Improving the Lower Bound in Branch-and-Bound Algorithms for MaxSAT

Shuolin Li[1], Chu-Min Li[1,2,*], Jordi Coll[3], Djamal Habet[1], Felip Manyà[4]

[1]Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
[2]Université de Picardie Jules Verne, Amiens, France
[3]Universitat de Girona, Girona, Spain
[4]Artificial Intelligence Research Institute, CSIC, Bellaterra, Spain
shuolin.li@lis-lab.fr, chu-min.li@u-picardie.fr, jordicoll@udg.edu, Djamal.Habet@univ-amu.fr, felip@iiia.csic.es

Abstract

The MaxSAT problem is an optimization version of the satisfiability problem (SAT). A tight lower bound (LB) on the number of falsified soft clauses in a MaxSAT solution is crucial for the efficiency of Branch-and-Bound (BnB) MaxSAT solvers. To compute an LB, modern BnB solvers detect disjoint inconsistent subsets of soft clauses, called *cores*, using unit propagation. A notable feature of these solvers is that soft clauses belonging to already detected cores cannot be reused to detect additional cores, limiting the number of cores that can be detected. In this paper, we propose an unlocking mechanism that allows the reuse of soft clauses in already detected cores while ensuring the soundness of LB. Experimental results show that this unlocking mechanism consistently improves the performance of a state-of-the-art BnB solver. In addition, it allowed us to win the first two places in the exact unweighted category of the MaxSAT Evaluation 2024.

Introduction

The MaxSAT problem is an optimization version of the satisfiability problem (SAT), which is known to be NP-hard (Papadimitriou 1994). Given a Conjunctive Normal Form (CNF) formula, generally represented as a set of clauses, solving MaxSAT involves finding an assignment to all Boolean variables that maximizes (minimizes) the number of satisfied (falsified) clauses. MaxSAT has three variants: partial MaxSAT, in which some clauses, called hard clauses, cannot be falsified, and the others, called soft clauses, can be falsified; weighted MaxSAT, in which each clause can be falsified and has a positive integer weight representing the penalty of its violation; and weighted partial MaxSAT, which has hard clauses and weighted soft clauses. In weighted partial MaxSAT, an optimal solution satisfies all the hard clauses and minimizes the sum of weights of the falsified soft clauses. Thanks to the efforts of the scientific community, MaxSAT has become a competitive generic approach to solve combinatorial optimization problems such as scheduling (Demirović, Musliu, and Winter 2019), maximum clique (Li and Quan 2010), FPGA routing (Fu and Malik 2006), and protein design (Allouche et al. 2014).

As with other NP-hard problems, algorithms for MaxSAT can be divided into two categories: heuristic and exact algorithms. The former quickly find good solutions without guaranteeing optimality, see, e.g., (Lei and Cai 2018; Chu, Cai, and Luo 2023b; Zheng et al. 2022). The latter return a proven optimal solution. There are also hybrid algorithms that combine heuristic and exact algorithms, see e.g., (Chu, Cai, and Luo 2023a; Nadel 2023; Zheng et al. 2023; Lei et al. 2021). In this paper, we focus on exact algorithms.

Exact algorithms for MaxSAT can be broadly categorized into two groups. The first group refers to SAT-based MaxSAT solvers (Bacchus, Järvisalo, and Ruben 2021). They transform MaxSAT into a series of SAT problems, which are then solved using a SAT solver. Representative solvers in this category include WPM1-3 (Ansótegui, Bonet, and Levy 2009; Ansótegui, Bonet, and Levy 2010; Ansótegui and Gabàs 2017), QMaxSAT (Koshimura et al. 2012), Open-WBO (Martins, Manquinho, and Lynce 2014; Martins et al. 2014), CGSS (Berg et al. 2023), Pacose (Paxian and Becker 2020), MaxHS (Bacchus 2020; Narodytska and Bacchus 2014), and EvalMaxSAT (Avellaneda 2020). The second group refers to branch-and-bound (BnB) MaxSAT solvers (Li and Manyà 2021). They find an optimal assignment by constructing a binary search tree in which each node corresponds to a partial assignment. A BnB solver calculates a lower bound (*LB*) on the number of falsified soft clauses at each node and compares it with the current upper bound (*UB*) calculated from the best solution found so far. If LB $\geq$ UB, the current branch cannot yield a better solution, leading the solver to prune and backtrack; otherwise, the search continues. Consequently, having a tight LB is crucial for solving MaxSAT efficiently. State-of-the-art BnB solvers include MaxSatz (Li, Manyà, and Planes 2007), MiniMaxSat (Heras, Larrosa, and Oliveras 2008), Ahmaxsat (Abramé and Habet 2014; Cherif, Habet, and Abramé 2020), Akmaxsat (Kuegel 2010), and (W)MaxCDCL (Li et al. 2021a,b).

BnB MaxSAT solvers usually compute an LB by detecting disjoint local cores using unit propagation. A local core is a subset of soft clauses that cannot all be satisfied without falsifying a hard clause at the current search tree node. A significant limitation of current BnB MaxSAT solvers is their inability to reuse soft clauses from previously identified cores when detecting new cores. This constraint is im-

*Corresponding author

posed to maintain the disjointness of the cores but severely restricts the number of cores that can ultimately be detected. In this paper, we propose an unlocking mechanism that allows the reuse of soft clauses from previously detected cores to identify additional cores, while still ensuring the soundness of LB. Intuitively, the soft clauses in the previously detected cores are strong constraints and have high probability to form new cores with other soft clauses. Therefore, reusing these soft clauses with the proposed unlocking mechanism could lead to the detection of additional cores.

We integrated the proposed unlocking mechanism into WMaxCDCL and MaxCDCL, two state-of-the-art BnB MaxSAT solvers. Experimental results on benchmarks from the MaxSAT Evaluations 2019-2023 demonstrate that the unlocking mechanism consistently enhances the performance of both WMaxCDCL and MaxCDCL. In addition, we submitted the two solvers equipped with the unlocking mechanism to the MaxSAT Evaluation 2024[1]. Using OpenWBO as a preprocessing during 1200s and 300s respectively, WMaxCDCL and MaxCDCL won the first two places of the exact unweighted category.

This paper is structured as follows. We first present some preliminaries. Next, we describe the unlocking mechanism separately for both unweighted and weighted MaxSAT. Then, we present experimental results to show the performance and some properties of the mechanism. Finally, we conclude and suggest some future work.

Preliminaries

Given a set of propositional variables, a *literal* is a variable x or its negation $\neg x$. A clause cl is a disjunction of literals, and it is *unit* if it contains exactly one literal. A CNF (Conjunctive Normal Form) formula ϕ=$\{cl_1, ..., cl_m\}$ is a set of clauses representing their conjunction. An *assignment* α assigns either 0 or 1 to each propositional variable, and satisfies a literal x (resp. $\neg x$) if $\alpha(x)$=1 (resp. $\alpha(x)$=0), satisfies a clause if it satisfies at least one of its literals, and satisfies a formula if it satisfies all its clauses. If all variables are assigned a value, α is *complete*; otherwise it is *partial*. When a literal l is satisfied (resp. falsified), we say also it is assigned 1 (resp. 0), and denote α as a (sub)set of literals assigned 1. When α is partial, formula $\phi|_\alpha$ is ϕ simplified under α, by removing all satisfied clauses and deleting falsified literals from other clauses. An assignment α is said to be feasible if no hard clause is empty in $\phi|_\alpha$. A feasible complete assignment is also called a solution of ϕ. In the sequel, we assume that all assignments under consideration are feasible.

