}} \vee b_1^{\kappa_2})$ also propagates $b_1^{\kappa_2} = 1$. We emphasize that $b_1^{\kappa_2} = 1$ can not be derived by unit propagation alone in the disjoint structures depicted in Figure 1.

Recall that, while the implication constraints of totalizers are needed in order to compute an optimal solution to a MaxSAT instance, the equivalence constraints are not. Instead, there is an inherent-trade off between the number of equivalence clauses that are added, and the number of additional propagations they enable. While extra propagations have the potential of speeding up subsequent SAT solver calls, adding too many clauses may instead end up slowing down the solver. Structure sharing seeks to limit the number of equivalence clauses added by identifying which ones of them are most likely to result in additional propagation. Example 4 offers some intuition on this. Note that propagating $b_1^{\kappa_2} = 1$ in the shared subtree does not require all of the equivalence constraints of the structure. Instead, the equivalence constraints of all nodes except for the ones in the subtree rooted at $d_0^{\{b_3,b_4\}}, d_1^{\{b_3,b_4\}}$ suffice. We detail the realization of structure sharing and the selective addition of equivalence constraints in the next section.

▶ Remark 5. We shortly note the distinguishing featuers of structure sharing as proposed to related recently proposed techniques from the literature. The *iterative encoding of totalizers* [23] allows for extending a single totalizer with more inputs. While commonly used in MaxSAT algorithms that extend cardinality constraints during search (such as MSU3 [3, 22]), in constrast to structure sharing, the iterative encoding it is not applicable to OLL or PMRES. The *WPM3 core-guided MaxSAT algorithm* [5] uses the semantics of blocking variables introduced during core relaxations to maintain knowledge of the global core structure in order to obtain a better encoding of any core containing blocking variables introduced in previous relaxation steps. However, in contrast to structure sharing, WPM3 does not maintain knowledge of blocking variables potentially being found in multiple cores. If a set of previously relaxed blocking variables are extracted as part of a core, the structure introduced by WPM3 will be disjoint from the structure introduced when originally relaxing the variables. In this sense structure sharing as we propose it here is orthogonal to the ideas of WPM3. Indeed, structure sharing and WCE could be integrated into WPM3 as well and appears interesting future work. The *abstract cores* technique proposed in the context of the implicit hitting set (IHS) approach to MaxSAT [10] also aims to build totalizers with variables that often appear in cores together being assigned to the same subtree. However, the abstraction sets over which totalisers are built in the abstract cores technique are not overlapping. In that sense, the way totalisers are used in the abstract cores technique resembles more closely the incremental encoding. Finally, the idea of sharing structure with the aim of obtaining *more effective representations of weight rules* (tightly connected to pseudo-Boolean constraints) in the context of answer set programming was explored in [13].

4.4 Realizing Structure Sharing

Structure sharing is realized in an OLL algorithm making use of WCE with a greedy procedure. Given a set $\mathcal{K}$ of cores to be relaxed, the procedure maintains a collection $\mathcal{S}$ of sets of blocking variables initialized to $\mathcal{K}$ and proceeds iteratively. In each iteration the sets in $\mathcal{S}$ are compared pairwise in order to find a maximal subset M of blocking variables shared by two sets in $\mathcal{S}$. After finding such M, the set $\mathcal{S}$ is refined by: i) adding a pointer $s \to M$ for each $s \in \mathcal{S}$ for which $M \subseteq s$, ii) removing the variables of M from every set $s \in \mathcal{S}$ for which $M \subseteq s$ and iii) adding M to $\mathcal{S}$. Finally, a totalizer is built for every set in $\mathcal{S}$ and linked following the pointers, i.e., if there is a pointer $a \to b$ for $a, b \in \mathcal{S}$, then $\mathcal{T}(b)$ is linked as a subtree to $\mathcal{T}(a)$.

We note that the use of WCE is central in enabling structure sharing. This is due to the fact that there are several possible totalizer structures for a core. Figure 2 (right) depicts another possible structure of $\mathcal{T}(\kappa_1)$ (from before) that does not include any subtrees that can be shared with $\mathcal{T}(\kappa_2)$ in Figure 1 (nor with any other possible structure of $\mathcal{T}(\kappa_2)$). The existence of different choices of equivalent totalizer structures makes it very difficult—if not impossible—to realize structure sharing without WCE. As an example, assume that OLL *without* WCE is invoked on $\mathcal{F}$ and the core κ_1 is extracted first. After refining the blocking variables, a totalizer $\mathcal{T}(\kappa_1)$ is built and added to the working instance. At this point, there is no obvious reason to prefer either one of the structures depicted in Figure 1 left or Figure 2 (right) for building $\mathcal{T}(\kappa_1)$; however, only the tree in Figure 1 enables structure sharing.

Selective Addition of Equivalences

After building shared totalizer structures for a set of cores, we next select which equivalence constraints should be included in the working instance. Intuitively, the aim is to add—in some sense—useful equivalence constraints that enable propagation without inflating the size of the working instance beyond the capabilities of modern SAT solvers.

Example 4 provides more intuition. The additional propagation enabled by structure sharing follows from the outputs of shared structures being propagated to 1. Furthermore, such outputs are propagated to 1 due to either (i) the implication constraints of the substructure itself (which have to be included anyway) or (ii) the equivalence constraints of the rest of the structure.

