Combining Clause Learning and Branch and Bound for MaxSAT

Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, Kun He

Combining Clause Learning and Branch and Bound for MaxSAT

Chu-Min Li
Huazhong University of Science and Technology, Wuhan, China
Université de Picardie Jules Verne, Amiens, France
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Zhenxing Xu
Huazhong University of Science and Technology, Wuhan, China

Jordi Coll
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Felip Manyà
Artificial Intelligence Research Institute, CSIC, Bellaterra, Spain

Djamal Habet
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Kun He
Huazhong University of Science and Technology, Wuhan,China

Abstract

Branch and Bound (BnB) is a powerful technique that has been successfully used to solve many combinatorial optimization problems. However, MaxSAT is a notorious exception because BnB MaxSAT solvers perform poorly on many instances encoding interesting real-world and academic optimization problems. This has formed a prevailing opinion in the community stating that BnB is not so useful for MaxSAT, except for random and some special crafted instances. In fact, there has been no advance allowing to significantly speed up BnB MaxSAT solvers in the past few years, as illustrated by the absence of BnB solvers in the annual MaxSAT Evaluation since 2017. Our work aims to change this situation and proposes a new BnB MaxSAT solver, called MaxCDCL, by combining clause learning and an efficient bounding procedure. The experimental results show that, contrary to the prevailing opinion, BnB can be competitive for MaxSAT. MaxCDCL is ranked among the top 5 solvers of the 15 solvers that participated in the 2020 MaxSAT Evaluation, solving a number of instances that other solvers cannot solve. Furthermore, MaxCDCL, when combined with the best existing solvers, solves the highest number of instances of the MaxSAT Evaluations.

2012 ACM Subject Classification Software and its engineering → Constraints

Keywords and phrases MaxSAT, Branch&Bound, CDCL

Digital Object Identifier 10.4230/LIPIcs.CP.2021.38

Supplementary Material *Software (Source Code)*: `https://home.mis.u-picardie.fr/~cli/EnglishPage.html`

Funding This work has been partially funded by the French Agence Nationale de la Recherche, reference ANR-19-CHIA-0013-01, and the Spanish AEI project PID2019-111544GB-C2.

Acknowledgements This work has been partially supported by Archimedes Institute, Aix-Marseille University. We thank the anonymous reviewers for their comments and suggestions that helped to improve the manuscript.

1 Introduction

The Maximum satisfiability problem (MaxSAT) is an optimization version of a well-studied and canonical NP-Complete problem, the satisfiability problem (SAT). Although SAT and MaxSAT share many aspects, solving MaxSAT is much harder than solving SAT in practice.

Indeed, since several clauses can be falsified in an optimal MaxSAT solution, some fundamental SAT techniques such as unit propagation cannot be used in MaxSAT as they are used in SAT. Despite this difficulty, huge efforts made by researchers make it possible nowadays to solve many interesting real-world and academic NP-hard optimization problems encoded as MaxSAT instances [11, 26]. For this reason, MaxSAT has attracted increasing interest from the academy and industry in recent years.

As for many NP-hard problems, algorithms for MaxSAT are divided into two categories: exact algorithms, which return optimal solutions and prove their optimality; and heuristic algorithms, which quickly find solutions of good quality without guaranteeing their optimality. This paper will focus on exact algorithms for (unweighted) MaxSAT.

We roughly distinguish two types of exact algorithms for MaxSAT: branch-and-bound (BnB) algorithms [26], which directly tackle MaxSAT with a bounding procedure, but without unit propagation and clause learning; and SAT-based algorithms [11], which transform MaxSAT into a sequence of SAT instances and call a CDCL (Conflict-Driven Clause Learning) SAT solver to solve them. The performance of SAT-based MaxSAT algorithms is usually much better than BnB MaxSAT solvers in solving many real-world NP-hard optimization problems, because they indirectly exploit clause learning via the SAT solver. Unfortunately, it is hard for a BnB solver to exploit clause learning. In a CDCL SAT solver, a backtracking happens only when a clause is falsified, from which a sequence of resolution steps is performed to learn a clause explaining the backtracking. However, a BnB MaxSAT solver also need to backtrack when it computes a lower bound equal to the upper bound. In this case, no clause is explicitly falsified, making it hard to learn a clause. Probably because of this difficulty, there has been no advance allowing to significantly speed up BnB MaxSAT solvers in recent years, as illustrated by their absence in the annual MaxSAT Evaluation since 2017.

In this paper, we propose an original approach that allows a BnB MaxSAT solver to learn a clause when it computes a lower bound equal to the upper bound, together with a new bounding procedure, because the one in current BnB MaxSAT solvers is not adequate for large instances. This approach is implemented in a new BnB MaxSAT solver, called MaxCDCL, that combines the new bounding procedure and clause learning. The experimental results show that MaxCDCL is ranked among the top 5 solvers of the 15 solvers that participated in the 2020 MaxSAT Evaluation, solving a number of instances that other solvers cannot solve. Furthermore, MaxCDCL, when combined with the best existing solvers, solves the highest number of instances of the MaxSAT Evaluations.

Combining clause learning and BnB, as clause learning itself, belongs to the general framework consisting in explaining a failure in the search to avoid the same failure in the future. In other fields such as Pseudo-Boolean Optimization (PBO), there are also works in this framework (e.g. [17]). The general framework is not hard to understand. However, making it effective for solving a particular problem such as SAT or MaxSAT is quite challenging, because this requires a deep understanding of the problem and the related solving techniques. So, one important contribution of our work is that we found a configuration and an efficient implementation of this configuration allowing to make the combination of clause learning and BnB effective for MaxSAT, as presented in this paper.

More importantly, our results refute a prevailing opinion in the field stating that, although BnB is a powerful technique that has successfully been used to solve many combinatorial optimization problems, it is not so useful for MaxSAT. Indeed, as the first BnB MaxSAT solver successfully exploiting clause learning, MaxCDCL opens promising research directions.

This paper is organized as follows: Section 2 presents the preliminaries. Section 3 reviews state-of-the-art MaxSAT solvers. Section 4 describes MaxCDCL. Section 5 empirically evaluates and analyzes MaxCDCL. Section 6 concludes.

2 Preliminaries

A propositional *variable* x can take values 0 or 1 (false or true). A *literal* is a variable x or its negation $\neg x$. A *clause* is a disjunction of k literals $l_1 \vee \cdots \vee l_k$. A *propositional formula in Conjunctive Normal Form* (CNF) is a conjunction (or a set) of m clauses $c_1, \ldots, c_m$. An *assignment* of truth values to the propositional variables satisfies a literal x if $x = 1$, and satisfies a literal $\neg x$ if $x = 0$. A literal x or $\neg x$ is *assigned* if x is assigned a value, otherwise it is *free*. An assignment is *complete* if all the variables are assigned a value, otherwise it is *partial*. A (partial) assignment is usually represented by a (sub)set of satisfied literals. A clause is satisfied if at least one of its literals is satisfied. A CNF is satisfied if all its clauses are satisfied. The *Boolean satisfiability* (SAT) problem for a CNF ϕ is to determine whether there exists an assignment that satisfies ϕ.

MaxSAT is the problem of finding an assignment that satisfies the maximum number of clauses in a given multiset of clauses. In partial MaxSAT, there are *hard* and *soft* clauses, and the goal is to satisfy all the hard clauses and the maximum number of soft clauses. In weighted (partial) MaxSAT, each soft clause has a cost to pay if it is violated. This paper will focus exclusively on unweighted (partial) MaxSAT.