A weighted partial MaxSAT instance ϕ can be divided into two subsets: the set of *hard clauses* H and the set of *soft clauses* S, where each clause s in S has associated an integer *weight* $w(s)$. An optimal *solution* of ϕ is a complete assignment that satisfies all clauses in H and minimizes the sum of weights of falsified soft clauses in S. A partial MaxSAT instance is a particular case in which the weight of all soft clauses is identical and thus can be represented by 1.

Given a weighted partial MaxSAT instance ϕ, a BnB MaxSAT solver constructs a binary search tree to find an optimal assignment. At each node, the solver selects a branching variable x and develops two branches corresponding to $\alpha(x)$=0 and $\alpha(x)$=1, respectively. Thus, each node corresponds to a partial assignment α including the variables assigned in the path from the tree root to the current node. The formula in the current node becomes $\phi|_\alpha$. To speed up the search, the solver computes LB $=falsifiedWeight + estimatedWeight$ before branching, where $falsifiedWeight$ is the sum of weights of the soft clauses falsified under α, and $estimatedWeight$ is an underestimation of the sum of weights of the soft clauses that will be falsified if α is extended to a complete assignment. The function to underestimate $estimatedWeight$ is commonly referred to as *lookahead*. Let UB denote the total weight of falsified soft clauses in the best solution found so far. If LB$\geq$UB, no better solution can be found in the subtree rooted at the current node, which is then pruned.

Therefore, a crucial factor in the performance of a BnB solver is the accurate computation of $estimatedWeight$, which is typically achieved by detecting disjoint inconsistent subsets of soft clauses under α, called local *cores* w.r.t. α. As this paper exclusively considers local cores, we will omit the word "local" in the sequel. Each core c is associated with a weight $w(c)$, defined as the minimum total weight of its soft clauses that are falsified across all possible extensions of α. In existing BnB MaxSAT solvers, the core weight is always 1 for unweighted MaxSAT, while for weighted MaxSAT, it is the minimum weight among its soft clauses. Since the cores are disjoint, $estimatedWeight$ is the sum of the weights of these cores.

Different methods for detecting disjoint cores have been described in the literature. The first method, proposed by Wallace and Freuder (1993), involves counting the number of pairs of complementary unit soft clauses, $\{x_i\}$ and $\{\neg x_i\}$, where each pair constitutes a core. Shen and Zhang (2004) improved this approach by using linear resolution to generate additional unit clauses before counting. Alsinet et al. (2004) introduced the use of the star rule to detect sets of clauses of the form $\{\{x_1\}, ..., \{x_k\}, \{\neg x_1, ..., \neg x_k\}\}$, with each set representing a core.

MaxSatz (Li, Manyà, and Planes 2005, 2006) was the first BnB MaxSAT solver to use unit propagation (UP) to detect cores. UP is the process of repeatedly propagating a unit clause l, i.e., satisfying l (and falsifying $\neg l$), removing all clauses containing l and deleting $\neg l$ from the remaining clauses until there is no unit clause or an empty clause cl is produced (all literals of cl are falsified and deleted). When an empty clause cl is produced, an implication graph can be constructed from cl. The detected core is the set of the soft clauses in a path to cl in the implication graph. Figure 1 shows an example of UP-based core detection.

Current unweighted MaxSAT BnB solvers lock the soft clauses in the already detected cores, so that they cannot be used to detect additional cores, ensuring the disjointness of the cores. In weighted MaxSAT, these solvers lock part of the weight of the soft clauses instead of the soft clauses themselves. For example, let c=$\{cl_1, cl_2, cl_3\}$ be a detected core, and let $w(cl_1)$=2, $w(cl_2)$=3, $w(cl_3)$=6. Then, $w(c)$=2. The weight of cl_2 (cl_3) is split to lock its

[1]https://maxsat-evaluations.github.io/2024/

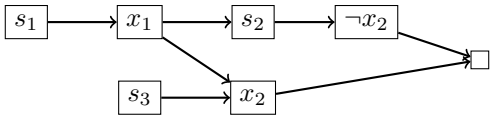

Figure 1: An example of an implication graph to identify a core. The formula ϕ contains 3 soft clauses: $\{s_1\}$, $\{s_2\}$, $\{s_3\}$, and 4 hard clauses: $\{\neg s_1, x_1\}, \{\neg x_1, s_2\}, \{\neg s_2, \neg x_2\}$ and $\{\neg s_3, \neg x_1, x_2\}$. Each node represents a satisfied literal l. The incoming edges represents the hard clause that allows to propagate l. For example, the node x_2 represents the clause $\{\neg s_3, \neg x_1, x_2\}$. Propagating $\{s_1\}$ satisfies x_1, s_2 and $\neg x_2$, while propagating $\{s_3\}$ satisfies x_2, making $\{\neg s_2, \neg x_2\}$ or $\{\neg s_3, \neg x_1, x_2\}$ empty. By retracing the implication graph from the empty clause to the propagated soft clauses, we know that $\{s_1\}$ and $\{s_3\}$ cannot be satisfied at the same time. So, the core $\{\{s_1\}, \{s_3\}\}$ is identified.

weight 2, but its remaining weight 1 (4) can be used to detect additional cores. The detected cores are considered disjoint in a broader sense, because cl_2 (cl_3) can be considered as two distinct copies, one with weight 2 that is locked into c to decide $w(c)$, and the other with weight 1 (4), referred to as the remaining weight and denoted by $rw(cl_2)$ ($rw(cl_3)$) in the general case, which can be used to detect new cores.

We use $wlock(cl, c)$ to denote the weight of cl locked in core c, which is the minimum weight locked into c to ensure $w(c)$ and cannot be used to detect other cores in current BnB solvers. In the above example, $wlock(cl_1, c) = wlock(cl_2, c) = wlock(cl_3, c) = 2$ ensures that $w(c) = 2$. If for any $cl \in \{cl_1, cl_2, cl_3\}$, $wlock(cl, c) < 2$, then $w(c) < 2$.

We next present an unlocking mechanism to reuse the locked soft clauses in unweighted MaxSAT or $wlock(cl, c)$ in weighted MaxSAT to detect other cores. These locked clauses or weights are strong constraints and have a high probability of forming other cores with other soft clauses, as suggested by the fact that they are already part of a core. Their reuse presumably could allow to detect more cores.

Lookahead with Unlocking Mechanism for Unweighted MaxSAT

As in MaxCDCL, we replace each soft clause cl by a soft literal s and add hard clauses encoding $s \leftrightarrow cl$. With this transformation, solving (any variant of) MaxSAT is equivalent to minimizing a (pseudo-Boolean) cost function over soft literals under hard constraints expressed as a CNF formula. This allows us to deal with cores of soft literals instead of cores of soft clauses. For any core c, $w(c)$ represents the minimum cost of c. In unweighted MaxSAT, $w(c)$ indicates that c contains at least $w(c)$ falsified soft literals for any complete extension of the partial assignment α. A core c is *locked* if $w(c) > 0$. An unassigned soft literal is *locked* if it belongs to a locked core. Our *unlocking mechanism* consists in unlocking the soft literals in c when there are $w(c)$ falsified soft literals in c, allowing its unassigned soft literals to be reused for detecting other cores.

