Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)
Sister Conferences Best Papers Track

Combining Clause Learning and Branch and Bound for MaxSAT (Extended Abstract)*

Chu-Min Li[1,2,3], **Zhenxing Xu**[1], **Jordi Coll**[3], **Felip Manyà**[4], **Djamal Habet**[3] and **Kun He**[1]
[1]Huazhong University of Science and Technology, Wuhan, China
[2]Université de Picardie Jules Verne, Amiens, France
[3]Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
[4]Artificial Intelligence Research Institute (IIIA), CSIC, Bellaterra, Spain
chu-min.li@u-picardie.fr, lxadd515@hust.edu.cn, jordi.coll@lis-lab.fr, felip@iiia.csic.es,
Djamal.Habet@univ-amu.fr, brooklet60@hust.edu.cn

Abstract

Branch and Bound (BnB) has been successfully used to solve many combinatorial optimization problems. However, BnB MaxSAT solvers perform poorly when solving real-world and academic optimization problems. They are only competitive for random and some crafted instances. Thus, it is a prevailing opinion in the community that BnB is not really useful for practical MaxSAT solving. We refute this opinion by presenting a new BnB MaxSAT solver, called MaxCDCL, which combines clause learning and an efficient bounding procedure. MaxCDCL is among the top 5 out of a total of 15 exact solvers that participated in the 2020 MaxSAT Evaluation, solving several instances that other solvers cannot solve. Furthermore, MaxCDCL solves the highest number of instances from different MaxSAT Evaluations when combined with the best existing solvers.

1 Introduction

The Maximum satisfiability problem (MaxSAT) is an optimization version of the satisfiability problem (SAT). The goal of MaxSAT is to find an assignment that satisfies the maximum number of clauses in a given multiset. If we distinguish between *hard* and *soft* clauses, we have Partial MaxSAT and its goal is to satisfy all the hard clauses and the maximum number of soft clauses. This paper considers both MaxSAT and Partial MaxSAT.

SAT and MaxSAT share many features, but in practice solving MaxSAT is much harder than solving SAT. Indeed, since several clauses can be falsified in a MaxSAT solution, some fundamental SAT techniques such as unit propagation cannot be used in MaxSAT as they are used in SAT. Despite this difficulty, thanks to the huge efforts of the scientific community, it is currently possible to solve many challenging real-world and academic NP-hard optimization problems encoded as MaxSAT instances. For this reason, MaxSAT has received a growing interest from academy and industry in recent years.

We distinguish two types of exact MaxSAT algorithms: Branch-and-Bound (BnB) algorithms [Li and Manyà, 2021], which directly tackle MaxSAT with a bounding procedure but without unit propagation and clause learning; and SAT-based algorithms [Bacchus *et al.*, 2021], which transform MaxSAT into a sequence of SAT instances that are solved with a Conflict-Driven Clause Learning (CDCL) SAT solver.

SAT-based MaxSAT solvers are usually much better than BnB MaxSAT solvers in solving many real-world 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 needs to backtrack when it computes a lower bound equal to the upper bound [Li *et al.*, 2007]. 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.

This paper proposes 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. The experimental results show that MaxCDCL is among the top 5 out of a total of 15 exact solvers in the 2020 MaxSAT Evaluation, solving several instances that other solvers cannot solve. Furthermore, MaxCDCL, when combined with the best existing solvers, solves the highest number of instances from different MaxSAT Evaluations. More importantly, our results refute the prevailing opinion stating that, although BnB is a powerful technique in combinatorial optimization, it is not really useful for MaxSAT.

This paper is an extended abstract of [Li *et al.*, 2021]. The original paper contains additional experiments and a more detailed description of MaxCDCL, including techniques and implementation details that have a high impact on the perfor-

*This is an extended abstract of a paper that won the Best Paper Award at the 27th International Conference on Principles and Practice of Constraint Programming (CP 2021).

mance of the solver.

2 MaxCDCL: A BnB Algorithm Using CDCL for MaxSAT

This section describes the general structure of MaxCDCL and the two most relevant techniques of MaxCDCL: the combination of BnB and clause learning, and the new bounding procedure.

We first review some basic concepts: A Partial MaxSAT instance ϕ is formed by a multiset of hard and soft clauses, where a clause is a disjunction of literals and a literal is a propositional variable x or its negation $\neg x$. An assignment for ϕ assigns either true (1) or false (0) to each variable of ϕ, satisfies a clause if some literal is true, and falsifies a clause if all its literals are false. A clause is unit if all literals are false but one is unassigned. Partial MaxSAT is to find an assignment that satisfies all the hard clauses and the maximum number of soft clauses. SAT is Partial MaxSAT without soft clauses and (unweighted) MaxSAT is Partial MaxSAT without hard clauses.