The state-of-the-art SAT solvers implement the CDCL algorithm. CDCL alternates a search phase, where literals are assigned until either a solution or a conflict is found, and a learning phase, which is executed after finding a conflict in order to learn a new clause. *Unit Propagation* (UP) is the main inference rule applied during the search: If there is a unit clause $\{l\}$ in ϕ, literal l must be satisfied (i.e., set to 1). Then, any clause containing l is removed from ϕ, and all the occurrences of $\neg l$ in clauses of ϕ are (implicitly) removed. UP is applied during the search until an empty clause (*conflict*) is found or no unit clause exists in ϕ. If UP finishes without finding a conflict, a new literal is picked following a heuristic and is set to 1 (we make a *decision*), and UP is applied again. If all the variables are assigned without finding a conflict, ϕ is satisfiable. The *decision level* of an assigned literal is the number of decisions made before being assigned. When a conflict is found, a *conflict analysis* is performed on the *implication graph* to derive a new clause explaining the conflict.

Figure 1 shows an example of implication graph. It is a directed acyclic graph where each node represents an assignment $l@dl$, where l is a literal set to 1 and dl is its decision level. The negations of the literals of incoming edges of a node $l@ld$ represent the reason (clause) why UP has set $l = 1$. For instance, node $x_{12}@5$ is propagated due to clause $\neg x_6 \vee \neg x_4 \vee x_{12}$ (i.e., $x_6 \wedge x_4 \rightarrow x_{12}$), given that x_4 and x_6 have been set to 1. A node without incoming edges represents a decision. All decisions are painted in grey, and the last one is dashed. A conflicting clause is represented by incoming edges to $\square$; in this example, clause $\neg x_3 \vee \neg x_{35}$. A *Unique Implication Point* (UIP) is a node of the implication graph that belongs to all paths from the last decision to the conflict. Figure 1 contains three UIPs: the last decision x_4, and literals x_{13} and x_{25}. The most used learning schema in CDCL SAT solvers is called *first UIP (1UIP)*, guided by the closest UIP to the conflict in the implication graph.

When a conflict is found in the decision level dl (i.e., dl decisions were made before the conflict is found), a node in the decision level dl is said *active* if it is not the 1UIP but is in a path from the 1UIP to the conflict. In other words, the active nodes are those nodes that allow to reach the conflict from the 1UIP in the decision level dl. For example, in Figure 1, a conflict is found in the decision level 5, and $x_3@5$ and $x_{35}@5$ are active nodes. The 1UIP learning schema identifies, in each path from a node in a decision level lower than dl to an active node, the last literal in the lower decision level. The new learnt clause is composed of

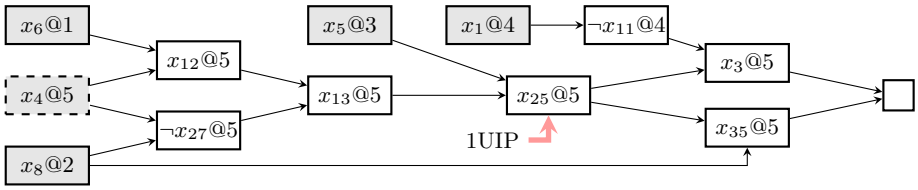

Figure 1 Example of implication graph.

the negation of these literals and the 1UIP. For instance, in Figure 1, clause $\neg x_8 \vee x_{11} \vee \neg x_{25}$ is learnt, because there are two paths from lower decision levels to an active node, in which x_8 and $\neg x_{11}$ are the last literals in lower decision levels, respectively.

The learnt clause explains the conflict: when all its literals are falsified, unit propagation reproduces the implication graph to derive the same conflict. So, adding the learnt clause prevents the same conflict in the future search. After learning a clause, CDCL backtracks to the second highest decision level of the learnt clause (level 4 in the above example). Unsatisfiability is determined when a conflict is found at decision level 0.

3 Related Work

A major difference between MaxSAT and SAT solvers is that each clause must be satisfied in a SAT solution while a soft clause can be falsified in an optimal MaxSAT solution, making MaxSAT much harder to solve than SAT in practice. Despite this, the MaxSAT community has made huge efforts to implement exact MaxSAT solvers with impressive performance over the last decade [11, 26]. On the other hand, heuristic MaxSAT algorithms such as SatLike [25] have also been proposed.

Roughly speaking, we find two main groups of exact MaxSAT solvers: branch-and-bound (BnB) and SAT-based solvers. BnB MaxSAT solvers implement the branch-and-bound scheme and incorporate a lookahead procedure that detects inconsistent subsets of soft clauses by applying unit propagation and computes a lower bound [26]. They also apply some inference rules at each node of the search tree. Representative BnB solvers are MaxSatz [30], MiniMaxSat [20], Ahmaxsat [1, 14] and Akmaxsat [24]. Closely related to MaxSAT, we can find BnB solvers for the Weighted Constraint Satisfaction Problem (WCSP). Recently, it was presented a technique to improve BnB WCSP solving by avoiding branching on variables which are unlikely to increase the lower bound [43].

SAT-based MaxSAT solvers proceed by reformulating the MaxSAT optimization problem into a sequence of SAT decision problems [11]. These solvers could still be divided into three subgroups: model-guided, core-guided and Minimum Hitting Sets (MHS-)guided. Model-guided approaches reduce to SAT the problem of deciding whether there exists an assignment for the MaxSAT instance with a cost less than or equal to a certain k, and successively decrease k until an unsatisfiable SAT instance is found. Among such solvers we find SAT4J-Maxsat [12], QMaxSat [23, 44], Open-WBO [37] or Pacose [40]. Core-guided and MHS-guided approaches consider a MaxSAT instance as a SAT instance and call a CDCL SAT solver to identify an unsatisfiable subset of soft clauses, called a *core*. Then, they relax this core and solve the relaxed instance with a CDCL SAT solver to identify another core, repeating this process until deriving a satisfiable instance. The difference between them is that core-guided solvers relax a core using cardinality constraints, while MHS-guided solvers remove one clause from each detected core so that the number of different clauses removed from the cores is minimized by solving a minimum hitting set instance with an

integer programming solver. The most representative core-guided solvers include msu1.2 [36], WBO [35], Open-WBO [37], WPM1 [3], PM2 [5], WPM2 [4], WPM3 [6], Eva [39], RC2 [21], and the most representative MHS-guided solvers include MHS [41] and MaxHS [8, 10, 15, 16]. Core-guided search has also been extended to constraint programming [19].

A common point of SAT-based MaxSAT solvers is that they *indirectly* exploit the clause learning technique by repeatedly calling a CDCL SAT solver. Unfortunately, it is hard for BnB solvers to exploit clause learning, which might explain their poor performance on real-world optimization problems. We are aware of only one tentative in [2]: when the number of falsified soft clauses reaches the upper bound, the falsification of these soft clauses is analyzed to learn a clause. Nevertheless, no clause is learnt when the lookahead procedure returns a lower bound equal to the upper bound, and the reported results are not competitive.

4 MaxCDCL: A BnB Algorithm Using CDCL for MaxSAT

This section first presents the general structure of MaxCDCL, and then the different components of our approach implemented in MaxCDCL.

4.1 General Structure of MaxCDCL

We distinguish between hard and soft conflicts in the MaxSAT context. A *hard* conflict occurs when the current partial assignment falsifies a hard clause. Given an upper bound UB, a *soft* conflict occurs when the current partial assignment cannot be extended to a complete one falsifying fewer than UB soft clauses. A CDCL SAT solver only considers hard conflicts, learns a hard clause from each hard conflict and backtracks. A BnB CDCL MaxSAT solver extends the CDCL SAT solver by also learning a hard clause from each discovered soft conflict and backtracks. Algorithm 1 depicts such a BnB CDCL MaxSAT solver called MaxCDCL.