Alg. 1 depicts the lookahead process for detecting a set of cores using the unlocking mechanism, given a set of hard clauses H and a set of unassigned soft literals S. It propagates the soft literals in S one by one as assumptions until an empty hard clause cl is produced or a literal s' in S is falsified. Propagating a literal s amounts to adding the unit clause s in H and performing unit propagation. Then, the algorithm retraces the implication graph by collecting all propagated assumptions starting a path to cl or s' to form a core d. The main difference with MaxCDCL is the unlocking operations from lines 6 to 11, which reinsert into S soft literals previously locked in a core c, but unlocked because $w(c)$ literals in c were falsified during UP ($w(c)$ becomes 0). Since $w(c)$ changes during UP, we save its original value in $ow(c)$ to restore $w(c)$ after detecting a core. Note that the propagated assumptions contributing to unlocking the cores in C must also be added to d (line 17).

Algorithm 1: Lookahead(H, S, α, UB), the algorithm to compute LB

Input: H, a set of hard clauses; S, a set of unassigned soft literals; α, a partial assignment; UB, a given upper bound
Output: LB, a lower bound on the number of falsified soft literals for any extension of α.

```
1: LB ← number of soft literals falsified by α
2: cores ← ∅ //will contain identified cores
3: H' ← H //save the hard clauses
4: while S is not empty and LB < UB do
5:    Pick s ∈ S and propagate s
6:    for each soft literal s' belonging to a core c ∈ cores
      and falsified at line 5 such that w(c) > 0 do
7:       w(c) ← w(c) − 1;
8:       if w(c)=0 then
9:          add all unassigned literals of c into S
10:      end if
11:   end for
12:   if an empty hard clause cl is produced or an unassigned soft literal s' in S is falsified then
13:      for each core c ∈ cores do w(c) ← ow(c)
14:      d ←{soft literals propagated at line 5 starting a
         path to cl or s' in the implication graph G}
15:      add s' to d if cl is not produced
16:      C ← {c ∈ cores|∃s ∈ d such that s ∈ c}
17:      d ← d∪{soft literals propagated at line 5 starting a
         path to a falsified literal in a core c ∈ C}
18:      d ← d ∪ ⋃_{c∈C} c
19:      ow(d) ← w(d) ← 1+ Σ_{c∈C} ow(c)
20:      LB++
21:      cores ← (cores ∪ {d}) \ C
22:      H ← H' //restore the hard clauses
23:      remove all locked literals from S
24:   end if
25: end while
26: return LB
```

Example 1. *Let $c_1 = \{s_1, s_2, s_3\}$ be a core already detected, and let $ow(c_1) = w(c_1) = 1$. Its soft literals s_1, s_2 and s_3 are locked. Alg. 1 now propagates s_4 as an assumption, falsifying s_1, so that $w(c_1)$ is decreased to 0, and s_2 and s_3 are unlocked. It then propagates s_2 as an assumption,*

which falsifies s_5, an unassigned soft literal in S. So, a new core c_2=$\{s_4, s_2, s_5\}$ is identified, in which s_2 belongs to c_1. So, c_2 is updated to the union with c_1 and becomes $\{s_1, s_2, s_3, s_4, s_5\}$ at line 18, and $w(c_2)$ and $ow(c_2)$ are set to 1+$ow(c_1)$=2 at line 19 to lock all its literals. Note that c_1 should be removed after finalizing c_2.

After detecting a new core c_3=$\{s_6, s_7\}$ with $w(c_3)$=1, Alg. 1 propagates s_8 as an assumption, which falsifies s_5 and s_6, unlocking s_7 from c_3, and $w(c_2)$ is decreased to 1. Propagating s_7 as an assumption falsifies s_4, unlocking s_1, s_2 and s_3 from c_2 because $w(c_2)$ is 0 now. Continuing UP, s_3 is falsified, giving a new core c_4=$\{s_8, s_7, s_3\}$, which is updated at line 18 to the union with c_2 and c_3. At line 19, $w(c_4)$ and $ow(c_4)$ are set to 1+$ow(c_3)$+$ow(c_2)$=4. At line 21, c_2 and c_3 are removed.

If no additional cores can be detected, Alg. 1 will return 4, which is the sum of the weights of all remaining cores.

Note that without unlocking, s_2 could not be propagated as an assumption, and c_2=$\{s_4, s_2, s_5\}$ would not be detected because s_2 remains locked in c_1. Similarly, c_4=$\{s_8, s_7, s_3\}$ would not be detected because s_3 is locked in c_1. Consequently, LB would be 2 without unlocking.

Proposition 1 establishes the soundness of Alg. 1.

Proposition 1. *Any complete assignment extended from α falsifies at least $ow(d)$ soft literals in the core d identified at line 18 of Alg. 1. After removing cores in C at line 21, d is disjoint with the remaining cores, and the returned LB is equal to $\sum_{c \in cores} ow(c)$.*

Proof. By induction. When $ow(d)$=1, by construction, d does not contain any soft literal of other cores, and at least one soft literal must be falsified by any complete assignment.

Assume the proposition holds for $ow(d)$<k. When $ow(d)$=k, for any c in C obtained at line 16, we have $ow(c) < k$. So, these cores c are pairwise disjoint, and an assignment falsifies at least $\sum_{c \in C} ow(c)$ soft literals. Let d_0 denote the subset of soft literals of d that do not belong to any core in C. If an assignment falsifies a soft literal in d_0, it falsifies at least $1 + \sum_{c \in C} ow(c)$ soft literals in d, because $d_0 \cap c$=$\emptyset$ for any $c \in C$. If an assignment satisfies all soft literals in d_0, note that if a literal in $c \in C$ is in S, it is because at least $ow(c)$ literals in c were already falsified, so that it was added into S at lines 8 and 9. There are two cases under the condition that all literals in d_0 are satisfied: (1) A hard empty clause cl is produced, or (2) $s' \in S$ is falsified. In case (1), at least one literal $s'' \in S$ propagated at line 5 must be falsified in a feasible extension of α. So, s'' must be in a core $c \in C$ with already $ow(c)$ falsified literals. In case (2), $s' \in S$ also must be in a core $c \in C$ with already $ow(c)$ falsified literals. In both cases, UP falsifies at least $ow(c)$ literals in each c in C, except the core c containing s'' or s' in which UP falsifies $ow(c) + 1$ soft literals.

Note that d is disjoint with cores not in C and cores in C are removed after finalizing d.

Finally, each new core increases LB by 1 at line 20, because $\sum_{c \in C} ow(c)$ was already counted in LB. So, the final LB is equal to $\sum_{c \in cores} ow(c)$. □

Lookahead with Unlocking Mechanism for Weighted MaxSAT

Unfortunately, the extension of the unlocking mechanism to weighted MaxSAT is not trivial, and a naive extension of Alg. 1 could result in a less accurate LB, because of the different weights of soft literals.

Example 2. *Let s_1, s_2, s_3, s_4 and s_5 be soft literals with weights 3, 3, 3, 1 and 1, respectively. Using Alg. 1, we first identify a core c_1=$\{s_1, s_2, s_3\}$, and $w(c_1)$=3, giving LB=3. Next, we propagate s_4, which falsifies s_1, and unlocks s_2 and s_3 from c_1. We then propagate s_5, which falsifies s_3. We thus obtain a new core c_2=$\{s_1, s_2, s_3, s_4, s_5\}$ by removing c_1. By construction, there are at least two falsified literals in c_2. However, since s_4 and s_5 have the smallest weight 1, we have $w(c_2) = 2$. So, the trivial application of Alg. 1 gives a smaller LB=2.*

In a more complex situation, we cannot even guarantee the number of falsified soft literals in weighted MaxSAT as we can in unweighted MaxSAT, because a soft literal can belong to multiple cores in weighted MaxSAT, differently from the case of unweighted MaxSAT, in which a soft literal belongs to at most one core. So, the beneficial property established by Proposition 1 for unweighted MaxSAT does not hold for weighted MaxSAT.