More generally, we propose to selectively include equivalence constraints by looping over the leaves of the structure—which are treated as roots of a subtree containing a single node—and the roots of any shared subtrees. For each node two values are computed: 1) the number of decisions needed before the equivalence constraints would propagate the shared variable to true; the decisions are either setting some leaves outside the subtree to false, or setting the output variables of the structure to true, and 2) the (estimated) total number of equivalence clauses for the nodes outside of the subtree, which is quadratic in the number of leaves in the structure outside of the subtree. If both of these numbers are below some user given parameter, we include the equivalence constraints of all variables outside of the subtree. Even in structures with shared subtrees both values are computed w.r.t. the totalizer tree corresponding to a single core.

▶ **Example 6.** Consider the shared structure in Figure 2 and the root of the shared subtree corresponding to the variables $d_0^{\{b_3,b_4\}}, d_1^{\{b_3,b_4\}}$. Assume that OLL has managed to derive the unit clause $(b_1^{\kappa_1})$, i.e. fix $b_1^{\kappa_1} = 1$. This could happen for example via the so-called core-exhaustion heuristic [18]. In order for the equivalence constraints of the structure corresponding to κ_1 to propagate $d_0^{\{b_3,b_4\}} = 1$ one more decision is needed, either $b_1 = 0$, $b_2 = 0$ or $b_2^{\kappa_1} = 1$. Similarly, in order for the equivalence constraints to propagate $d_1^{\{b_3,b_4\}} = 1$ two additional decisions are needed, one of the following four sets of alternatives: (i) $b_1 = b_2 = 0$, (ii) $b_2^{\kappa_1} = 1 \wedge b_1 = 0$, (iii) $b_2^{\kappa_1} = 1 \wedge b_2 = 0$, or (iv) $b_2^{\kappa_1} = b_3^{\kappa_1} = 1$. As for the the estimated number of equivalence clauses for the nodes outside of the subtree; in this case there are two leaves outside of the subtree (b_1, b_2) so the estimate is 4 constraints.

Similarly, in order for the equivalence constraints of the structure corresponding to κ_2 to propagate $d_0^{\{b_3,b_4\}} = 1$, two decisions are needed. One of the following four sets of alternatives: (i) $b_5 = b_6 = 0$, (ii) $b_1^{\kappa_2} = 1 \wedge b_5 = 0$, (iii) $b_1^{\kappa_2} = 1 \wedge b_6 = 0$, or (iv) $b_1^{\kappa_2} = b_2^{\kappa_2} = 1$. For propagating $d_1^{\{b_3,b_4\}} = 1$ three decisions are needed, one of the following four sets of alternatives: (i) $b_5 = b_6 = 0 \wedge b_1^{\kappa_2} = 1$, (ii) $b_1^{\kappa_2} = b_2^{\kappa_2} = 1 \wedge b_5 = 0$, (iii) $b_1^{\kappa_2} = b_2^{\kappa_2} = 1 \wedge b_6 = 0$, (iv) $b_1^{\kappa_2} = b_2^{\kappa_2} = b_3^{\kappa_2} = 1$. This structure also has two leaves outside of the shared tree (b_5, b_6), so the estimated number of added equivalence constraints is again 4.

5 Empirical Evaluation

We evaluate the impact of WCE and structure sharing on OLL. For the evaluation, we extending the state-of-the-art RC2 MaxSAT solver [18] implementation of OLL with WCE and structure sharing, using the RC2 version that performed best in MaxSAT Evaluation 2020. The implementation is available online at: `https://bitbucket.org/coreo-group/cgss/`.

More specifically, we compare the following solvers:

- **RC2**: the original RC2 solver from MaxSAT Evaluation 2020.
- **RC2***: our **RC2** refactorization with a reimplementation of totalizers i geared towards implementing WCE and structure sharing (SS).
- **RC2*+WCE**: **RC2*** extended with WCE.
- **RC2*+WCE+SS**: **RC2*+WCE** extended with SS and selective addition of equivalences.

In the implementation of SS, we stop the greedy procedure for computing shareable structures once the set M containing overlapping blocking variables has $|M| < 16$. Furthermore, for **RC2*+WCE+SS** we only add equivalence constraints of a node in a shared subtree if doing so adds at most 50 clauses and at most 50 decisions are needed in order for the variable to be propagated to 1. All of these parameter values were chosen based on preliminary experimentation.

As benchmarks we used the combined set of 1033 weighted MaxSAT instances from the complete tracks of MaxSAT Evaluation 2019 and 2020. The experiments were run single-threaded using 2.6-GHz Intel Xeon E5-2670 processors. A per-instance time limit of 3600 seconds and a memory limit of 32GB was enforced.

Figure 3 (top) shows the effect of WCE and structure sharing on RC2. First, we observe that our refactorization **RC2*** actually improves on the performance of **RC2**, improving on the number of solved instances from 745 to 752. Employing WCE and SS allows for solving the greatest number of 761 instances (**RC2*+WCE+SS**). The marginal impact of SS is significant, as employing WCE alone allows for solving 756 instances (**RC2*+WCE**) (As a side-note, to the best of our understanding, this is the first time that WCE is integrated to OLL and shown to provide performance improvements).