Given a MaxSAT instance with a set of hard clauses and a set of soft clauses, MaxCDCL works with H and S, where H contains the hard clauses and S contains a new literal y for each soft clause sc after adding the hard clauses encoding $y \leftrightarrow sc$ to H. We call such y *soft literal*, because it represents a soft clause (i.e., a soft literal is satisfied if and only if the corresponding soft clause is satisfied). To solve the MaxSAT instance, MaxCDCL is repeatedly called with UB$= 2^0, 2^1, 2^2, 2^3, \ldots$ without exceeding $|S| + 1$ or $\text{UB}_f + 1$ where UB_f is a feasible upper bound computed by a heuristic solver with a short cutoff. This process stops when it obtains an assignment satisfying all clauses in H and falsifying fewer than UB soft literals in S. After that, MaxCDCL is repeatedly called with UB set to the number of falsified soft literals in the previous call, until no better solution can be found.

MaxCDCL is like a CDCL SAT solver except for two points. First, when UP does not falsify any hard clause in the UPLA procedure, it calls, under certain condition, a lookahead (LA) procedure to compute a lower bound $|cores|$ of the number of soft literals that will be falsified if the current partial assignment is extended. If a soft conflict is discovered, i.e., if $|falseS| + |cores| \geq$UB, it is analyzed to learn a clause for backtracking. Second, if no soft conflict is discovered, but it is discovered that the falsification of any free soft literal y not used in $|cores|$ would result in a soft conflict, y is satisfied by a procedure called *hardening*.

Note that model-guided MaxSAT solvers also set a UB in their search. However, they do not compute a lower bound to discover soft conflicts as MaxCDCL. Consequently, MaxCDCL is able to backtrack much earlier than model-guided MaxSAT solvers for a given UB. See the next subsection for details and illustrative examples.

Algorithm 1 MaxCDCL(H, S,UB), a generic CDCL procedure with lookahead for MaxSAT.

Input: H: a set of hard clauses, S: a set of soft literals, UB: an upper bound.
Output: $|falseS|$, where $falseS$ is the set of falsified soft literals, if $|falseS|$ <UB; or UB otherwise

```
1  begin
2      while true do
3          currentLevel ← 0; /* start or restart search */
4          while true do
5              (cl, falseS, cores, reasons) ← UPLA(H, S, UB, currentLevel);
6              if cl is a falsified hard clause or |falseS| + |cores| ≥ UB then
7                  if currentLevel = 0 then
8                      return UB;
9                  else
10                     newLearntClause ← analyze(cl, falseS, reasons);
11                     level ← the second highest level in newLearntClause;
12                     backtrackTo(level);
13                     currentLevel ← level;
14             else
15                 if all variables are assigned then
16                     return |falseS|;
17                 else if |falseS| + |cores| = UB−1 then
18                     hardening();
19                 else if restart condition is satisfied then
20                     backtrackTo(0);
21                     break; /* restart */
22                 else
23                     currentLevel++;
24                     H ← H ∪ {l} where l is a free literal selected using a heuristic;
```

4.2 Combining Lookahead and Clause Learning

A subset of soft literals $S_i = \{y_1, \ldots, y_{|S_i|}\}$ is inconsistent if they cannot be simultaneously satisfied. This inconsistency can be represented by the hard clause $\neg y_1 \vee \cdots \vee \neg y_{|S_i|}$. Note that $|S_i|$ can be 1. If the inconsistency is independent of any partial assignment, the subset is called a *global core*. Otherwise, the inconsistency is implied by a subset of literals and the inconsistent subset of soft literals is called a *local core*. The core-guided or MHS-guided SAT-based MaxSAT solvers only detect global cores to relax them, while our approach detects local cores, given a partial assignment, to discover a soft conflict.

▶ **Example 1.** Let $H = \{\neg y_1 \vee x_1 \vee \neg x_2, \neg x_1 \vee \neg x_3 \vee \neg x_4, \neg y_2 \vee x_3, \neg y_3 \vee x_5\}$, where y_1, y_2 and y_3 are soft literals. If no variable is assigned, all soft literals can be simultaneously satisfied. So, no global core exists. However, if the current partial assignment is $\{x_2 = 1, x_4 = 1\}$, the subset of soft literals $\{y_1, y_2\}$ is a local core implied by the partial assignment. We write the implication by $H \cup \{x_2, x_4\} \rightarrow \neg y_1 \vee \neg y_2$.

Proposition 2 provides the foundation of our approach in the general case.

▶ **Proposition 2.** *Let H be a set of hard clauses, $S = \{y_1, \ldots, y_{|S|}\}$ be the set of all soft literals, k be an integer, and $L_i = \{l_{i1}, \ldots, l_{i|L_i|}\}$ $(1 \leq i \leq k)$ be a set of literals. If, for every i $(1 \leq i \leq k)$, $H \cup L_i$ implies a local core $S_i = \{z_{i1}, \ldots, z_{i|S_i|}\} \subset S$ (i.e., $H \cup L_i \rightarrow \neg z_{i1} \vee \cdots \vee \neg z_{i|S_i|}$), and S_i and S_j are disjoint for any $j \neq i$ such that $1 \leq j \leq k$, then every assignment that satisfies $H \cup \{\neg y_1 + \cdots + \neg y_{|S|} < k\}$ also satisfies the clause $\neg l_{11} \vee \cdots \vee \neg l_{1|L_1|} \vee \cdots \vee \neg l_{k1} \vee \cdots \vee \neg l_{k|L_k|}$.*

Proof. It is easy to see that $H \cup L_1 \cup \cdots \cup L_k$ implies $(\neg z_{11} \vee \cdots \vee \neg z_{1|S_1|}) \wedge \cdots \wedge (\neg z_{k1} \vee \cdots \vee \neg z_{k|S_k|})$, meaning that $H \cup L_1 \cup \cdots \cup L_k$ falsifies the constraint $\neg y_1 + \cdots + \neg y_{|S|} < k$, because each clause $\neg z_{i1} \vee \cdots \vee \neg z_{i|S_i|}$ implies at least one different falsified soft literal. Hence, any assignment satisfying $H \cup \{\neg y_1 + \cdots + \neg y_{|S|} < k\}$ must falsify at least one literal in $L_1 \cup \cdots \cup L_k$, and satisfy the clause $\neg l_{11} \vee \cdots \vee \neg l_{1|L_1|} \vee \cdots \vee \neg l_{k1} \vee \cdots \vee \neg l_{k|L_k|}$. Note that we use "z" (instead "y") to denote a soft literal in a local core to avoid complex subscripts. ◀

Given a partial assignment F of H, the application of Proposition 2 consists in first detecting a local core S_i and then identifying the smallest $L_i \subset F$ such that $H \cup L_i$ implies S_i. We call L_i the *reason* of S_i. If k is the current upper bound UB and k disjoint local cores are detected, a soft conflict is discovered, and the clause $\neg l_{11} \vee \cdots \vee \neg l_{1|L_1|} \vee \cdots \vee \neg l_{k1} \vee \cdots \vee \neg l_{k|L_k|}$, which is implied by $H \cup \{\neg y_1 + \cdots + \neg y_{|S|} < k\}$ and is falsified by the current partial assignment, can be considered by an implicit clause explaining the soft conflict. This clause can be further analyzed using the 1UIP schema in line 10 of Algorithm 1 to learn a clause to be explicitly added to H to prevent the same soft conflict in the future as in the hard conflict case.

The detection of a local core S_i is implemented by using UP in a lookahead procedure as in existing BnB MaxSAT solvers [27, 28, 29]. The advantage of this procedure is that S_i is minimal w.r.t. UP, in the sense that UP cannot detect any local core that is a proper subset of S_i under the same partial assignment [27], which is essential for our approach, because MaxCDCL needs to learn clauses of good quality from the detected local cores. Recall that BnB MaxSAT solvers detect disjoint local cores but do not explain them. When a soft conflict is discovered, they simply backtrack without learning a hard clause from the soft conflict, which is very different from MaxCDCL.