Example 3. *Let s_1, s_2, s_3, s_4, s_5, s_6 and s_7 be soft literals with weights 1, 1, 4, 3, 5, 6 and 7, respectively. Using Alg. 1, we first identify a core c_1=$\{s_1, s_2, s_3\}$ and $w(c_1)$=1. The remaining weight of s_3, $rw(s_3)$=3. Next, we identify another core c_2=$\{s_3, s_4, s_5\}$ and $w(c_2)$=3, resulting in an LB of 4. Subsequently, we propagate s_6, which falsifies s_3 and unlocks soft literals in c_1 and c_2. This allows us to propagate s_1 and s_4, which falsifies s_7. Finally, we obtain c_3=$\{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$ and remove c_1 and c_2 using a naive extension of Alg. 1, because we have propagated s_1 in c_1 and s_4 in c_2 to obtain c_3. The problem is that s_3 belongs to both c_1 and c_2, so that we cannot guarantee three falsified soft literals in c_3, because the construction of c_3 guarantees one falsified soft literal among the literals of c_1 and one among those of c_2, respectively, together with another falsified literal in c_3, but the falsified soft literal in c_1 and c_2 can both be s_3. Therefore, we can only guarantee two different falsified soft literals in c_3. Since s_1 and s_2 have the smallest weight of 1 in c_3, this results in an LB of 2, instead of the previous LB of 4 before detecting c_3.*

We have identified the main difficulty to extend the unlocking mechanism to weighted MaxSAT, which lies in how to compute the weight $w(c)$ of a newly detected core c, the part of weight of a soft literal s locked in c to decide $w(c)$, denoted as $wlock(s, c)$, and its remaining weight $rw(s)$ not locked in any core, when c includes soft literals already locked in previous cores. Example 4 and Example 5 suggest our approach to compute $w(c)$, $wlock(s, c)$ and $rw(s)$, based on the following unlocking mechanism: an unassigned soft literal s such that $rw(s)$>0 can be freely propagated as an assumption to detect a new core. If $rw(s)$=0, s is locked, but if $\sum_{s' \in c, s' \text{ assigned } 0} wlock(s', c)$ reaches $w(c)$ during UP, then s is unlocked with available weight $wlock(s, c)$,

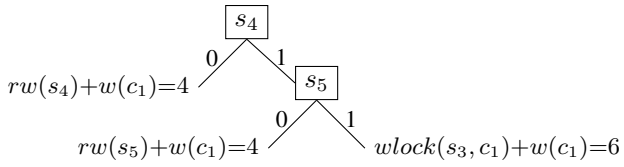

Figure 2: Illustration of Example 4.

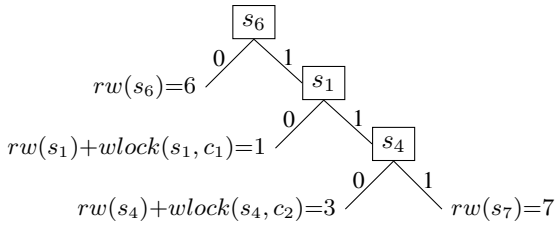

Figure 3: Illustration of Example 5. Only additional falsified weight is showed in each case.

and can also be freely propagated as an assumption. In this case, we say that c is *unlocked* and c *unlocks* s. For convenience, $wlock(s,c)$=0 if s is not in c. Recall that α denotes the current partial assignment.

Example 4. *Refer to Example 2, where the weights of s_1 to s_5 are 3, 3, 3, 1, 1, respectively. After detecting c_1=$\{s_1,s_2,s_3\}$, $w(c_1)$=3, $wlock(s_1,c_1)$=$wlock(s_2,c_1)$=$wlock(s_3,c_1)$=3, $rw(s_1)$=$rw(s_2)$=$rw(s_3)$=0, $rw(s_4)$=$rw(s_5)$=1. Note that s_1, s_2 and s_3 are locked.*

As in Example 2, propagating s_4 falsifies s_1, and unlocks s_2 and s_3 from c_1. Then, propagating s_5 falsifies s_3, giving a new core c_2=$\{s_1,s_2,s_3,s_4,s_5\}$ after removing c_1. We focus on the two propagated soft literals s_4 and s_5 in the ordering of their propagation to compute $w(c_2)$, $wlock(s,c_2)$ and $rw(s)$ for each soft literal s.

In fact, any assignment extending α falls into one of the following three disjoint cases: (1) s_4=0, (2) s_4=1 and s_5=0, (3) s_4=s_5=1 and s_3=0. Figure 2 illustrates the three disjoint cases from left to right. In case (1), the total guaranteed falsified weight of soft literals is $rw(s_4)+w(c_1)$=1+3=4, because s_4 is falsified. In case (2), it is $rw(s_5)+w(c_1)$=3+1=4, because s_5 is falsified. In case (3), it is $wlock(s_3,c_1)+w(c_1)$=3+3=6, because the propagation of s_4 and s_5 can be done, and the falsified weight in c_1 is already $w(c_1)$ by falsifying s_1, allowing to unlock s_3 and its weight $wlock(s_3,c_1)$=3 locked in c_1 before s_3 is falsified. Thus, $w(c_2)$ is 4, the minimum total guaranteed falsified weight among the three cases, giving a better LB=4. Note that $rw(s_i)$ is not involved in the computation of $w(c_2)$ for i=1, 2 or 3. So, for i=1, 2 or 3, $rw(s_i)$ is not changed, and $wlock(s_i,c_2)$ is equal to $wlock(s_i,c_1)$. But $rw(s_4)$ and $rw(s_5)$ contribute to $w(c_2)$ by 1. Thus, $wlock(s_4,c_2)$=$wlock(s_5,c_2)$=1 and $rw(s_4)$=$rw(s_5)$=0 after computing $w(c_2)$.

Example 5. *Refer to Example 3. We have 7 soft literals s_1 to s_7 with weights 1, 1, 4, 3, 5, 6 and 7, respectively. Once c_1=$\{s_1,s_2,s_3\}$ and c_2=$\{s_3,s_4,s_5\}$ are detected, $w(c_1)$=1, $w(c_2)$=3, $wlock(s_1,c_1)$=$wlock(s_2,c_1)$=$wlock(s_3,c_1)$=1, $wlock(s_3,c_2)$=$wlock(s_4,c_2)$=$wlock(s_5,c_2)$=3, $rw(s_1)$=$rw(s_2)$=$rw(s_3)$=$rw(s_4)$=0, $rw(s_5)$=2, $rw(s_6)$=6, and $rw(s_7)$=7.*

Propagating s_6 as an assumption falsifies s_3 and unlocks c_1 and c_2, because $wlock(s_3,c_1)$=$w(c_1)$=1 and $wlock(s_3,c_2)$=$w(c_2)$=3. Then, s_1∈c_1 and s_4∈c_2 can be propagated as assumption, falsifying s_7 and giving a new core c_3. Since s_6, s_1 and s_4 are propagated assumptions which finally falsify s_7, and s_1 and s_4 are from c_1 and c_2 respectively, we have c_3=$\{s_6,s_1,s_4,s_7\}$ ∪ c_1 ∪ c_2. We now focus on s_6, s_1, s_4 in the ordering of their propagation as assumption to compute $w(c_3)$. Indeed, any complete assignment extending α falls into one of the following four disjoint cases: (1) s_6=0, (2) s_6=1 and s_1=0, (3) s_6=s_1=1 and s_4=0, and (4) s_6=s_1=s_4=1 and s_7=0, as illustrated in Figure 3. We will set $w(c_3)$ to the minimum total guaranteed falsified weight among the four cases.

