A Proof Builder for Max-SAT

Matthieu Py, Mohamed Sami Cherif, Djamal Habet

A Proof Builder for Max-SAT

Matthieu Py, Mohamed Sami Cherif, and Djamal Habet

Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{matthieu.py,mohamed-sami.cherif,djamal.habet}@univ-amu.fr

Abstract. Complete Max-SAT solvers are able to return the optimal value of an input instance but they do not provide any certificate of its validity. In this paper, we introduce for the first time a Max-SAT proof builder, called MS-Builder, which generates Max-SAT proofs under the particular form of a sequence of Max-SAT equivalence-preserving transformations. To generate a Max-SAT proof, MS-Builder iteratively calls a SAT oracle to get a SAT refutation which is handled and adapted into a sound refutation for Max-SAT. We also propose an extendable tool, called MS-Checker, able to verify the validity of any proof using Max-SAT inference rules.

Keywords: Max-SAT · Proof · Max-SAT Resolution

1 Introduction

Given a Boolean formula in Conjunctive Normal Form (CNF), the Maximum Satisfiability (Max-SAT) problem consists in determining the maximum number of clauses that it is possible to satisfy by an assignment of the variables. Max-SAT is an optimization extension of the Satisfiability (SAT) problem and a natural way to model many real world and crafted problems [32,15,12] making it a well studied problem in theory as well as in practice. Different complete solving paradigms for Max-SAT have seen the day in recent years including Branch and Bound algorithms [1,23,18], SAT-based algorithms [2,26,27] and reduction to other problems (such as ILP [13], Max-ASP [3] and WCSP [14]).

Recent years have also witnessed a particular interest in proof systems for Max-SAT [10,11,19,20,21,9,28]. In particular, Max-SAT resolution [10,11,19] was one of the first known complete systems for Max-SAT and was later extensively used in the context of Max-SAT solving [23,1,27]. However, generating proofs establishing the optimum cost of Max-SAT formulas remains an unexplored topic in practice. Indeed, current Max-SAT solvers are not able to output certificates as it is the case for SAT solvers. This is in part due to the variety of paradigms and techniques for Max-SAT solving which make it difficult to devise a generalized approach to compute certificates.

In this paper, we devise an independent proof builder for Max-SAT, called MS-Builder, which builds proofs for Max-SAT by iteratively calling a SAT oracle to get a resolution refutation. The builder relies on recent work [28] to adapt the SAT refutation into a Max-SAT refutation which is then applied to the current

formula. Moreover, we introduce an extendable Max-SAT proof checker, called MS-Checker, to verify the validity of any proof using Max-SAT inference rules. Both tools are experimentally evaluated on the unweighted partial benchmark of the 2020 Max-SAT Evaluation [4].

This paper is organized as follows. Section 2 includes some necessary definitions and notations. Section 3 recalls related work on the adaptation of SAT refutations to Max-SAT refutations. MS-Builder and MS-Checker are respectively presented in Sections 4 and 5 and their experimental evaluation is detailed in Section 6. Finally, we conclude and discuss future work in Section 7.

2 Preliminaries

2.1 Definitions and Notations

Let X be a set of propositional variables. A literal l is a variable $x \in X$ or its negation $\overline{x}$. A clause c is a disjunction of literals $(l_1 \vee l_2 \vee \cdots \vee l_k)$. A formula ϕ in Conjunctive Normal Form (CNF) is a conjunction of clauses $\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$. An assignment $I : X \longrightarrow \{true, false\}$ maps each variable to a boolean value and can be represented as a set of literals. A literal l is satisfied (resp. falsified) by I if $l \in I$ (resp. $\overline{l} \in I$). A clause c is satisfied by I if at least one of its literals is satisfied by I, otherwise it is falsified by I. The empty clause $\square$ contains zero literals and is always falsified. A clause c opposes a clause c' if c contains a literal whose negation is in c', i.e. $\exists l \in c,\ \overline{l} \in c'$. For a given CNF formula, solving the Satisfiability (SAT) problem consists in determining whether there exists an assignment I (called model) that satisfies it. The cost of an assignment I is the number of clauses falsified by I. For a given CNF formula ϕ, solving the (plain) Max-SAT problem consists in determining the maximum number of satisfied clauses in ϕ.