Concretely, MaxCDCL calls Algorithm 2 at decision level dl with a partial assignment F, under which the already falsified soft literals are stored in a set named $falseS$. The lookahead procedure starts at line 5 in Algorithm 2 when no hard conflict is found, and terminates at line 10 or line 24. This procedure proceeds in decision level $dl{+}1$ by maintaining a set of detected local cores ($cores$). Every iteration of the loop (line 8) propagates a free soft literal y not occurring in $cores$, until a clause h in H is falsified or a soft literal sl not occurring in $cores$ is falsified (line 13). These unit propagations construct an implication graph G. The propagated free soft literals $y_1, y_2, \ldots$ in the loop can be seen as temporary assumptions at decision level $dl{+}1$ that have no incoming edge in G.

Inspecting G, let $z_1, \ldots, z_b$ be the subset of literals $y_1, y_2, \ldots$ at level $dl{+}1$ and without incoming edge, from which there is a path to h or to $\neg sl$. Then, $\{z_1, \ldots, z_b\}$ (resp. $\{z_1, \ldots, z_b, sl\}$) is a local core S_i implied by the partial assignment F at decision level dl. Based on G, we can also define L_i as the set containing, from each path to h (resp. $\neg sl$), the last literal assigned before level $dl + 1$. Clearly, $H \cup L_i$ implies $\neg z_1 \vee \ldots \vee \neg z_b$ (resp. $\neg z_1 \vee \ldots \vee \neg z_b \vee \neg sl$).

Note that each already falsified soft literal y in $falseS$ constitutes a local core $S_i{=}\{y\}$ not in $cores$, implied by $H \cup \{\neg y\}$ (i.e. $L_i{=}\{\neg y\}$). Therefore, when $k = |falseS| + |cores| = \text{UB}$ in MaxCDCL, we have UB disjoint local cores. According to Proposition 2, $\neg l_{11} \vee \cdots \vee$

Algorithm 2 UPLA(H, S, UB, dl), Unit propagation followed by lookahead.

Input: H: a set of hard clauses, S: a set of soft literals, UB: an upper bound, dl: the current decision level.

Output: cl: a hard clause; $falseS$: the set of falsified soft literals; $cores$: a set of disjoint local cores; $reasons$: a set of literals that are reasons of $cores$

```
1  begin
2      (cl, falseS) ← UP(H);
3      if cl is a falsified hard clause or |falseS| ≥ UB or the condition to lookahead is
        not satisfied then
4          return (cl, falseS, ∅, ∅);
5      H' ← H; F ← {¬l|l is falsified in H};
6      reasons ← ∅; cores ← ∅; S' ← ∅;
7      Increase the decision level to dl + 1;
8      while true do
9          if all soft literals of S either are non-free or occur in cores then
10             return (cl, falseS, cores, reasons);
11         Let y be a free soft literal not occurring in cores;
12         H ← H ∪ {y}; S' ← S' ∪ {y};
13         (h, sl) ← UPforLA(H) ;
14         if h is a falsified hard clause or sl a falsified soft literal not in cores then
15             if h is a falsified hard clause then
16                 core ← {l|l ∈ S' and l has no incoming edge and
                     there is a path from l to h in the implication graph};
17             else
18                 core ← {sl} ∪ {l|l ∈ S' and l has no incoming edge and
                     there is a path from l to ¬sl in the implication graph};
19             reason ← {l|l ∈ F and there is a path from l to h or ¬sl
                 in the implication graph, and the literal next to l in the path is
                 of decision level dl + 1};
20             cores ← cores ∪ {core};
21             reasons ← reasons ∪ reason;
22             H ← H'; S' ← ∅; /*Cancel UP done by lookahead*/
23             if |cores| + |falseS| ≥ UB then
24                 return (cl, falseS, cores, reasons);
```

$\neg l_{1|L_1|} \vee \cdots \vee \neg l_{k1} \vee \neg l_{k2} \vee \cdots \vee \neg l_{k|L_k|}$ could be considered as an implicit clause in H, which is falsified by the current partial assignment and is analyzed using the 1UIP schema in line 10 of Algorithm 1 to learn a clause as in the hard conflict case.

▶ **Example 3.** Let $y_1, \ldots, y_7$ be soft literals, and let H be formed by

$$\neg y_1 \vee y_2 \vee \neg x_2 \quad \neg y_2 \vee \neg x_3 \vee \neg x_4 \quad \neg y_2 \vee \neg y_5 \vee x_6 \quad \neg x_1 \vee x_4 \quad \neg y_3 \vee x_3$$
$$\neg x_6 \vee \neg x_5 \vee y_6 \quad \neg y_6 \vee \neg x_2 \vee x_7 \quad \neg x_7 \vee \neg y_6 \quad \neg y_4 \vee x_8 \quad \neg y_7 \vee x_9$$

If no variable is assigned, all soft literals can be simultaneously satisfied. So, no global core exists. Let the current partial assignment be $\{y_7{=}0, x_2{=}1, x_1{=}1, x_4{=}1, x_5{=}1\}$, where x_4 is propagated due to $\neg x_1 \vee x_4$, and the other literals have been decided. Hence $falseS = \{y_7\}$ and the current decision level is 4. Let also UB=3. We show how Algorithm 2 detects $cores$ and reaches UB by propagating the free soft literals $y_1, \ldots, y_6$, by means of Figure 2.

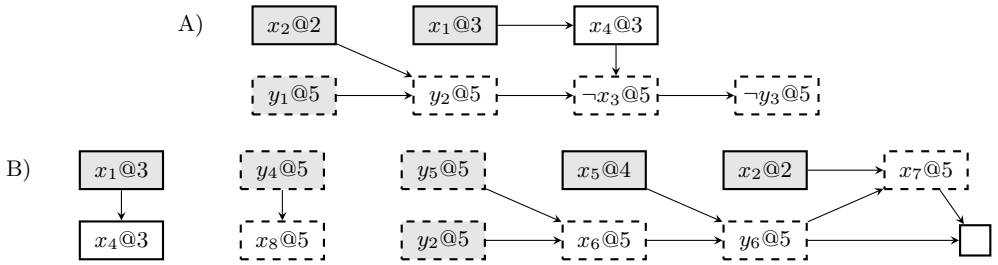

Figure 2 Implication graphs involved in the detection of *cores* in Example 3. Grey nodes represent decisions or assumptions, and dashed nodes represent assignments made in the lookahead process.

After propagating y_1, y_2 is satisfied and y_3 is falsified. Hence $sl{=}y_3$ is detected, the implication graph in Figure 2A is obtained, looking at which $core{=}\{y_1, y_3\}$ will be identified at line 18 with reason$=\{x_2, x_4\}$, because there are paths from x_2 and x_4 to $\neg y_3$, and the next literals to x_2 and x_4 in the paths are from decision level 5.

Note that since y_2 is not in *cores*, it can be used to detect new cores. After propagating y_2, y_4, y_5, the implication graph of Figure 2B is obtained, and a conflicting clause $h{=}\{\neg x_7 \vee \neg y_6\}$ is detected. Then, $core{=}\{y_2, y_5\}$ is detected at line 16 with reason$=\{x_2, x_5\}$.