In case (1), the total guaranteed falsified weight is $rw(s_6)+w(c_1)+w(c_2)$, because s_6 is falsified. In cases (2), (3) and (4), c_1 and c_2 are unlocked, meaning that the falsified weight is already $w(c_1)+w(c_2)$ because of the falsification of s_3, independently of s_1 and s_4. The additional falsified weight in cases (2), (3) and (4) is $rw(s_1)+wlock(s_1,c_1)$=1, $rw(s_4)+wlock(s_4,c_2)$=3 and $rw(s_7)$=7, respectively. Consequently, $w(c_3)$ is set to $w(c_1)+w(c_2)+min(6,1,3,7)$=5. Note that the basic guaranteed falsified weight is $w(c_1)+w(c_2)$ in all cases. Figure 3 only shows the additional guaranteed falsified weight in each case because of space limitation.

Now we compute $wlock(s,c_3)$ for each s∈c_3, which is the minimum weight of s contributing to compute $w(c_3)$=5. First, s_2, s_3 and s_5 do not participate in the computation of $w(c_3)$ apart from that they belong to c_1 or c_2. So, $wlock(s,c_3)$=$wlock(s,c_1)$+$wlock(s,c_2)$ for s ∈ $\{s_2,s_3,s_5\}$. For s∈$\{s_6,s_1,s_4,s_7\}$, $wlock(s,c_3)$ must not be less than the weight already locked in c_1 and c_2, and not less than $min(6,1,3,7)$, the increase of $w(c_3)$ w.r.t. $w(c_1)+w(c_2)$. For example, to keep $w(c_3)$=5 unchanged, the contribution of s_6 to c_3 must not be less than 1. Formally, $wlock(s,c_3)$=$max(wlock(s,c_1)+wlock(s,c_2), min(6,1,3,7))$. So, $wlock(s_6,c_3)$=1, $wlock(s_1,c_3)$=1, $wlock(s_4,c_3)$=3, $wlock(s_7,c_3)$=1. It is under these conditions that we have $w(c_3)$=5.

Finally, we compute the remaining weight $rw(s)$ for each s∈c_3. Again, $rw(s)$ does not change for s∈$\{s_2,s_3,s_5\}$. But $rw(s_6)$ and $rw(s_7)$ should be reduced by their weight newly locked in c_3. So, $rw(s_6)$=5 and $rw(s_7)$=6, and $rw(s_1)$ and $rw(s_4)$ remain 0, because they contribute to c_3 by their weight locked in c_1 or c_2.

Note that the weight of s_5, s_6 or s_7 locked in c_3 is substantially smaller than $w(c_3)$, leaving their remaining weight to detect other cores.

We now introduce Alg. 2, which, given a set of hard clauses H, a set of weighted soft literals (S,w) and a partial assignment α, detects cores and computes their weight as illustrated in Examples 4 and 5. The main *while* loop can be divided into two parts. The first part (line 5 to line 15)

Algorithm 2: WLookahead(H, (S, w), α, UB), the algorithm to compute LB for weighted MaxSAT

Input: H, a set of hard clauses; (S, w), a set of unassigned free soft literals with their weight function w; α, a partial assignment; UB, a given upper bound

Output: LB, a lower bound of total weight of falsified soft literals for any extension of α.

Notation: $c \prec s$ means c is unlocked before s is propagated or s is the last falsified soft literal *fs*

1: LB $\leftarrow$ total weight of soft literals falsified by α
2: $cores \leftarrow \emptyset$; $H' \leftarrow H$ `//save the hard clauses`
3: *fs* $\leftarrow$ NULL; `//a falsified soft literal`
4: **while** S is not empty and LB<UB **do**
5: Pick literal $s \in S$ and propagate s as assumption
6: **if** a soft literal s' is falsified such that $rw(s')>0$ or s' is in a core c with $wlock(s', c)>w(c)$ **then**
7: *fs* $\leftarrow s'$
8: **else**
9: **for** each falsified soft literal s' in a core c such that $rw(s')=0$ and $w(c)>0$ and $wlock(s', c)\leq w(c)$ **do**
10: $w(c) \leftarrow w(c) - wlock(s', c)$;
11: **if** $w(c)=0$ **then**
12: add $\{u \mid u \in c$ and u is unassigned and $rw(u)=0\}$ into S
13: **end if**
14: **end for**
15: **end if**
16: **if** an empty hard clause cl is produced or *fs* is not NULL **then**
17: $d \leftarrow$ {the propagated assumptions starting a path to cl or *fs* in the implication graph G}
18: add *fs* into d if cl is not produced
19: $C \leftarrow \{c \in cores \mid wlock(fs, c)>w(c)$ or c unlocks $s \in d\}$
20: $d \leftarrow d \cup$ {soft literals propagated at line 5 starting a path to a falsified literal in a core $c \in C$}
21: **for** each s in d **do**
22: $aw(s) \leftarrow rw(s) + \sum_{c \in C, c \prec s} (wlock(s, c) - w(c))$
23: **end for**
24: *minaw* $\leftarrow min_{s \in d} aw(s)$
25: **for** each s in d **do**
26: $wlock(s, d) \leftarrow max(\sum_{c \in C} wlock(s, c)$, *minaw*$)$
27: $rw(s) \leftarrow max(0, aw(s) - wlock(s, d))$
28: **end for**
29: $w(c) \leftarrow ow(c)$ for each core c in $cores$
30: $ow(d) \leftarrow w(d) \leftarrow$ *minaw* $+ \sum_{c \in C} ow(c)$
31: $d \leftarrow d \cup \bigcup_{c \in C} c$
32: LB $\leftarrow$ LB + *minaw*
33: $cores \leftarrow (cores \cup \{d\}) \setminus C$
34: remove all locked literals from S
35: restore hard clauses and set *fs* to NULL
36: **end if**
37: **end while**
38: return LB

repeatedly picks and propagates a soft literal from S. The distinguishing feature of this part is the unlocking of cores and their locked soft literals u (i.e., u with $rw(u)=0$) when the weight of these cores is decreased to 0 (lines 9 to 14). Note that the cases $rw(s')>0$ or $wlock(s', c)>w(c)$ indicate the detection of a new core, and are treated at line 7. This occurs because the total falsified weight during UP is greater than the total weight of the unlocked cores. Similar to Alg. 1, since $w(c)$ changes during UP, we save its original value computed at line 30 in $ow(c)$.

The second part (lines 16-36) is executed when a new core is detected. The propagated assumptions starting a path to the empty hard clause cl or the falsified literals *fs* in the implication graph G are collected into d, where *fs* is also added, and the cores that unlocked a propagated assumption in d are collected in C. In addition, those propagated assumptions contributing to unlock cores in C are also added into d (lines 16-20). Then, $w(d)$, $wlock(s, d)$ and $rw(s)$ for each soft literal $s \in d$ are computed (lines 21-30).

Let $s_1, \ldots, s_k$ be assumptions in d propagated in this ordering. The assignments extended from α can be partitioned into the following $k+1$ disjoint cases: (1) $s_1=0$; (2) $\cdots$ (k) $s_1=\cdots=s_{i-1}=1, s_i=0$ for $i=2, \ldots, k$; (k+1) $s_1=\cdots=s_k=1$. Denoting *fs* by s_{k+1}, we show below that the total guaranteed falsified weight in case i ($1 \leq i \leq k+1$) can be written as $rw(s_i) + \sum_{c \in C, c \prec s_i} (wlock(s_i, c) - w(c)) + \sum_{c \in C} ow(c)$, where $c \prec s_i$ means that c is unlocked before s_i is assigned a value. We will set $w(d)$ to the minimum of them. Recall $wlock(s_i, c)=0$ for $s_i \notin c$.