For more details on impact of SS and in particular the idea of selective addition of equivalences, consider Figure 3 (bottom). First recall that **RC2*+WCE+SS** employs selective addition of equivalence. Now, **RC2*+EQ**, **RC2*+WCE+EQ** correspond to

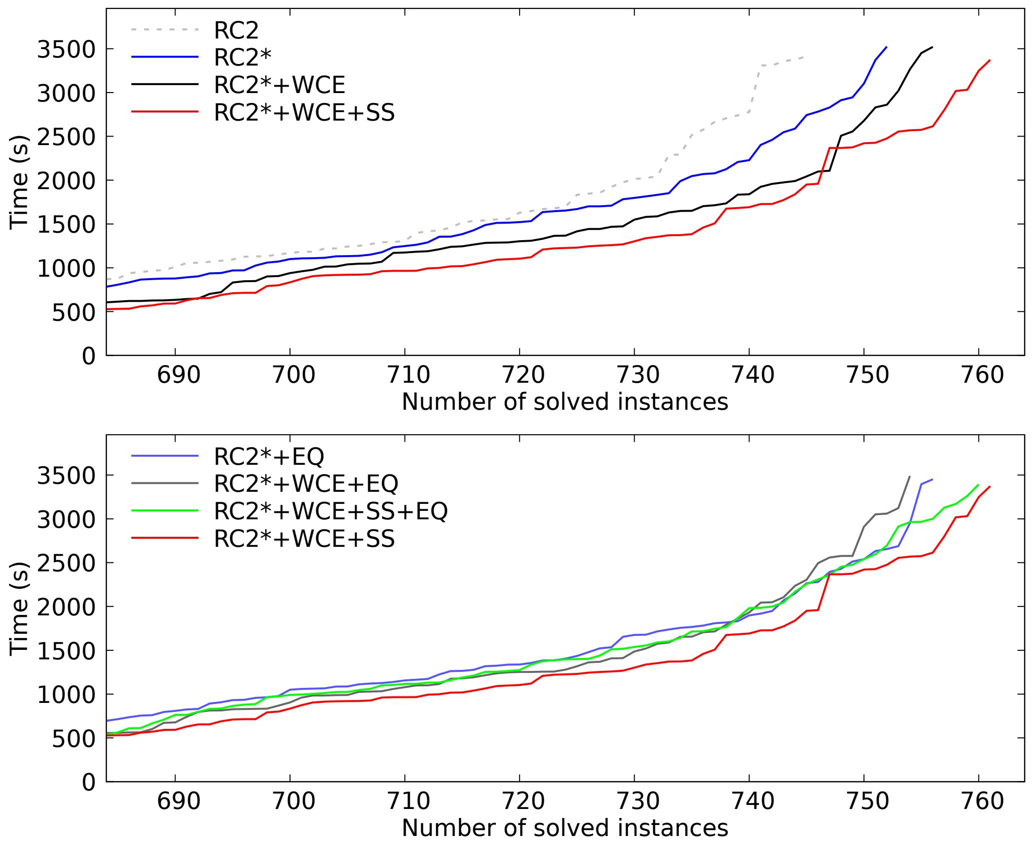

Figure 3 Top: The effect of WCE and structure sharing (SS) on OLL. Bottom: Effect of equivalence constraints on the different variants of OLL.

RC2* and **RC2*+WCE**, respectively, which add all equivalence constraints of each totalizer when relaxing cores. The solver **RC2*+WCE+SS+EQ** corresponds the modification of **RC2*+WCE+SS** that adds all equivalence constraints of each totalizer. Somewhat surprisingly, adding all equivalence constraints to the totalizers in **RC2*** actually increases performance in our evaluation: **RC2*+EQ** solves 756 instances. In contrast, including all equivalence constraints into **RC2*+WCE** degrades performance, **RC2*+WCE+EQ** solves 754 instances. Furthermore, **RC2*+WCE+SS+EQ** exhibits slightly weaker performance than **RC2*+WCE+SS**, solving in total 760 instances.

Further detailed results are provided in Table 1. Here we detail for each of the 63 benchmark domains within the benchmark set, the number of instances solved and the PAR-2 score (i.e. cumulative runtimes, with timeouts penalized by a factor of 2) of **RC2***, **RC2*+WCE** and **RC2*+WCE+SS**. We observe that **RC2*+WCE+SS** solves the greatest number of instances in 5 domains and obtains the best (lowest) PAR-2 score in 22 further domains, achieving overall a PAR-2 score that is $\sim 98\%$ of the PAR-2 score of **RC2*+WCE** and $\sim 96\%$ of the PAR-2 score of **RC2***.

Figure 4 demonstrates the relative number of variables and clauses added during the solving process. In more detail, Figure 4 (top) provides a comparison of the the relative of implication constraints added during solving by **RC2*+WCE** and **RC2*+WCE+SS**. Here the height of the bar at a x-axis value p gives the number of instances for which the number of implication constraints introduced by **RC2*+WCE+SS** was $p\%$ of the number

Table 1 Detailed comparison of RC2*, RC2*+WCE and RC2*+WCE+SS per benchmark domain.