Since $|cores| + |falseS| = 2 + 1 = \text{UB}$, a soft conflict is found at line 23, with $reasons{=}\{x_2, x_4, x_5, \neg y_7\}$. According to Proposition 2, clause $\neg x_2 \vee \neg x_4 \vee \neg x_5 \vee y_7$ is implied by $H \cup \{\neg y_1 + \cdots + \neg y_{|S|} < 3\}$, which is falsified at decision level 4, and hence can be analysed with the 1UIP scheme at level 4 to learn a new clause and backtrack. Note that a model-guided MaxSAT solver would not discover this soft conflict at this stage, because UB=3 but only one soft literal is falsified.

Proposition 2 also provides the basis for hardening. If $k = |falseS| + |cores| = \text{UB}{-}1$, for each free soft literal z not in the k local cores, $H \cup \{\neg z\}$ implies a new local core $\{z\}$. The hardening procedure (line 18 in Algorithm 1) satisfies z with the reason $z \vee \neg l_{11} \vee \cdots \vee \neg l_{1|L_1|} \vee \cdots \vee \neg l_{k1} \vee \cdots \vee \neg l_{k|L_k|}$. Note that this hardening satisfies a fundamental requirement of CDCL SAT solvers: except the decision (i.e., branching) variables, the value of every variable must be explicitly associated with a reason. In contrast, although existing BnB MaxSAT solvers also implement hardening, no reason is associated with the hardening.

Let $m = |H| + |S|$. UP is in $O(m)$ for detecting one local core. Since the lookahead procedure detects at most UB cores, its whole complexity is in $O(m{\times}\text{UB})$.

4.3 A Probing Strategy for Lookahead

Existing BnB MaxSAT solvers usually tackle random or crafted instances of limited size and look ahead at each branch. However, such a treatment might be too costly and useless for large instances. If the lower bound is not tight enough to prune the current branch, the time spent to compute the lower bound is lost. When $k = \text{UB}{-}|falseS|$, the lookahead procedure has to detect k disjoint local cores to be successful. Generally, the greater the value of k, the lower the probability of lookahead to be successful.

MaxCDCL uses a probing strategy to determine if lookahead has to be applied at the current branch. With probability p, where p is a parameter intuitively fixed to 0.01, lookahead is applied for probing purpose. The mean $avgp$ and the standard deviation $devp$ of the number of detected disjoint local cores in a successful probing lookahead are computed (not shown here due to the lack of space) to select the branches where lookahead is applied.

Inspired by the 68-95-99.7 rule in statistics, which says that the values within one (two, three) standard deviation of the mean account for about 68% (95%, 99.7%) of a normal data set, we reasonably assume that the number of cores detected in a successful lookahead is probably lower than $avgp + coef * devp$ when $coef = 3$. So, lookahead is not applied at the current branch when $k > avgp + coef * devp$. However, since the probing may not get exact information and the values may not follow a perfect normal distribution, $coef$ is dynamically adjusted to maintain the success rate of lookahead between $lowRate$ and $highRate$, where $lowRate$ and $highRate$ are parameters intuitively fixed to 0.6 and 0.75, respectively.

Concretely, $coef$ is initialized to 2 for each UB. At each probing, if the success rate of lookahead since the last probing is greater than $highRate$, it is increased by 0.1; and if it is lower than $lowRate$, it is decreased by 0.1. There are no lower and upper bounds on $coef$. When it is too low so that no lookahead is performed between two probings, it is reset to 2.

4.4 Soft literal ordering in lookahead

Existing BnB MaxSAT solvers such as MaxSatz usually propagate soft unit clauses in the ordering these clauses become unit, or in their ordering in the input formula when detecting disjoint cores [28]. MaxCDCL propagates first the soft literals in the cores detected in the previous lookahead. We use this ordering for two intuitive reasons.

- Before backtracking, a local core detected in the previous lookahead remains to be a core. Re-detecting a previous local core allows to obtain a possibly smaller local core due to additional assignments.
- After backtracking, re-detecting local cores in the previous soft conflict may allow to detect a new soft conflict sharing many local cores with the previous conflict. In this way, the clauses learnt from consecutive soft conflicts allow to intensify the search, because the clause learnt from a soft conflict is derived from the reasons of the detected cores.

The quality of a clause learnt from a soft conflict highly depends on the detected cores, which in turn highly depends on the soft literal ordering. How to improve the quality of the learnt clause by further improving the soft literal ordering definitely deserves future study.

4.5 Improving the VSIDS heuristic by lookahead

When there is no unit clause, a CDCL SAT solver chooses a free literal l using a decision heuristic, satisfies l and then performs unit propagation. VSIDS [38] is one of the widely used decision heuristics: it initializes the score of each variable to 0, and then at each conflict, it increases the score of each variable in a path to the conflict in the implication graph by var_inc, where var_inc is initialized to 1 and, after each conflict, it is divided by a parameter usually set to 0.95 to give more importance to the next conflict. Note that VSIDS is a heuristic based on lookback, because it is based on the conflicts in the past.

VSIDS is also used in MaxCDCL by increasing the score of a variable in a soft or hard conflict as in a CDCL SAT solver, but is modified as follows by taking lookahead into account. Every time the lookahead procedure detects a local core, the score of the variables encountered when identifying the reason of the core (see Subsection 4.2) is increased by $var\_inc \times \gamma$, where γ is a discount-rate parameter as in reinforcement learning, and is empirically fixed to 0.1. The intuition of this modification is the following. A soft conflict is derived when UB local cores are detected. So a variable contributing to many local cores should be favoured to reach a soft conflict as early as possible. However, a local core represents only a component of a future possible soft conflict but not a soft conflict for sure. So the increase of the score of a variable contributing to a core should be discounted by γ.

4.6 Implementation of MaxCDCL

Since MaxCDCL can be considered as an extension of a CDCL SAT solver, it is implemented on top of the CDCL SAT solver MapleCOMSPS_LRB [32], winner of the application track of SAT competition 2016. We chose MapleCOMSPS_LRB because it was one of the best SAT solvers with deterministic behavior when we started this work in 2017. Nevertheless, some of the recent advances in SAT solving are incorporated, including[1]:

- The approach from [31, 34] is applied to minimize the learnt clauses.
- The clause size reduction with the all-UIP learning technique from [18] is applied to reduce the clauses learnt from soft conflicts. This technique is particularly useful for MaxCDCL, because a clause learnt from a soft conflict is usually longer than a clause learnt from a hard conflict. The all-UIP technique allows to significantly reduce the size of a clause learnt from a soft conflict.
- The learnt clause management technique proposed in [22] is incorporated into MaxCDCL, allowing more learnt clauses to be kept in the clause database.

In addition, let $S = \{y_1, \ldots, y_{|S|}\}$ be the set of all soft literals. When $|S| \times (\text{UB}-1) \leq 10^4$, the sequential SAT encoding [42] of the cardinality constraint $\neg y_1 + \neg y_2 + \cdots + \neg y_{|S|} < \text{UB}$ is added to the input instance before starting the search. MaxCDCL alternates LRB phases and VSIDS phases for its search as MapleCOMSPS_LRB, using the LRB heuristic and the VSIDS heuristic modified as in subsection 4.5, respectively. Each phase is limited to a number of unit propagations specified by the parameter *phaseLength*, which is initialized to 2×10^7 and is doubled every cycle of LRB phase and VSIDS phase. In the VSIDS phase, the glucose restart strategy [7] is used; in the LRB phase, the Luby restart strategy [33] is used.

We plan to implement MaxCDCL on top of Kissat [13], the winner of the SAT2020 competition, which might further improve its performance.