In case (1), no core is unlocked. So, the total guaranteed falsified weight is in fact $rw(s_1) + \sum_{c \in C} ow(c)$, as in the leftmost branch of Figures 2 and 3.

In cases (2) $\cdots$ (k), some cores in C could be unlocked because the propagation of s_1, ..., s_{i-1} could falsify some locked literals other than s_i in these cores, giving falsified weight $\sum_{c \in C, c \prec s_i} ow(c)$. Since the unlocking of these cores is independent of s_i, the weight of s_i locked in these cores c ($wlock(s_i, c)$) becomes available. The total guaranteed falsified weight is thus $rw(s_i) + \sum_{c \in C, c \prec s} wlock(s_i, c) + \sum_{c \in C} ow(c)$, which can be written as $rw(s_i) + \sum_{c \in C, c \prec s} (wlock(s_i, c) - w(c)) + \sum_{c \in C} ow(c)$, because $w(c)=0$ when c is unlocked.

Case (k+1) has two subcases: (i) an empty hard clause cl is produced, then $s_1=\cdots=s_k=1$ is not a feasible assignment and can be ignored; (ii) we have a falsified soft literal *fs* with $rw(fs)>0$ or $\exists c \in C$ such that $wlock(fs, c)>w(c)$, as in the rightmost branch of Figures 2 and 3. If *fs* is not in any core, the total guaranteed falsified weight is equal to $rw(fs) + \sum_{c \in C} ow(c)$. Otherwise, note that the cores in C are unlocked, except eventually those c with $wlock(fs, c)>w(c)$. The falsified weights in these cores due to $fs=0$ in addition to $\sum_{c \in C} ow(c)$ can be written as $\sum_{c \in C} (wlock(fs, c) - w(c))$, where $w(c)=0$ if c is unlocked before *fs* is falsified, and $wlock(fs, c)=0$ if *fs* $\notin c$. Then, the total guaranteed falsified weight is $rw(fs) + \sum_{c \in C, c \prec fs} (wlock(fs, c) - w(c)) + \sum_{c \in C} ow(c)$.

In all the k+1 cases, the first two terms $rw(s_i) + \sum_{c \in C, c \prec s_i} (wlock(s_i, c) - w(c))$ is called *available weight* of s_i when s_i is propagated or falsified (recall s_{k+1}=*fs*), and is denoted as $aw(s_i)$ in line 22.

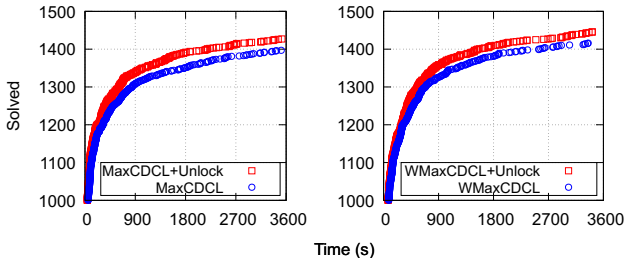

Figure 4: Number of instances solved within x seconds.

In summary, $w(d)$ is the minimum guaranteed falsified weight among the $k{+}1$ cases, which can be written as $min_{s\in d}aw(s){+}\sum_{c\in c} ow(c)$ at line 30.

Note that $w(d){>}\sum_{c\in C} ow(c)$. The increase is $min_{s\in d}aw(s)$ and depends only on the propagated assumptions and fs. So, the remaining weight and the weight locked in d of other literals are not changed. For a propagated assumption s in d, $wlock(s,d)$ is the minimum value that is not smaller than $min_{s\in d}aw(s)$ and its total weight already locked in cores in C. So, $wlock(s,d)$ is equal to $max(\sum_{c\in C} wlock(s,c), min_{s\in d}aw(s))$ (line 26). Its remaining weight $rw(s)$ is computed accordingly (line 27).

From the above analysis, we obtain Proposition 2.

Proposition 2. *Let d be the core identified at line 31 in Alg. 2. Any complete extension of α falsifies at least weight $ow(d)$ in d, and at least weight LB in ϕ.*

Remark. For unweighted MaxSAT, Alg. 2 behaves as Alg. 1, in which every thing is much simpler. We introduced Alg. 1 before Alg. 2 to facilitate the understanding of Alg. 2.

Experimental Evaluation

In this section, we evaluate and analyze the performance of the unlocking mechanism. To this end, we implemented Alg. 1 into MaxCDCL for unweighted MaxSAT, and Alg. 2 into WMaxCDCL for weighted MaxSAT. Since WMaxCDCL frequently splits the weight of a soft literal into locked and remaining weight, which is in complex interaction with the unlocking mechanism, we implemented Alg. 2 in two stages: first, with the unlocking mechanism disabled (i.e., lines 9 to 14). If the first stage is not successful, i.e., if LB<UB after the first stage, Alg. 2 is re-run with the unlocking mechanism enabled to detect additional cores by unlocking existing cores and their locked soft literals.

Experimental settings

Benchmarks We use all instances of the weighted and unweighted exact tracks of the MaxSAT evaluation (MSE) from 2019 to 2023, and divide these 1962 unweighted instances and 2019 weighted instances into the subsets below.

(W)MS19∩20: subset of (weighted) instances used both in MSE19 and MSE20, denoted as 19∩20.

(W)MS19–(W)MS23: (W)MSy, denoted as y, where y ranges from 19 to 23, represents the subset of (weighted) instances used in MSEy but not in (W)MS19∩20 and MSEz for $z < y$.

Set:	19∩20	19	20	21	22	23	Sum
M	**102**	322	290	323	183	177	1397
MU	**102**	**331**	**298**	**330**	**185**	**181**	**1427**
WM	95	281	315	378	203	144	1416
WMU	**97**	**285**	**326**	**384**	**205**	**149**	**1446**

Table 1: Number of instances solved by (W)MaxCDCL with and without the unlocking mechanism.

Compared solvers

(W)MaxCDCL: the baseline solvers that can be found in the website of MSE23, listed as (W)M in the tables.

(W)MaxCDCL+Unlock: it is (W)MaxCDCL in which Alg. 1 (2) is implemented, listed as (W)MU. The source code is available on the MSE24's website under the name (W)MaxCDCL.

WMaxCDCL-S6-HS12 (Coll et al. 2023), **MaxCDCL-S6-HS9** (Li et al. 2023), **EvalMaxSAT-SCIP** (Avellaneda 2023) and **CASHWMAXSAT-CorePlus** (Pan et al. 2023)**:** The best three solvers in the weighted and unweighted exact tracks of MSE23. All are portfolios that incorporate the integer programming solver SCIP (Bestuzheva et al. 2021), and are listed as W23, M23, Eval and Cash, respectively. Specifically, M23 (W23) runs SCIP for 600s, then MaxHS (Bacchus 2022) for 900s (1200s), and finally, (W)MaxCDCL2023 for 2100s (1800s) to solve an instance.

MaxSAT solving has reached a high level of maturity, making further improvements increasingly difficult. In fact, the winner of the exact unweighted (weighted) track of MSE23 solved only 3 (4) more instances than the second-place solver, out of a total of 572 (558) instances.

Experimental environment All experiments ran on Intel Xeon CPUs E5-2680@2.40GHz under Linux with 31GB of memory. The timeout is set to 3600s per instance, as in MSE.