Benchmark family	**RC2***		**RC2*+WCE**		**RC2*+WCE+SS**	
	solved	PAR-2	solved	PAR-2	solved	PAR-2
wpms	3	**18.12**	3	19.68	3	18.18
dalculus	27	**21.15**	27	21.50	27	21.26
MLIC	5	58737.43	5	**57821.14**	5	57885.03
causal-discovery	18	62988.31	18	61955.50	**19**	**56543.76**
relational-inference	5	26912.08	5	27023.60	5	**26886.39**
tcp	20	**23698.21**	20	26803.44	19	30008.36
protein_ins	11	738.45	11	697.47	11	**564.86**
staff-scheduling	2	74402.74	2	74108.37	2	**73207.89**
planning	4	**1.69**	4	1.85	4	1.82
scSequencing	19	**80608.19**	19	80678.55	19	80650.86
ParametricRBAC	32	**59118.70**	32	59603.27	32	59639.39
InterpretableClassifiers	7	36057.44	7	36062.51	7	**36056.42**
auc_paths	8	910.97	8	**310.29**	8	396.19
csg_2020	30	231.39	30	277.10	30	**230.93**
scp	10	953.63	10	610.38	10	**496.72**
preference_planning	16	**236.30**	16	372.53	16	403.98
pseudoBoolean	8	**7201.04**	8	7201.07	8	7201.07
lisbon-wedding	6	**131107.84**	6	131112.42	6	133287.46
frb	19	**175.57**	19	1501.58	19	2168.53
qcp	9	**5.04**	9	5.05	9	5.05
log	3	43204.83	3	43202.23	3	**43201.83**
miplib	2	**43294.76**	2	43320.13	2	43297.22
warehouses	8	217.34	8	154.79	8	**69.18**
haplotyping-pedigrees	29	691.74	29	733.20	29	**668.02**
min-width	8	240055.37	8	238353.95	8	**237878.82**
drmx-atmostk	17	516.92	17	675.43	17	**443.45**
upgradeability	19	111.14	19	126.34	19	**110.18**
correlation-clustering	10	96843.58	11	**89478.17**	11	90682.68
maxcut	0	43200.00	0	43200.00	0	43200.00
packup	5	30.83	5	30.59	5	**28.12**
abstraction-refinement	11	9251.50	11	10153.65	11	**8655.52**
railroad_sc	0	43200.00	0	43200.00	0	43200.00
security-witness	30	10481.58	30	9973.36	30	**9291.70**
rna-alignment	23	**99.31**	23	133.67	23	100.90
drmx-cryptogen	17	**2432.11**	17	2541.40	17	2456.42
CSG	10	**294.89**	10	315.85	10	329.77
css-refactoring	11	179.67	11	243.71	11	**166.37**
Security-CriticalCyber	39	**232.85**	39	262.01	39	235.87
ramsey	2	93978.71	2	94221.14	**3**	**89300.00**
railroad_scheduling	0	57600.00	0	57600.00	0	57600.00
wcsp	21	7.39	21	**6.95**	21	7.07
set-covering	9	**14668.38**	9	14795.22	9	14719.04
max-realizability	25	47187.54	26	40544.80	26	**40382.86**
MinWeightDomSet	0	50400.00	0	50400.00	0	50400.00
BTBNSL	7	123625.64	8	118890.94	8	**118397.70**
auc_regions	6	9280.05	6	9315.44	**7**	**1749.98**
mpe	24	38815.48	26	23566.10	**27**	**17579.18**
max-prob-min-cuts	30	**113.40**	30	131.24	30	114.82
hs-timetabling	2	83441.11	2	**82579.77**	2	83541.69
metro	27	3971.78	27	**3713.86**	27	3833.46
timetabling	14	60924.14	15	56410.41	15	**55920.20**
spot5	8	22237.25	**9**	**15141.17**	8	21657.60
wcnf_gz	8	**64828.57**	8	64847.09	8	64841.45
up-up98	6	15.90	6	16.55	6	**14.97**
scpnr	0	14400.00	0	14400.00	0	14400.00
dimacs_mod	1	**86516.74**	1	86589.22	1	86609.38
af-synthesis	**17**	**59316.18**	15	65557.61	15	66009.69
railway-transport	**2**	**23206.20**	1	28971.47	1	28948.40
RBAC	**9**	**154220.94**	7	166452.75	8	160087.41
shiftdesign	16	**6726.73**	16	8060.46	16	6949.01
auctions	11	34949.24	13	21297.08	**15**	**4909.61**
binaryNN	4	12465.98	4	12696.34	4	**12413.52**
dir	2	8171.10	2	**7317.87**	2	7373.82
SUM	752	2169531.16	756	2135809.26	**761**	**2097451.06**