5 Experimental Evaluation

We report on an experimental investigation to assess the performance of MaxCDCL. We ran all experiments with Intel Xeon CPUs E5-2680@2.40GHz under Linux with 32GB of memory, using the following benchmark sets, unless otherwise stated:

MSE19∪20: The union of all the instances used in the complete unweighted track of the MaxSAT Evaluations (MSE) 2019 and 2020, *a total of 1000 distinct instances.*
MC: A subset of the Master Collection of instances from the MaxSAT evaluations held until 2019 [2]. It contains 16080 unweighted (partial) MaxSAT instances, classified into 51 families and 76 subfamilies. MC includes all the instances of the 63 subfamilies having 100 instances or less, and the first 100 instances as they occur in the natural order in each of the remaining 13 subfamilies, *considering a total of 3614 instances.* This selection provides a simple, deterministic and objective criterion that does not favor any solver; and the experiments can be easily reproduced. MC contains 726 instances that also belong to MSE19∪20.

The cutoff time is one hour (3600s) per instance as in the MaxSAT Evaluation. For MaxCDCL and its variants, this includes 60 seconds to find a feasible upper bound UB_f with SatLike (version 3.0). Note that MaxCDCL and its variants do not start the search

[1] The source code of MaxCDCL is available at `https://home.mis.u-picardie.fr/~cli/EnglishPage.html`
[2] `https://www.cs.toronto.edu/maxsat-lib/maxsat-instances/master-set/unweighted/`

Table 1 Comparison of MaxCDCL with its variants for MSE19∪20 (left) and MC (right).

	#solv	avg	#solv	avg
MaxCDCL\LA	505	255s	2183	194s
MaxCDCL\harden	664	281s	2878	194s
MaxCDCLalwaysLA	681	249s	2962	193s
MaxCDCLioLA	704	268s	2963	168s
MaxCDCL\VSIDSbyLA	724	268s	3003	165s
MaxCDCL	734	256s	3022	156s

from UB=UB_f, but from UB=2^0, then $2^1, 2^2, \ldots$, until a feasible solution is found or $2^i > \text{UB}_f$. In the latter case, UB is set to $\text{UB}_f + 1$. Then, UB is gradually decreased until no better solution can be found (see Section 4.1).

The experiments are presented as follows. Firstly, we analyse the impact of the components implemented in MaxCDCL. Secondly, we compare the performance of MaxCDCL with that of the top 5 solvers in MSE2020. Thirdly, we show the complementarity of MaxCDCL with the top 5 solvers by comparing the number of instances solved using a portfolio solver with and without MaxCDCL. Finally, we compare MaxCDCL with a state-of-the-art BnB solver.

5.1 MaxCDCL Components

Table 1 compares MaxCDCL with the following variants:

MaxCDCL\LA. MaxCDCL without lookahead, i.e. the condition to lookahead is never satisfied in line 3 of Algorithm 2. Note that after finding a feasible UB, MaxCDCL\LA performs linear SAT-UNSAT search as a model-guided SAT-based MaxSAT solver.

MaxCDCL\harden. MaxCDCL without hardening (i.e., lines 17 and 18 in Algorithm 1 are removed).

MaxCDCLalwaysLA. MaxCDCL that looks ahead at every branch, i.e., the condition to lookahead is always satisfied in line 3 of Algorithm 2.

MaxCDCLioLA. MaxCDCL that, when detecting cores in lookahead, always propagates the soft literals in the ordering as the corresponding soft clauses occur in the input instance.

MaxCDCL\VSIDSbyLA. MaxCDCL that does not increase the VSIDS score of the variables contributing to a local core detected in lookahead as described in Subsection 4.5.

In Table 1, columns "#solv" give the number of solved instances and columns "avg" give the mean time in seconds (including the 60s used by SatLike) needed to solve these instances. These results indicate that a careful configuration combining clause learning and BnB is crucial for the performance of MaxCDCL, including: hardening based on local core detection and clause learning, the selective and adaptive application of lookahead and the ordering to propagate the soft literals when detecting local cores, because the absence of any of these components makes a significant number of instances out of reach of MaxCDCL. Without this configuration, MaxCDCL\LA solves 229 instances less than MaxCDCL in MES19∪20 and 839 instances less than MaxCDCL in MC. Note that although the hardening of a soft literal requires a distinct learnt clause, it does not increase the total memory usage, because it allows to avoid many learnt clauses by reducing search space.

Recall that the most fundamental feature of MaxCDCL is the clause learning from soft conflicts. We computed the average length of a clause learnt from a soft (hard) conflict for each solved instance by MaxCDCL in MSE19∪20, and found that the median average length

Table 2 Results for MSE19∪20 (left) and MC (right) with top 5 solvers.

	#solv	avg	#uniq	#win	#solv	avg	#uniq	#win
MaxHS	769	177s	11	36	3037	85.5s	26	116
EvalMaxSAT	759	129s	1	43	3002	69.7s	4	147
UWrMaxSAT	745	128s	3	42	2969	51.6s	7	141
RC2-B	728	164s	0	62	2948	70.1s	1	173
Open-WBO	695	157s	3	71	2906	89.7s	4	190
MaxCDCL	734	256s	**16**	–	3022	156s	**67**	–

of a clause learnt from a soft (hard) conflict is 23.67 (19.2) among the 734 instances solved in the set. Note that the learnt clause length averaged across the 734 instances does not make sense because it is biased by few instances with very long learnt clause length.

In addition, the comparison of MaxCDCL with MaxCDCL\VSIDSbyLA suggests that the VSIDS heuristic might be improved by lookahead, indicating a promising research direction.

To complete the subsection, we mention the impact of two other components of MaxCDCL: (1) the local search by SatLike in preprocessing to compute a feasible UB allows MaxCDCL to solve 8 more instances in MSE19∪20; (2) the sequential cardinality constraint encoding is applied to about the 20% of the instances in MSE19∪20 for at least one UB, helping solve 39 extra instances in MSE from highly symmetric problems such as drmx-at-most-k.

5.2 Comparison with the top 5 Solvers in MSE2020

A total of 15 solvers competed in the complete unweighted track of MSE2020 [9]. We consider the top 5 solvers: *MaxHS* (*mhs* in short), which is MHS-guided; *EvalMaxSAT* (*eval* in short), *RC2-B* (*rc2* in short) and *open-wbo-res-mergesat-v2* (*Open-WBO* or *owbo* in short), which are core-guided; and *UWrMaxSAT* (*uwr* in short), which combines both core-guided and model-guided solving. We executed the versions used in MSE2020 in all the experiments.

Table 2 shows the results for MSE19∪20 and MC, respectively. Column "#uniq" has the number of instances that were only solved by the solver in the row. Column "#win" has the number of instances solved by MaxCDCL but not by the solver in the row.

We observe that MaxCDCL solves more instances than two top 5 solvers in MSE19∪20 and four top 5 solvers in MC. More importantly, MaxCDCL solves a significant number of instances that other solvers cannot solve. For example, MaxCDCL solves 116 instances in MC that MaxHS does not solve. If we consider all the solvers together, there is also a significant number of instances solved by MaxCDCL that no other solver is able to solve: 16 instances in MSE19∪20 and 67 instances in MC that mainly come from the subfamilies MaxClique, MaxCut and UAQ. These results show that the existing MaxSAT solvers, especially the model-guided and core-guided ones, are able to solve similar kinds of instances. Nevertheless, MaxCDCL has the potential to solve new kinds of instances that are not solvable with the current MaxSAT techniques. It is important to note that MaxCDCL is far from being as optimized as the other solvers, which are the result of a process of continuous improvements since more than ten years.

5.3 Combining MaxCDCL with existing solvers

Given two deterministic solvers X and Y and a time limit T to solve an NP-hard problem such as MaxSAT, the simplest way to try to solve more instances than X and Y alone within the time limit T is to combine X and Y by running X within the time limit $T/2$, and then Y from scratch within the remaining time $T/2$ if the instance is not solved by X.