Impact of the unlocking mechanism

Figure 4 displays cactus plots of instances solved within x seconds. Across all values of x, (W)MaxCDCL+Unlock solves more instances than the baseline solver.

Table 1 presents the results for (W)MaxCDCL with and without the unlocking mechanism for 3600s. The data show that this mechanism consistently enables solving more instances. If duplicated instances had not been removed, the difference between them would be even greater. For instance, with the results of (W)MS19∩20 added, the unlocking mechanism increases the number of solved instances by 9 (6) in MSE19, and by 8 (13) in MSE20. A closer analysis reveals that 8 (3) instances are solved by (W)MaxCDCL but not by (W)MaxCDCL+Unlock, and 38 (33) instances are solved by (W)MaxCDCL+Unlock but not by (W)MaxDCL.

We replace (W)MaxCDCL in the portfolio MaxCDCL-S6-HS9 (WMaxCDCL-S6-HS12) by (W)MaxCDCL+Unlock and compare the resulting solver M23U (W23U) with MaxCDCL-S6-HS9 (WMaxCDCL-S6-HS12), Eval and Cash, which are the best three solvers in the unweighted

and weighted exact tracks of MSE23. Tables 2 and 3 show that although a good part of the instances solved by (W)MaxCDCL+Unlock but not by (W)MaxCDCL are also solved by the two other components of MaxCDCL-S6-HS9 (WMaxCDCL-S6-HS12), M23U (W23U) still increases the total number of solved instances. This improvement is particularly notable in the weighted track, where WMaxCDCL -S6-HS12 already outperforms Eval and Cash by a significant margin, indicating that the remaining unsolved instances are especially hard to solve.

We submitted (W)MaxCDCL+Unlock to MSE24. Using Open-WBO as a preprocessing for 300s (1200s), (W)MaxCDCL+Unlock won the first two places in the exact unweighted category among a total of 15 participant solvers.

Empirical analysis

(W)MaxCDCL deals with two types of conflicts during its search: hard conflicts, which occur when a hard clause becomes empty, and soft conflicts, which arise when LB$\geq$UB. Among the 2019 weighted instances, WMaxCDCL solves 118 instances during preprocessing without any conflict. Table 4 partitions the remaining 1901 instances according to the percentage of soft conflicts of an instance over the total number of conflicts detected during the search of WMaxCDCL. We classify the instances in ranges of 20, e.g., column (80,60] contains instances with a percentage of soft conflicts between 80% (excluded) and 60% (included).

We identify the two key groups of instances: 34% with at least 80% soft conflicts, and 39% with at most 20% soft conflicts. Recall that WMaxCDCL+Unlock implements Alg. 2 in two stages. The first stage is executed without the unlocking mechanism, and the second stage is executed only if the first stage is unsuccessful, i.e., if LB$<$UB. The success rate of the second stage is significantly different for these two groups of instances: 34% vs 9%, and so is the number of extra solved instances: 27 vs 2, representing 82% and 6% of the total 33 extra instances solved with the unlocking mechanism, respectively.

The data in Table 5 for unweighted MaxSAT show a similar trend, where the number of remaining unsolved instances after preprocessing is 1786 and 38 instances solved by MaxCDCL+Unlock but not MaxCDCL. The unlocking in MaxCDCL does not adopt the two stages scheme so we do not have the %s2 succ data.

These data offer an intuitive explanation of the effectiveness of the unlocking mechanism: it is most beneficial when hard clauses, including those learned from previous soft conflicts, are not sufficiently effective in pruning the search space. In such cases, the unlocking mechanism should be used to cut subtrees as early as possible. The intricate relationship between hard clauses and the unlocking mechanism warrants further investigation.

In weighted MaxSAT, WMaxCDCL+Unlock outperforms WMaxCDCL by solving 4 or more additional instances in the following families: *scSequencing (7)*, *timetabling (4)* and *judgement (4)*. In unweighted MaxSAT, MaxCDCL+Unlock solves 5 or more additional instances compared to MaxCDCL in the following families: *optic (5)*, *mindset2 (5)* and *phylogeneric-trees (13)*.

Set:	19∩20	19	20	21	22	23	Sum
M23	116	350	327	360	190	180	1523
M23U	**118**	**352**	329	**362**	**193**	182	1536
Eval	**118**	350	330	361	191	**189**	**1539**
Cash	115	344	**333**	359	**193**	183	1527

Table 2: Results for benchmarks of the unweighted track.

Set:	19∩20	19	20	21	22	23	Sum
W23	**111**	**321**	**355**	408	212	165	1572
W23U	**111**	318	**355**	**409**	**219**	166	**1578**
Eval	104	309	335	398	215	**168**	1529
Cash	110	311	349	402	213	154	1539

Table 3: Results for benchmarks of the weighted track.

soft confl %:	[100,80]	(80,60]	(60,40]	(40,20]	(20,0]
#inst	645	185	142	187	742
%inst	34%	10%	7%	10%	39%
#unlock+	27	2	1	1	2
%unlock+	82%	6%	3%	3%	6%
%s2 succ	34%	20%	22%	21%	9%

Table 4: In weighted benchmarks, for each percentage range of soft conflicts: number of instances (#inst) and percentage (%inst) over the 1901 considered instances; extra instances solved thanks to unlocking mechanism(#unlock+) and percentage of the extra solved instances in the range (%unlock+); success rate for stage 2 lookahead (%s2 succ).

soft confl %:	[100,80]	(80,60]	(60,40]	(40,20]	(20,0]
#inst	727	187	92	119	661
%inst	41%	10%	5%	7%	37%
#unlock+	22	10	4	0	2
%unlock	58%	26%	11%	0%	5%

Table 5: Same information as in Table 4 for unweighted MaxSAT benchmarks (stage 2 success does not apply).

Conclusions and Future Work

We proposed an unlocking mechanism that reuses soft clauses from existing cores to detect additional cores, thereby improving the lower bound for BnB MaxSAT solvers. The most challenging aspect of this mechanism is in weighted MaxSAT for calculating the weight of a new core d, the weight of a soft clause cl locked in d, and the remaining weight of cl, when d involves existing cores. We implemented the mechanism in the state-of-the-art solvers (W)MaxCDCL. Extensive experiments on all instances of the MaxSAT Evaluation from 2019 to 2023 demonstrate the effectiveness of the mechanism, which also allowed us to win the first two places in the exact unweighted category in the MaxSAT evaluation 2024. Moreover, we provided an intuitive explanation on when and why it is effective.

As future work, we plan to apply the unlocking mechanism to improve lower or upper bound in other NP-hard combinatorial optimization problems such as MaxClique.

Acknowledgements

This work is partially supported by AI CHAIR reference ANR-19-CHIA-0013-01 (MASSAL'IA) and project ANR-20-ASTR-0011 (POSTCRYPTUM) funded by the French Agence Nationale de la Recherche, and projects PID2022-139835NB-C21 and PID2021-122274OB-I00 funded by MCIN/AEI/10.13039/501100011033, and partially supported by Archimedes Institute, Aix-Marseille University. We want to thank the Université de Picardie Jules Verne for providing the Matrics Platform and the anonymous reviewers for their comments and suggestions that helped to improve the manuscript.

References

Abramé, A.; and Habet, D. 2014. Ahmaxsat: Description and Evaluation of a Branch and Bound Max-SAT Solver. *J. Satisf. Boolean Model. Comput.*, 89–128.