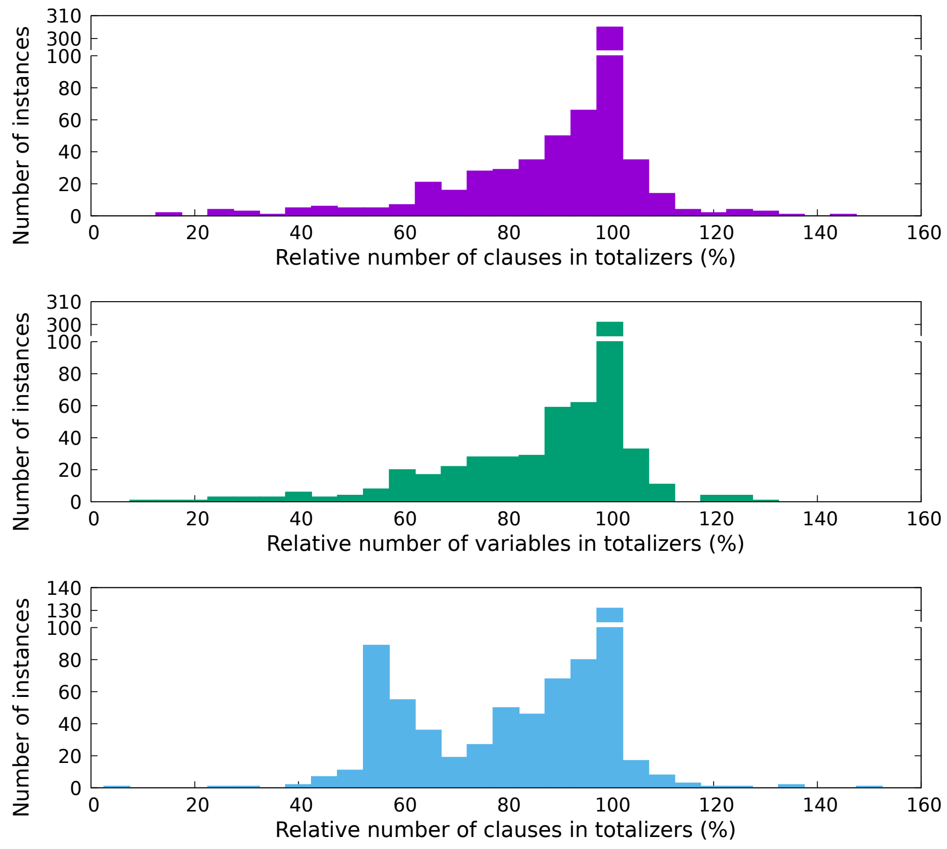

Figure 4 Impact of structure sharing on the number of implication clauses (top) and the number of variables (middle); impact of selective addition of equivalences on the number of equivalence constraints (bottom).

of implication constraints introduced by **RC2*+WCE**. We observe that structure sharing indeed decreases the number of implication constraints added on a significant number of instances. At times **RC2*+WCE+SS** adds less than 20% of the implication constraints added by **RC2*+WCE**. Figure 4 (middle) analogously provides the relative number of variables introduced. We again observe that for many benchmark instances the working instance of **RC2*+WCE+SS** contains fewer variables than the one of **RC2*+WCE**. Finally, Figure 4 (bottom) provides the relative number of equivalence constraints introduced by **RC2*+WCE+SS** and **RC2*+WCE+SS+EQ**. We observe that selective addition of equivalences introduces fewer clauses in many instances, which together with the runtime results presented in Figure 3 (bottom) suggests that the technique achieves the desideratum of introducing fewer equivalence clauses without sacrificing the potential benefits (for example, in the form of additional propagations).

Finally, we note that structure sharing is a general technique and not limited to the OLL algorithm. In order to demonstrate this, we reimplemented the PMRES algorithm in the same framework as RC2 (based on PySAT [17]). Figure 5 demonstrates the effect of WCE and structure sharing on the PMRES algorithm. We observe that both WCE and SS have a significant positive impact on the performance of PMRES, improving from 674 instances solved by the base algorithm to 679 when extended with WCE and further improving the number of solved instanced to 693 when extended by SS.

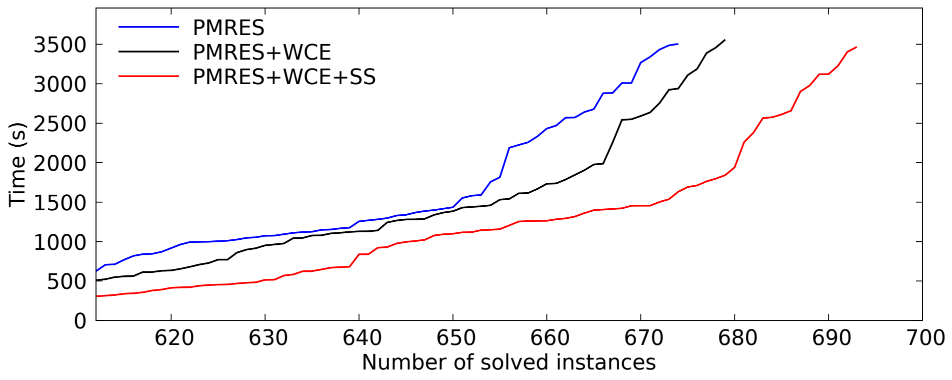

Figure 5 The effect of WCE and structure sharing (SS) on PMRES

6 Analysis on the Impact of WCE on OLL

Finally, complementing the empirical observations on the impact of WCE on OLL, we provide theoretical insights into the effects of weight-aware core extraction on the OLL algorithm. In particular, while the original paper proposing WCE [11] demonstrated its effectiveness on the PMRES algorithm in practice, it did not provide insight into the effect that WCE can have on core-guided MaxSAT algorithms on a theoretical level. In this section, we show that WCE can both decrease, and increase the number of iterations required by the OLL algorithm, depending on the instance. (We note that these observations extend to PMRES in a relatively straightforward way.)