Table 3 Results for MSE19∪20 (left) and MC (right). The entry in cell (X, Y) for $X \neq Y$ is the number of instances solved by running solver X for 1800 seconds, and then solver Y from scratch for 1800 seconds if the instance is not solved by X. The entry in cell (X, X) (in the diagonal in grey) is the number of instances solved by running solver X for 1800 seconds. Column X in the last row recalls the results of solver X with 3600 seconds. The best results are in bold.

	mhs	eval	uwr	rc2	owbo	mcdcl	mhs	eval	uwr	rc2	owbo	mcdcl
mhs	747	777	777	770	763	**785**	3009	3068	3073	3056	3049	**3130**
eval	777	745	760	751	760	**786**	3068	2972	3019	2986	3013	**3126**
uwr	777	760	730	745	746	774	3073	3019	2951	3000	2998	**3098**
rc2	770	751	745	713	745	**778**	3056	2986	3000	2921	2981	**3105**
owbo	763	760	746	745	675	746	3049	3013	2998	2981	2865	**3076**
mcdcl	**785**	**786**	774	**778**	746	711	**3130**	**3126**	**3098**	**3105**	3076	2992
3600s	769	759	745	728	695	734	3037	3002	2969	2948	2906	3022

Table 3 shows the results of all possible pairwise combinations of the top 5 solvers and MaxCDCL (*mcdcl*) in MSE2020 for $T = 3600s$. Each cell (X, Y) for $X \neq Y$ contains the number of instances solved by running solver X for 1800s and then solver Y from scratch for 1800s in MSE19∪20 (left) and MC (right). Each cell (X, X) (in the diagonal in grey) contains the number of instances solved by X in 1800s. Column X in the last row recalls the results of X with 3600 seconds. Combining any of the top 5 solvers with MaxCDCL solves more instances than this solver and MaxCDCL alone within 3600s, while this is not always true when combining two top 5 solvers. For example, combining MaxHS and Open-WBO solves 763 instances in MSE19∪20 within 3600s, while MaxHS alone solves 769 instances within 3600s. This shows that MaxCDCL is more complementary with the top 5 solvers than other solvers.

More importantly, MaxCDCL combined with the top 2 solvers, MaxHS and EvalMaxSAT, solves the highest numbers (785 and 786) of instances in MSE19∪20. This result is significantly better than that of MaxHS or EvalMaxSAT alone, and the best combination without MaxCDCL solves only 777 instances. The results are even more striking in MC, where the worst combination of MaxCDCL with a top 5 solver is better than any other combination not including MaxCDCL, and combining MaxHS and MaxCDCL gives the best results, solving 93 instances more than the previous best result achieved by MaxHS alone, and 57 instances more than the best combination without MaxCDCL.

Figure 3 shows cactus plots comparing the best two combinations of solvers with MaxCDCL, the best two combinations without MaxCDCL, as well as the best two mono-solvers, for MSE19∪20 (left) and MC (right). Other solvers and combinations are excluded for readability reasons. For any time T ($0 < T \leq 3600s$), a curve gives the number of instances solved by a mono-solver or a combination of two solvers within T, where the number of instances solved by a combination X-Y of solvers X and Y within T is the number of instances solved by running X for $T/2$, and then Y for $T/2$. The solving time of a combination X-Y of solvers X and Y for an instance is twice the minimum solving time among X and Y. This simulates a parallel execution of X and Y by alternating them in small time periods. The plots clearly show the advantage of combining an existing solver with MaxCDCL, allowing to solve the highest number of instances within 3600s, and the advantage becomes greater as the running time is increased.

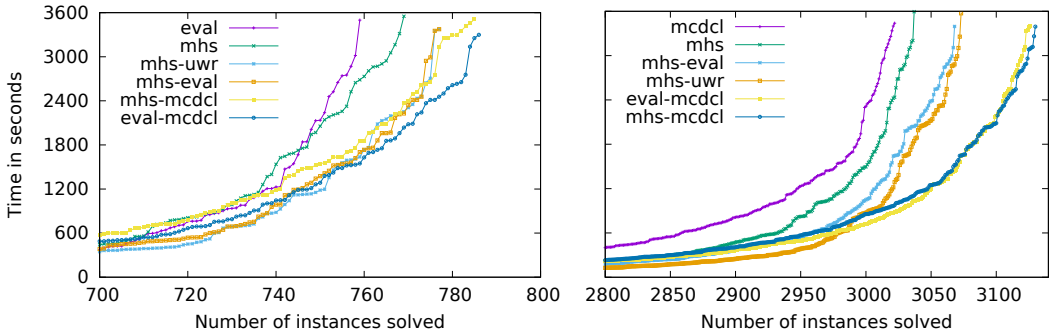

Figure 3 Cactus plots of the best two combinations with MaxCDCL (*mcdcl* in short), the best two without MaxCDCL, and the best two mono-solvers for MSE19∪20 (left) and MC (right). For each point (N, T) in a curve for a mono-solver X, N is the number of instances solved by X in T seconds; and for each point (N, T) in a curve for a combination X-Y of solvers X and Y, N is the number of instances solved by running X for $T/2$ seconds followed by running Y for $T/2$ seconds. The names of the solvers and combinations are listed in the order of the highest points of the corresponding curves from left to right for readability.

Table 4 Results for MSE16 and MSE19∪20.

			ahmaxsat		MaxCDCL		
		#ins	#solv	avg	#solv	avg	#win
MSE16	ms_ran	454	259	744s	27	1535s	0
	ms_craf	402	230	33.0s	47	324s	0
	ms_in	55	0	-	37	413s	37
	pms_ran	210	209	148s	141	772s	0
	pms_craf	678	369	242s	645	126s	276
	pms_in	601	183	515s	512	178s	330
MSE19∪20		1000	270	422s	734	256s	481

5.4 Comparison with a BnB Solver

We compare MaxCDCL with *ahmaxsat*, which was the best BnB solver in the last MaxSAT evaluation (MSE2016) in which BnB solvers competed. We use the MSE2016 instances of the categories MaxSAT (*ms*) and partial MaxSAT (*pms*). Each category has *random* (ran), *crafted* (craf), and *industrial* (in) instances. Thus, we consider a total of 6 families.

The results are shown in Table 4, where the number of instances for each family is given in column "#ins". We observe that ahmaxsat solves more random and non-partial crafted instances. A learnt clause for these instances with randomness and limited size usually contains most of the variables of the instances and is hardly useful. So, the higher use of lower bounding methods and the lack of clause learning in ahmaxsat are adequate for them, and the higher use of lower bounding methods (like in MaxCDCLalwaysLA) does not improve MaxCDCL for them because of clause learning. However, ahmaxsat has poor performance on other instances. Instead, MaxCDCL solves a much higher number of such instances.

6 Conclusion

We described MaxCDCL, a MaxSAT solver that combines, for the first time to the best of our knowledge, branch and bound and clause learning. The main differences of MaxCDCL with existing SAT-based MaxSAT solvers are the following:

- SAT-based MaxSAT solvers use a CDCL SAT solver as a black box and do not interfere in the internal operations of the SAT solver when solving an instance, while MaxCDCL itself can be considered a SAT solver extended to handle soft conflicts.
- Both MaxCDCL and model-guided MaxSAT solvers have an upper bound UB−1 of the number of soft clauses that can be falsified. The difference lies in how to exploit this UB. Let $falseS$ denote the set of already falsified soft clauses. On the one hand, model-guided MaxSAT solvers call a SAT solver after encoding UB into CNF. If no hard clause is falsified, the SAT solver backtracks only after $|falseS| \geq$ UB, because no clause encoding UB is falsified if $|falseS| <$ UB. On the other hand, MaxCDCL computes a lower bound LB of the number of soft clauses that will be falsified (but not yet falsified), and backtracks as soon as LB+$|falseS| \geq$ UB. Thus, MaxCDCL is able to backtrack much earlier than model-guided MaxSAT solvers.
- MaxCDCL, core-guided or MHS-guided MaxSAT solvers all identify cores. However, a core-guided or MHS-guided MaxSAT solver only identifies *global cores* (i.e., the cores that do not depend on any partial assignment) in order to relax them, while MaxCDCL detects *local cores* by using UP under a partial assignment to derive a soft conflict for learning a clause and backtracking early. Note that identifying a global core is NP-hard, while detecting a local core by applying UP is polynomial.