2.2 Resolution Refutations in SAT

To certify that a CNF formula is satisfiable, it is sufficient to exhibit a model of the formula. On the other hand, to prove that a CNF formula is unsatisfiable, we need to refute the existence of a model. A well-known SAT refutation system is based on an inference rule for SAT called resolution [29]. The resolution rule, defined below, deduces a clause called resolvent which can be added to the formula from two opposed clauses.

Definition 1 (Resolution [29]). *Given two clauses* $c_1 = (x \vee A)$ *and* $c_2 = (\overline{x} \vee B)$*, the resolution rule is defined as follows:*

$$\frac{c_1 = (x \vee A)\ c_2 = (\overline{x} \vee B)}{c_3 = (A \vee B)}$$

A resolution refutation is a sequence of resolutions leading to an empty clause. Many restricted classes of resolution refutations have been studied in the literature namely linear resolution [24], unit resolution [16], input resolution [16],

regular resolution [31], read-once resolution [17] and tree (or tree-like) resolution refutations [5] among others. In particular, a resolution refutation is tree-like if every intermediate clause, i.e. resolvent, is used at most once in the proof. Similarly, a resolution refutation is read-once if each clause is used at most once in the proof. Clearly, read-once resolution refutations are also tree-like since they form a restricted class of tree resolution refutations. It was shown in [17] that there exists unsatisfiable CNF formulas which cannot be refuted using read-once resolution. Finally, a resolution is regular if each branch (path from a clause of the initial formula to the empty clause) contains at most one resolution per variable.

Example 1. We consider the CNF formula $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$. The resolution refutation of ϕ, represented in Figure 1, is tree-like (and) regular, but not read-once because of clause (x_1).

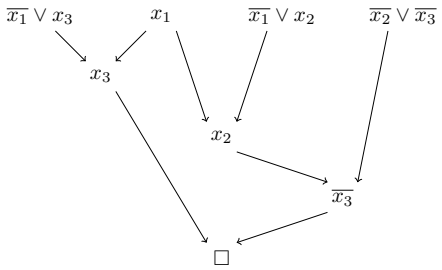

Fig. 1. Resolution refutation

2.3 Proofs for Max-SAT

In the last fifteen years, the study of inference rules for Max-SAT has led to major results in Max-SAT theory and solving. In particular, one of the first proof systems for Max-SAT is based on an inference rule called Max-SAT resolution, which is an extension of the resolution rule. Max-SAT resolution was shown sound and complete for Max-SAT, i.e. it is sufficient to prove the optimum cost of a given CNF formula.

Definition 2 (Max-SAT resolution [10,11,19]). *Given two clauses* $c_1 = x \vee a_1 \vee \cdots \vee a_s$ *and* $c_2 = \overline{x} \vee b_1 \vee \cdots \vee b_t$ *with* $A = a_1 \vee \cdots \vee a_s$ *and* $B = b_1 \vee \cdots \vee b_t$, *the Max-SAT resolution rule is defined as follows:*

$$\frac{c_1 = x \vee A \qquad c_2 = \overline{x} \vee B}{\begin{array}{c} c_3 = A \vee B \\ cc_1 = x \vee A \vee \overline{b_1} \\ \cdots \\ cc_t = x \vee A \vee b_1 \vee \cdots \vee b_{t-1} \vee \overline{b_t} \\ cc_{t+1} = \overline{x} \vee B \vee \overline{a_1} \\ \cdots \\ cc_{t+s} = \overline{x} \vee B \vee a_1 \vee \cdots \vee a_{s-1} \vee \overline{a_s} \end{array}}$$

where c_3 is the resolvent clause and $cc_1, \ldots, cc_{t+s}$ are compensation clauses.

In recent work, Max-SAT resolution was augmented with other rules such as the split rule [21,28] defined below or the extension rule [20]. It was shown that the addition of such rules to Max-SAT resolution can improve its efficiency in generating shorter proofs [21,20] or allow, given a resolution refutation for SAT, to generate a Max-SAT resolution refutation [28].

Definition 3 (Split rule). *Given a clause $c_1 = (A)$ where A is a disjunction of literals and x a variable, the split rule is defined as follows:*

$$\frac{c_1 = (A)}{c_2 = (x \vee A) \quad c_3 = (\overline{x} \vee A)}$$

Remark 1. Unlike the resolution rule, the Max-SAT resolution rule and the split rule replace the premise(s) by the conclusion(s).