In the following, let OLL be Algorithm 1 with Relax instantiated according to Definition 2 and OLL+WCE the OLL algorithm extended with WCE.

The following observation will be useful in the analysis.

▶ **Observation 7.** *Invoke Algorithm 1 on instance $\mathcal{F}$. Assume that $(\kappa_1, \ldots, \kappa_n)$ is the sequence of cores extracted after n iterations. Then $\sum_{i=1}^{n} \text{MINW}(\kappa_i) \leq \text{COST}(\mathcal{F})$. Furthermore, equality holds only if Algorithm 1 terminates after relaxing all of the cores.*

In other words, cores extracted by Algorithm 1 provide a lower bound on the optimal cost, and the algorithm terminates only when that lower bound equals the optimal cost.

Observation 7 forms the basis for a heuristic that is employed by most implementations of core-guided MaxSAT solvers that we are aware of. We say that Algorithm 1 uses bounds if it maintains a lower bound LB—computed using Observation 7—on the optimal cost of the instance $\mathcal{F}$ being solved. As soon as any solution τ is obtained, its cost $\text{COST}(\mathcal{F}, \tau)$ is compared to the lower bound. If $\text{LB} = \text{COST}(\mathcal{F}, \tau)$, the algorithm immediately terminates, without invoking Relax. In our setting, intermediate solutions are only obtained by OLL+WCE. In practice, implementations of OLL also obtain non-optimal solutions during search via the so-called stratification [2] heuristic.

WCE can allow Algorithm 1 using bounds to terminate without relaxing any cores. Consider an instance $\mathcal{F}^n$ with $\mathcal{B}(\mathcal{F}^n) = \{b_1, \ldots, b_{n+3}\}$ that contains the clauses $(b_i \vee b_{n+2})$ for $i = 1 \ldots n+1$ and $(b_{n+1} \vee b_{n+3})$. Assume that $w(b_i) = 1$ for $i = 1 \ldots (n+1), (n+3)$ and $w(b_{n+2}) = n+2$. This instance has one optimal solution τ of cost $\text{COST}(\mathcal{F}^n, \tau) = \text{COST}(\mathcal{F}^n) = n+1$ that sets $\tau(b_i) = 1$ for $i = 1 \ldots (n+1)$ and $\tau(b_{n+2}) = \tau(b_{n+3}) = 0$.

▶ **Proposition 8.** *Algorithm 1 extracts $n+1$ cores when computing an optimal solution to $\mathcal{F}^n$.*

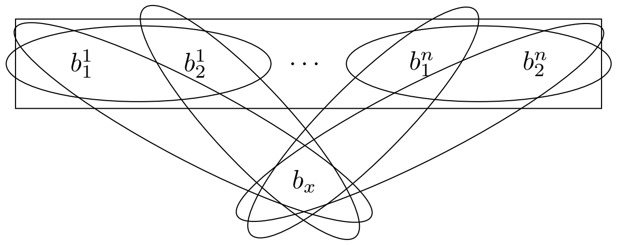

Figure 6 Structure of $\mathcal{F}^n$ in Observations 12 and 13. The ellipses and rectangles illustrate the clauses.

Proof. (Sketch.) The result follows from the fact that any core κ extracted by Algorithm 1 on $\mathcal{F}^n$ must have $\text{MINW}(\kappa) = 1$. This by Observation 7 implies that termination occurs only after $n+1$ cores have been extracted. ◀

▶ **Proposition 9.** *OLL+WCE using bounds can terminate without relaxing any cores when invoked on $\mathcal{F}^n$.*

Proof. Let the first core extracted be $\{b_{n+1}, b_{n+2}\}$. The weight of both b_{n+1} and b_{n+2} is then lowered by 1 before invoking the SAT solver again. In the second iteration, the set $\mathcal{B}(\mathcal{F}^1)$ contains $\{b_1, \ldots, b_n, b_{n+2}, b_{n+3}\}$. Assume that the algorithm continues in a similar manner, iteratively extracting a core of the form $\{b_i, b_{n+2}\}$ for each $i = 1 \ldots (n+1)$. Each extracted core κ will have $\text{MINW}(\kappa) = 1$, so at this point $\text{LB} = n+1$. In the next SAT solver call $\mathcal{B}(\mathcal{F}^1) = \{b_{n+2}, b_{n+3}\}$ so the solver is invoked assuming $b_{n+2} = b_{n+3} = 0$. The instance is satisfied by an assignment τ that sets $\tau(b_i) = 1$ for $i = 1 \ldots (n+1)$ and $\tau(b_{n+2}) = \tau(b_{n+3}) = 0$. This solution has $\text{COST}(\mathcal{F}^n, \tau) = n + 1 = \text{LB}$ so OLL+WCE terminates without invoking the function RELAX. ◀

We thereby arrive at the following.