The extensive experimentation conducted shows that MaxCDCL is ranked among the top 5 exact MaxSAT solvers in the 2020 MaxSAT evaluation. Furthermore, it solves a significant number of instances that other solvers cannot solve, suggesting that combining branch and bound and clause learning has the potential to solve new kinds of instances that are not solvable with current MaxSAT techniques. More importantly, combining MaxCDCL with the existing solvers allows to solve the highest number of MaxSAT instances.

Detailed analyses indicate that the performance of MaxCDCL comes from a careful configuration combining clause learning and BnB, including hardening based on local core detection and clause learning, the selective and adaptive application of lookahead, and the ordering to propagate the soft literals when detecting local cores.

We believe that the proposed approach opens new and promising research directions, including for example: (1) improving the quality of the clauses learnt from soft conflicts by designing new soft literal orderings in lookahead; (2) exploiting the relationship of SAT and MaxSAT for improving SAT and MaxSAT solving; (3) adapting the approach of MaxCDCL to other problems such as pseudo-Boolean optimization and Max-CSP; (4) extending MaxCDCL to weighted MaxSAT, in which each soft clause is weighted. In the extension of MaxCDCL to weighted MaxSAT, each soft clause is represented by a weighted soft literal. A local core detected by lookahead is weighted by the minimum soft literal weight it contains, and other weights are split to be used in other core detections. When the total weight of all detected local cores reaches UB, a soft conflict is discovered. The challenge here is that there can be more local cores to detect than in the unweighted case. Thus, the clause learnt from a soft conflict can be longer. Special strategies in clause and soft literal ordering should be designed to learn shorter clauses from soft conflicts.

References

1 André Abramé and Djamal Habet. Ahmaxsat: Description and evaluation of a branch and bound Max-SAT solver. *J. Satisf. Boolean Model. Comput.*, 9:89–128, 2014.

2 André Abramé and Djamal Habet. Learning nobetter clauses in Max-SAT branch and bound solvers. In *Proceedings of ICTAI 2016*, pages 452–459, 2016.

3 Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. Solving (Weighted) Partial MaxSAT through satisfiability testing. In *Proceedings of SAT 2009*, pages 427–440. Springer LNCS 5584, 2009.

4 Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial MaxSAT. In *Proceedings AAAI 2010*, pages 3–8, 2010.

5 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.

6 Carlos Ansótegui and Joel Gabàs. WPM3: An (in)complete algorithm for Weighted Partial MaxSAT. *Artificial Intelligence*, 250:37–57, 2017.

7 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings IJCAI 2009*, pages 399–404, 2009.

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

9 Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation 2020: Solver and benchmark descriptions, 2020.

10 Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in MaxSAT. In *Proceedings of CP 2017*, Springer LNCS, pages 641–651, 2017.

11 Fahiem Bacchus, Matti Järvisalo, and Martins Ruben. Maximum satisfiability. In *Handbook of satisfiability, second edition*, pages 929–991. IOS Press, 2021.

12 Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *J. Satisf. Boolean Model. Comput.*, 7(2-3):59–6, 2010.

13 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximilian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*, page 50, 2020.

14 Mohamed Sami Cherif, Djamal Habet, and André Abramé. Understanding the power of Max-SAT resolution through UP-resilience. *Artificial Intelligence*, 289:103397, 2020.

15 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proceedings of CP 2011*, page 225–239. Springer, 2011.

16 Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MAXSAT. In *Proceedings of SAT 2013*, pages 166–181. Springer, 2013.

17 Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-boolean conflict-driven search. *Constraints*, pages 1–30, 2021.

18 Nick Feng and Fahiem Bacchus. Clause size reduction with all-uip learning. In *Proceedings of SAT 2020, Springer LNCS 12178*, pages 28–45, 2020.

19 Graeme Gange, Jeremias Berg, Emir Demirović, and Peter J Stuckey. Core-guided and core-boosted search for CP. In *Proceedings of CPAIOR 2020*, pages 205–221, 2020.

20 Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSAT: An efficient Weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

21 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019.

22 Stepan Kochemazov. Improving implementation of SAT competitions 2017–2019 winners. In *Proceedings of SAT 2020, LNCS 12178*, pages 139–148, 2020.

23 Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A Partial Max-SAT solver. *J. Satisf. Boolean Model. Comput.*, 8(1/2):95–100, 2012.

24 Adrian Kuegel. Improved exact solver for the Weighted MAX-SAT problem. In *Proceedings of Workshop Pragmatics of SAT, POS-10, Edinburgh, UK*, pages 15–27, 2010.

25 Zhendong Lei and Shaowei Cai. Solving (Weighted) Partial MaxSAT by dynamic local search for SAT. In *Proceedings of IJCAI 2018*, pages 1346–1352, 2018.

26 Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In *Handbook of satisfiability, second edition*, pages 903–927. IOS Press, 2021.

27 Chu Min Li, Felip Manya, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in MaxSAT. *Constraints*, 15(4):456–484, 2010.

28 Chu Min Li, Felip Manya, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of CP 2005*, pages 403–414. Springer, 2005.

29 Chu Min Li, Felip Manya, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *Proceedings of AAAI 2006*, pages 86–91, 2006.

30 Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.

31 Chu-Min Li, Fan Xiao, Mao Luo, Felip Manyà, Zhipeng Lü, and Yu Li. Clause vivification by unit propagation in CDCL SAT solvers. *Artificial Intelligence*, 279, 2020.

32 Jia Hui Liang, Chanseok Oh, Vijay Ganesh, Krzysztof Czarnecki, and Pascal Poupart. MapleCOMSPS, MapleCOMSPS LRB, MapleCOMSPS CHB. In *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*, pages 52–53, 2016.

33 Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

34 Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü. An effective learnt clause minimization approach for CDCL SAT solvers. In *Proceedings of IJCAI 2017*, pages 703–711, 2017.

35 Vasco M. Manquinho, Joao Marques-Silva, and Jordi Planes. Algorithms for weighted Boolean optimization. In *Proceedings of SAT 2009*, pages 495–508. Springer LNCS 5584, 2009.

36 Joao Marques-Silva and Vasco M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *Proceedings of SAT 2008*, pages 225–230. Springer LNCS 4996, 2008.

37 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of SAT 2014*, volume 8561 of *LNCS*, pages 438–445. Springer, 2014.

38 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC 2001*, pages 530–535. ACM, 2001.

39 Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Proceedings of AAAI 2014*, pages 2717–2723, 2014.

40 Tobias Paxian and Bernd Becker. Pacose: An iterative SAT-based MaxSAT solver. In *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, page 12, 2020.

41 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In *Proceedings of SAT 2016*, volume 9710 of *LNCS*, pages 539–546, 2016.

42 Carsten Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *Proceedings of CP 2005*, pages 827–831. Springer LNCS 3709, 2005.

43 Fulya Trösser, Simon De Givry, and George Katsirelos. Relaxation-aware heuristics for exact optimization in graphical models. In *Prodeedings of CPAIOR 2020*, pages 475–491. Springer, 2020.

44 Aolong Zha. QMaxSAT in MaxSAT Evaluation 2018. In *Proceedings of the MaxSAT Evaluation 2020*, page 16, 2020.