2.1 General Structure of MaxCDCL

MaxCDCL represents the input partial MaxSAT instance using two sets: H and S. Set H contains the input hard clauses and set S contains a new literal y for each input soft clause sc after adding the hard clauses encoding $y \leftrightarrow sc$ to H. We say that y is a *soft literal* because it is satisfied if and only if the corresponding soft clause is satisfied. The *cost* of a solution is the number of falsified soft clauses and is equal to $|falseS|$, where $falseS$ denotes the set of falsified soft literals.

MaxCDCL is a generalization of the CDCL SAT algorithm. Roughly speaking, a CDCL SAT solver explores a search tree by alternating a search phase, where literals are assigned until either a solution or a conflict is found, and a learning phase that is executed after finding a conflict to learn a new clause. *Unit Propagation* (UP) is the main inference rule applied during the search: if there is a unit clause in H, its unassigned literal l must be set to 1. UP is applied during the search until a clause is falsified (a *conflict* is found) or no more unit clauses remain in H. If UP terminates 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. A solution is found when all the variables are assigned without finding a conflict.

Following the BnB procedure, MaxCDCL tries to find a solution whose cost is smaller than a given upper bound UB. When a solution is found, the UB is decreased by taking as value the cost of this solution. During the search, MaxCDCL distinguishes between hard and soft conflicts. A *hard* conflict occurs when the current partial assignment falsifies a hard clause. A *soft* conflict occurs when the current partial assignment cannot be extended to a complete one falsifying fewer than UB soft clauses. MaxCDCL extends the CDCL SAT solver by also learning a hard clause from each discovered soft conflict.

Concretely, when UP does not falsify any hard clause, it calls, under certain conditions, a lookahead (LA) procedure to compute a lower bound LB 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 LB$\geq$UB, it is analyzed to learn a clause for backtracking. Moreover, if no soft conflict is discovered, but $LB = UB-1$, MaxCDCL satisfies every free soft literal y not involved in the computation of LB by applying a procedure called *hardening*.

2.2 Combining Lookahead and Clause Learning

A subset of soft literals $S_i = \{y_1, \ldots, y_{|S_i|}\}$, where $|S_i| \geq 1$, is inconsistent if they cannot be simultaneously satisfied. This inconsistency can be represented by the hard clause $\neg y_1 \vee \cdots \vee \neg y_{|S_i|}$. 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*.

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 1 provides the foundation of our approach in the general case.

Proposition 1. *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 $\neg l_{11} \vee \cdots \vee \neg l_{1|L_1|} \vee \cdots \vee \neg l_{k1} \vee \cdots \vee \neg l_{k|L_k|}$.*

Given a partial assignment F of H, the application of Proposition 1 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. 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 an implicit clause explaining the soft conflict. This clause can be further analyzed with standard SAT conflict analysis techniques to learn a clause to be explicitly added to H to prevent the same soft conflict in the future, exactly 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 [Li *et al.*, 2009; Li *et al.*, 2005; Li *et al.*, 2006]. 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 [Li *et al.*, 2010]. This advantage is essential for our approach, because MaxCDCL needs to learn clauses of good quality from the detected local cores. The other existing BnB MaxSAT solvers [Abramé and Habet, 2014; Heras *et al.*, 2008; Kuegel, 2010; Li *et al.*, 2007] detect disjoint local cores, but instead of explaining them with a learnt clause they simply backtrack, which is very different from MaxCDCL.

2.3 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 0.01, lookahead is applied for probing purposes. The mean $avgp$ and the standard deviation $devp$ of the number of detected local cores in a successful probing lookahead are computed 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.

3 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:
MSE19∪20: The union of all the instances used in the complete unweighted track of the MaxSAT Evaluations (MSE) 2019 and 2020, *considering a total of 1000 distinct instances.*
MC: A subset of the Master Collection of instances from the MaxSAT evaluations held until 2019 [1], which contains 76 subfamilies of (partial) MaxSAT instances. 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.*

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) [Cai and Lei, 2020].

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. Finally, 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.

[1] www.cs.toronto.edu/maxsat-lib/maxsat-instances/master-set/unweighted/

Solver	#solv	avg	#solv	avg
MaxCDCL\LA	505	255s	2183	194s
MaxCDCL\harden	664	281s	2878	194s
MaxCDCLalwaysLA	681	249s	2962	193s
MaxCDCL	734	256s	3022	156s

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

3.1 MaxCDCL Components

Table 1 compares MaxCDCL with the following variants:

MaxCDCL\LA. MaxCDCL without lookahead, i.e., soft conflicts are only detected when $|falseS|$=UB.

MaxCDCL\harden. MaxCDCL without hardening.

MaxCDCLalwaysLA. MaxCDCL with lookahead at every branch without using the probing strategy of Section 2.3.