▶ **Theorem 10.** *There is a family of MaxSAT instances $\mathcal{F}^n$ with $|\mathcal{B}(\mathcal{F}^n)| = \mathcal{O}(n)$ on which OLL using bounds is guaranteed to relax n cores, while OLL+WCE using bounds can terminate without relaxing any cores.*

In other words, there are instances on which the core relaxation steps of Algorithm 1 are redundant. In addition to (unnecessarily) increasing the size of the working instance, the redundant core relaxation steps can also increase the size of the MUSes of the working instance.

▶ **Observation 11.** *Invoke OLL on $\mathcal{F}^n$ and assume that the first core extracted is $\kappa = \{b_{n+1}, b_{n+2}\}$, (the same as in the proof of Proposition 9). After refining the set of blocking variables and relaxing κ, the next working instance $\mathcal{F}^2$ has $\mathcal{B}(\mathcal{F}^2) = \{b_1, \ldots, b_n, b_{n+2}, b_{n+3}, b_1^\kappa\}$, where b_1^κ is the new blocking variable introduced by RELAX. The set $\{b_1, b_{n+3}, b_1^\kappa\}$ is an MUS of $\mathcal{F}^2$ of size 3, i.e., larger than any of the MUSes of $\mathcal{F}^n$. Note that all MUSes of $\mathcal{F}^n$ are of size 2 and that Algorithm 1 only extracts MUSes of $\mathcal{F}^n$ in the proof of Proposition 9.*

So far we have shown that WCE can lower the number invocations of RELAX of Algorithm 1, thus decreasing the complexity of the working instance which alleviates a main bottleneck of core-guided MaxSAT solvers. However, we can also identify that WCE may also increase the number iterations. In the following, consider the instance $\mathcal{F}^n$ with $\text{HARD}(\mathcal{F}^n) = \{(b_1^i \lor b_2^i), (b_1^i \lor b_x), (b_2^i \lor b_x) \mid i = 1 \ldots n\} \cup \{(b_1^1 \lor b_2^1 \lor b_1^2 \lor b_2^2 \lor \cdots \lor b_1^n \lor b_2^n)\}$,

$\mathcal{B}(\mathcal{F}^n) = \{b_x\} \cup \{b_1^i, b_2^i \mid i = 1 \ldots n\}$ and $w(b_1^i) = n$, $w(b_2^i) = n + 1$ for $i = 1 \ldots n$ as well as $w(b_x) = n$. The optimal cost of the instance is $\mathrm{COST}(\mathcal{F}^n) = n^2 + n$ and an example of an optimal solution τ sets $\tau(b_1^1) = \tau(b_1^2) = \ldots = \tau(b_1^n) = \tau(b_x) = 1$ and $\tau(b_2^j) = 0$ for $j = 1 \ldots n$. The structure of $\mathcal{F}^n$ is shown in Figure 6.

▶ **Observation 12.** *Invoke OLL using bounds on $\mathcal{F}^n$. Assume that the first core extracted is $\kappa_1 = \{b_1^i, b_2^i \mid i = 1 \ldots n\}$ with $\textsc{minw}(\kappa_1) = n$. After the first invokation of* Relax*, the working instance $\mathcal{F}^2$ has $\mathcal{B}(\mathcal{F}^2) = \{b_2^i \mid i = 1 \ldots n\} \cup \{b_x\} \cup \{b_1^{\kappa_1}, \ldots, b_{2n-1}^{\kappa_1}\}$. In the next $n-1$ iterations, the algorithm can extract and relax the cores $\kappa_2 = \{b_1^{\kappa_1}\}, \ldots, \kappa_n = \{b_{n-1}^{\kappa_1}\}$ and finally the core $\kappa_{n+1} = \{b_n^{\kappa_1}, b_x\}$, all having $\textsc{minw}(\kappa_i) = n$. As $\sum_{i=1}^{n} \textsc{minw}(\kappa_i) = n^2 + n = COST(\mathcal{F}^n)$ the algorithm will terminate after relaxing all $n + 1$ of these cores.*

▶ **Observation 13.** *Invoke OLL+WCE using bounds on $\mathcal{F}^n$. Assume that the first core extracted is $\kappa_1 = \{b_1^i, b_2^i \mid i = 1 \ldots n\}$ with $\textsc{minw}(\kappa_1) = n$. After refining the blocking variables, the instance will have $\mathcal{B}(\mathcal{F}^1) = \{b_2^i \mid i = 1 \ldots n\} \cup \{b_x\}$ with $w(b_2^i) = 1$ for all $i = 1 \ldots n$ and $w(b_x) = n$. There are still n more cores of the form $\kappa_i = \{b_2^i, b_x\}$ with $\textsc{minw}(\kappa_i) = 1$ to extract before the* Relax *function is invoked. Notice that these cores can be extracted in any order. At this point, $n + 1$ cores $(\kappa_1, \ldots, \kappa_{n+1})$ have been extracted. Since $\sum_{i=1}^{n+1} \textsc{minw}(\kappa_i) = 2n < n^2 + n$ (for $n > 1$), the algorithm does not terminate on the next SAT solver call; in particular, the algorithm needs to extract more than $n + 1$ cores before terminating.*

Thereby we arrive at the following.