Allouche, D.; André, I.; Barbe, S.; Davies, J.; de Givry, S.; Katsirelos, G.; O'Sullivan, B.; Prestwich, S.; Schiex, T.; and Traoré, S. 2014. Computational protein design as an optimization problem. *Artificial Intelligence*, 212: 59–79.

Alsinet, T.; Manyà, F.; and Planes, J. 2004. A Max-SAT solver with lazy data structures. In *Advances in Artificial Intelligence–IBERAMIA 2004: 9th Ibero-American Conference on AI, Puebla, Mexico, November 22-26, 2004. Proceedings 9*, 334–342. Springer.

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2009. Solving (weighted) partial MaxSAT through satisfiability testing. In *International conference on theory and applications of satisfiability testing*, 427–440.

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2010. A New Algorithm for Weighted Partial MaxSAT. In *Proceedings of AAAI 2010*, 3–8.

Ansótegui, C.; and Gabàs, J. 2017. WPM3: An (in)complete algorithm for weighted partial MaxSAT. *Artificial Intelligence*, 250: 37–57.

Avellaneda, F. 2020. A short description of the solver EvalMaxSAT. *MaxSAT Evaluation*, 8: 364.

Avellaneda, F. 2023. EvalMaxSAT 2023. *MaxSAT Evaluation 2023*, 12–13.

Bacchus, F. 2020. MaxHS in the 2020 MaxSAT Evaluation. In *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, 19–20.

Bacchus, F. 2022. MaxHS in the 2022 MaxSat Evaluation. *MaxSAT Evaluation 2022*, 17–18.

Bacchus, F.; Järvisalo, M.; and Ruben, M. 2021. Maximum Satisfiability. In *Handbook of satisfiability, second edition*, 929–991. IOS Press.

Berg, J.; Bogaerts, B.; Nordström, J.; Oertel, A.; and Vandesande, D. 2023. Certified core-guided MaxSAT solving. In *International Conference on Automated Deduction*, 1–22. Springer.

Bestuzheva, K.; Besançon, M.; Chen, W.-K.; Chmiela, A.; Donkiewicz, T.; van Doornmalen, J.; Eifler, L.; Gaul, O.; Gamrath, G.; Gleixner, A.; et al. 2021. The SCIP optimization suite 8.0. *arXiv preprint arXiv:2112.08872*.

Cherif, M. S.; Habet, D.; and Abramé, A. 2020. Understanding the power of Max-SAT resolution through UP-resilience. *Artificial Intelligence*, 103397.

Chu, Y.; Cai, S.; and Luo, C. 2023a. NuWLS-c-2023: Solver description. *MaxSAT Evaluation 2023*, 23–24.

Chu, Y.; Cai, S.; and Luo, C. 2023b. NuWLS: Improving local search for (weighted) partial MaxSAT by new weighting techniques. In *Proceedings of AAAI 2023*, volume 37, 3915–3923.

Coll, J.; Li, S.; Li, C.-M.; Manyà, F.; Habet, D.; Cherif, M. S.; and He, K. 2023. WMaxCDCL in MaxSAT Evaluation 2023. *MaxSAT Evaluation 2023*, 16–17.

Demirović, E.; Musliu, N.; and Winter, F. 2019. Modeling and solving staff scheduling with partial weighted maxSAT. *Annals of Operations Research*, 275: 79–99.

Fu, Z.; and Malik, S. 2006. On Solving the Partial MAX-SAT Problem. In *Proceedings of SAT 2006*, 252–265.

Heras, F.; Larrosa, J.; and Oliveras, A. 2008. MiniMaxSAT: An Efficient Weighted Max-SAT Solver. *Journal of Artificial Intelligence Research*, 1–32.

Koshimura, M.; Zhang, T.; Fujita, H.; and Hasegawa, R. 2012. QMaxSAT: A partial Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 95–100.

Kuegel, A. 2010. Improved Exact Solver for the Weighted MAX-SAT Problem. In *Proceedings of Workshop Pragmatics of SAT, POS-10, Edinburgh, UK*, 15–27.

Lei, Z.; and Cai, S. 2018. Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT. In *Proceedings of IJCAI 2018*, 1346–1352.

Lei, Z.; Cai, S.; Geng, F.; Wang, D.; Peng, Y.; Wan, D.; Deng, Y.; and Lu, P. 2021. Satlike-c: Solver description. *MaxSAT Evaluation*, 2021: 19–20.

Li, C.-M.; Coll, J.; Li, S.; Habet, D.; Manyà, F.; and He, K. 2023. MaxCDCL in MaxSAT Evaluation 2023. *MaxSAT Evaluation 2023*, 14–15.

Li, C. M.; and Manyà, F. 2021. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability, second edition*, 903–927. IOS Press.

Li, C. M.; Manyà, F.; and Planes, J. 2005. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *International conference on principles and practice of constraint programming*, 403–414. Springer.

Li, C. M.; Manyà, F.; and Planes, J. 2006. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *AAAI*, volume 6, 86–91.

Li, C. M.; Manyà, F.; and Planes, J. 2007. New Inference Rules for Max-SAT. *Journal of Artificial Intelligence Research*, 321–359.

Li, C. M.; and Quan, Z. 2010. An Efficient Branch-and-Bound Algorithm Based on MaxSAT for the Maximum Clique Problem. In *Proceedings of AAAI 2010*, 128–133.

Li, C.-M.; Xu, Z.; Coll, J.; Manyà, F.; Habet, D.; and He, K. 2021a. Boosting branch-and-bound MaxSAT solvers with clause learning. *AI Communications*, 1–21.

Li, C.-M.; Xu, Z.; Coll, J.; Manyà, F.; Habet, D.; and He, K. 2021b. Combining clause learning and branch and bound for MaxSAT. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*.

Martins, R.; Joshi, S.; Manquinho, V.; and Lynce, I. 2014. Incremental cardinality constraints for MaxSAT. In *Proceedings of CP 2014*, 531–548.

Martins, R.; Manquinho, V. M.; and Lynce, I. 2014. Open-WBO: A Modular MaxSAT Solver. In *Proceedings of SAT 2014*, 438–445.

Nadel, A. 2023. TT-Open-WBO-Inc-23: an Anytime MaxSAT Solver Entering MSE'23. *MaxSAT Evaluation 2023*, 29.

Narodytska, N.; and Bacchus, F. 2014. Maximum Satisfiability Using Core-Guided MaxSAT Resolution. In *Proceedings of AAAI 2014*, 2717–2723.

Pan, S.; Wang, Y.; Lei, Z.; Cai, S.; Wang, S.; and Yin, M. 2023. CASHWMaxSAT-CorePlus: Solver Description. *MaxSAT Evaluation 2023*, 8.

Papadimitriou, C. H. 1994. *Computational complexity*. Addison-Wesley.

Paxian, T.; and Becker, B. 2020. Pacose: An Iterative SAT-based MaxSAT Solver. In *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, 12.

Shen, H.; and Zhang, H. 2004. Study of lower bound functions for max-2-sat. In *AAAI*, volume 2004, 185–190.

Wallace, R. J.; and Freuder, E. C. 1993. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In *Cliques, Coloring, and Satisfiability*, 587–615. Citeseer.

Zheng, J.; He, K.; Jin, M.; Chen, Z.; and Xue, J. 2023. Combining BandMaxSAT and FPS with NuWLS-c. *MaxSAT Evaluation 2023*, 25–26.

Zheng, J.; He, K.; Zhou, J.; Jin, Y.; Li, C.-M.; and Manyà, F. 2022. BandMaxSAT: A Local Search MaxSAT Solver with Multi-armed Bandit. In *Proceedings of IJCAI 2022*, 1901–1907.