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 the selective and adaptive application of lookahead. The absence of any of these components leaves a significant number of instances out of reach of MaxCDCL.

3.2 Comparison with Top 5 Solvers in MSE2020

A total of 15 solvers competed in the complete unweighted track of MSE2020 [Bacchus *et al.*, 2020]. We consider the top 5 solvers: *MaxHS* (*mhs* in short) [Bacchus, 2020], *EvalMaxSAT* (*eval* in short) [Avellaneda, 2020], *RC2-B* (*rc2* in short) [Ignatiev *et al.*, 2019], *open-wbo-res-mergesat-v2* (*Open-WBO* or *owbo* in short) [Martins *et al.*, 2014], and *UWrMaxSAT* (*uwr* in short) [Piotrów, 2020]. We executed the versions used in MSE2020 in all the experiments. Table 2 shows the results for MSE19∪20 and MC, respectively.

We observe that MaxCDCL solves more instances than two and four top 5 solvers in MSE19∪20 and MC, respectively. 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 can solve: 16 instances in MSE19∪20 and 67 instances in MC. These results show that the existing MaxSAT solvers can solve similar classes of instances. However, 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 in a broad community.

3.3 Combining MaxCDCL with Existing Solvers

Given two deterministic solvers X and Y, the easiest way to try to solve more instances than X and Y alone within a

Solver	#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**	-

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

$X \setminus Y$	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: 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.

time limit T is to combine X and Y by running X within a 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 shows the results of all possible pairwise combinations of the top 5 solvers in MSE2020 and MaxCDCL (*mcdcl*) for $T = 3600s$.

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. 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 only solves 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 MaxHS alone, and 57 instances more than the best combination without MaxCDCL.

4 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 performance of MaxCDCL comes from a careful configuration combining clause learning and BnB, including hardening based on local core detection and the selective and adaptive application of lookahead.

Unlike SAT-based MaxSAT solvers, which use a CDCL SAT solver as a black box and do not interfere in the internal operations of the SAT solver, MaxCDCL itself can be considered a SAT solver extended to handle soft conflicts. Also, unlike model-guided SAT-based solvers, MaxCDCL computes a lower bound LB of the number of soft clauses that will be falsified (but are not yet falsified), and therefore it can backtrack much earlier. MaxCDCL, core-guided or MHS-guided MaxSAT solvers all identify cores. However, the latter only identify *global cores* (i.e., the cores that do not depend on any partial assignment), while MaxCDCL detects *local cores* by using UP under a partial assignment to derive a soft conflict for learning a clause and backtracking early.

MaxCDCL is ranked among the top 5 exact MaxSAT solvers in MSE2020. Furthermore, it solves a significant number of instances that other solvers cannot solve, suggesting that combining BnB and clause learning has the potential to solve new kinds of instances that current MaxSAT techniques cannot solve.

The proposed approach opens new and promising research directions, including the exploitation of the relationships between SAT and MaxSAT, the adaptation of the MaxCDCL approach to other problems such as pseudo-Boolean optimization and Max-CSP, or the extension of MaxCDCL to weighted MaxSAT.

Acknowledgments

This work has been partially funded by the French Agence Nationale de la Recherche, reference ANR-19-CHIA-0013-01, and project PID2019-111544GB-C21 funded by MCIN/AEI/10.13039/501100011033, and partially supported by Archimedes Institute, Aix-Marseille University. F. Manyà was supported by mobility grant PRX21/00488 of the *Ministerio de Universidades*. We thank the Université de Picardie Jules Verne for providing the Matrics Platform.

References

[Abramé and Habet, 2014] André Abramé and Djamal Habet. Ahmaxsat: Description and evaluation of a branch and bound Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:89–128, 2014.

[Avellaneda, 2020] Florent Avellaneda. A short description of the solver evalmaxsat. *MaxSAT Evaluation 2020*, page 8, 2020.

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

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

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

[Cai and Lei, 2020] Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020.

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

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

[Kuegel, 2010] 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.

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

[Li *et al.*, 2005] Chu Min Li, Felip Manyà, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of CP 2005*, volume 3709 of *LNCS*, pages 403–414. Springer, 2005.

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

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

[Li *et al.*, 2009] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Exploiting cycle structures in Max-SAT. In *In Proceedings of SAT 2009*, volume 5584 of *LNCS*, pages 467–480. Springer, 2009.

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

[Li *et al.*, 2021] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining clause learning and branch and bound for MaxSAT. In *Proceedings of CP 2021*, LIPIcs, pages 38:1–38:18, 2021.

[Martins *et al.*, 2014] 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.

[Piotrów, 2020] Marek Piotrów. UWrMaxSat: an efficient solver in MaxSAT evaluation 2020. *MaxSAT Evaluation 2020*, page 34, 2020.