▶ **Theorem 14.** *For every $n \in \mathbb{N}$, there is a MaxSAT instance $\mathcal{F}^n$ with $|\mathcal{B}(\mathcal{F}^n)| = \mathcal{O}(n)$, and a core κ of $\mathcal{F}^n$ such that if κ is the first core extracted, then (i) OLL can terminate after extracting and relaxing in total $n + 1$ cores, while (ii) OLL+WCE has to extract at least $n + 2$ cores before terminating.*

7 Conclusions

The exact details of the transformations steps, which compile a newly extracted unsatisfiable core into the current working instance, are a key to efficient core-guided MaxSAT solving, and this is also where the various existing core-guided MaxSAT algorithms differ. We proposed a novel form of structure sharing that can be applied within the core extraction steps, aiming at speeding up runtimes of the core-guided approach as well as avoiding unnecessary introduction of additional variables and clauses to the working instances during iterations of the algorithms. In contrast to earlier approaches to lowering the number of introduced variables and clauses via incremental cardinality constraints, structure sharing is an intrinsically different approach. In particular, it builds on weight-aware core extraction, which allows for extracting multiple cores during a single iteration from weighted MaxSAT instances, and exploits shared substructures among cores for structure sharing. Putting structure sharing into practice in a state-of-the-art core-guided MaxSAT solver, we showed that structure sharing in combinations with WCE provides empirical runtime improvements. In addition to these main contributions, we also provided a theoretical analysis of the worst and best case impact of WCE on the central OLL MaxSAT algorithm.

References

1 Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in CLASP. In Agostino Dovier and Vítor Santos Costa, editors, *Technical*

Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.ICLP.2012.211.

2 Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving SAT-based weighted MaxSAT solvers. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2012. doi:10.1007/978-3-642-33558-7_9.

3 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 427–440. Springer, 2009. doi:10.1007/978-3-642-02777-2_39.

4 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013. doi:10.1016/j.artint.2013.01.002.

5 Carlos Ansótegui and Joel Gabàs. WPM3: an (in)complete algorithm for weighted partial MaxSAT. *Artificial Intelligence*, 250:37–57, 2017. doi:10.1016/j.artint.2017.05.003.

6 Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins, editors. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, volume B-2020-2 of *Department of Computer Science Report Series B*. Department of Computer Science, University of Helsinki, Finland, 2020.

7 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation 2018: New developments and detailed results. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):99–131, 2019. doi:10.3233/SAT190119.

8 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. *Maximum Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 24, pages 929–991. IOS Press BV, 2021. doi:10.3233/FAIA201008.

9 Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003. doi:10.1007/978-3-540-45193-8_8.

10 Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set MaxSat solving. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2020. doi:10.1007/978-3-030-51825-7_20.

11 Jeremias Berg and Matti Järvisalo. Weight-aware core extraction in SAT-based MaxSAT solving. In Christopher Beck, editor, *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 652–670. Springer, 2017. doi:10.1007/978-3-319-66158-2_42.

12 Nikolaj Bjørner and Nina Narodytska. Maximum satisfiability using cores and correction sets. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 246–252. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/041.

13 Jori Bomanson, Martin Gebser, and Tomi Janhunen. Improving the normalization of weight rules in answer set programs. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2014. doi:10.1007/978-3-319-11558-0_12.

14 Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003. `doi:10.1016/S1571-0661(05)82542-3`.

15 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006. `doi:10.1007/11814948_25`.

16 Federico Heras, António Morgado, and João Marques-Silva. Lower bounds and upper bounds for MaxSAT. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization - 6th International Conference, LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*, volume 7219 of *Lecture Notes in Computer Science*, pages 402–407. Springer, 2012. `doi:10.1007/978-3-642-34413-8_35`.

17 Alexey Ignatiev, António Morgado, and João Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018. `doi:10.1007/978-3-319-94144-8_26`.

18 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, 2019. `doi:10.3233/SAT190116`.

19 Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 613–631. IOS Press, 2009. `doi:10.3233/978-1-58603-929-5-613`.

20 Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 86–91. AAAI Press, 2006. URL: `http://www.aaai.org/Library/AAAI/2006/aaai06-014.php`.

21 Vasco Manquinho, João Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009. `doi:10.1007/978-3-642-02777-2_45`.

22 João Marques-Silva and Jordi Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, abs/0712.1097, 2007. URL: `http://arxiv.org/abs/0712.1097`, `arXiv:0712.1097`.

23 Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental cardinality constraints for MaxSAT. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 531–548. Springer, 2014. `doi:10.1007/978-3-319-10428-7_39`.

24 Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. On using incremental encodings in unsatisfiability-based MaxSAT solving. *Journal on Satisfiability, Boolean Modeling and Computation*, 9(1):59–81, 2014. `doi:10.3233/sat190102`.

25 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014. `doi:10.1007/978-3-319-09284-3_33`.

26 António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In Barry O'Sullivan, editor, *Principles and Practice of Constraint*

Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014. doi:10.1007/978-3-319-10428-7_41.

27 António Morgado, Federico Heras, Mark Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013. doi:10.1007/s10601-013-9146-2.

28 Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In Carla Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 2717–2723. AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8513.