To be more exhaustive, we must also mention that other Max-SAT proof systems exist like the Clause Tableau Calculus [22]. If these proofs systems have been extensively studied in theory, generating proofs remains an unexplored topic in practice. Hence, this work aims to contribute to this topic by proposing tools to build and check Max-SAT proofs.

3 Related work

In this section, we briefly recall recent results established in [28] on the adaptation of resolution refutations to Max-SAT refutations. One of the main results deals with tree resolution refutations showing that a linear adaptation to a Max-SAT refutation is possible in this case. Indeed, if the resolution refutation is tree-like, it is possible to transform it into a smaller refutation which is tree-like regular by iteratively eliminating irregularities (sequences of successive resolutions whose first and last are on the same variable) [30]. To adapt tree regular refutations, Max-SAT resolution is augmented with the split rule to fix the non-read-once clauses, i.e. clauses which are used more than once in the proof. The split rule is applied on a non-read-once clause to augment it with the variable resolved on in the junction point of all the branches starting from it. Thus, the obtained clauses can replace the non-read-once clause as a premise without affecting the validity of the proof. The same treatment is applied until all non-read-once clauses are fixed. Finally, when the proof becomes read-once, it is sufficient to replace each resolution step by a Max-SAT resolution to get a valid Max-SAT refutation [17]. An adaptation of tree regular resolution refutation is showcased in Example 2.

Example 2. We consider the tree regular resolution refutation in Example 1, represented in Figure 1. We observe that the clause (x_1) is used two times as a premise of a resolution step. The junction point of the left and right branches eliminates variable x_3. Thus, we apply the split rule on clause (x_1) to get $(x_1 \vee x_3)$

and $(x_1 \vee \overline{x_3})$ and we replace (x_1) by $(x_1 \vee x_3)$ and $(x_1 \vee \overline{x_3})$ respectively on the left and right branches. Finally, we replace all resolutions by Max-SAT resolutions to obtain the complete Max-SAT refutation represented in Figure 2.

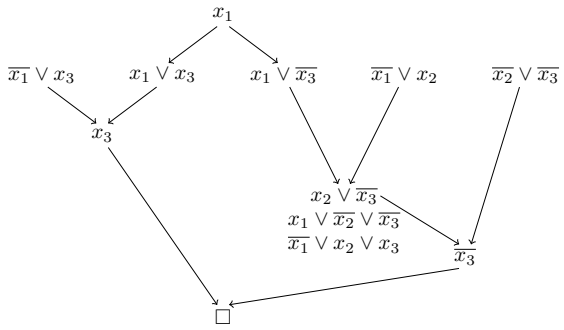

Fig. 2. Adaptation of a tree-like regular resolution refutation

In the generic case where the resolution refutation is not tree-like, it is also possible to adapt it into a Max-SAT refutation but with an exponential cost. To adapt the refutation, the entire proof leading to each non-read-once intermediate clause is duplicated, thus generating as many copies of the clauses as needed to render the proof tree-like. Then, the tree refutation is adapted into a Max-SAT refutation as explained above.

4 MS-Builder

In this section, we describe a Max-SAT proof builder, called MS-Builder based on the adaptation of resolution refutations for Max-SAT recalled in Section 3. The idea is to iteratively call a SAT oracle in order to get a resolution refutation for the current formula, adapt it into a Max-SAT refutation and apply it to the formula. The proof builder repeats this step until the SAT oracle returns a model for the final formula.

For practical reasons, we add an additional treatment to the Max-SAT proof builder. If the resolution refutation is not read-once, we first try to fix the effect of unit propagation in the non-read-once part of the proof. To do that, we discard the non-read-once unit clauses and we re-inject them at the end of the resolution refutation. Indeed, some resolution refutations are non-read-once simply because of the effects of unit propagation. It is the case of the resolution refutation proposed in Example 1, which is in fact read-once after fixing the unit propagation as showcased in the following example.

Example 3. We consider the tree regular resolution refutation in Example 1 (represented in Figure 1). We fix the unit propagation by discarding (x_1) and re-injecting it at the end of the resolution refutation to get the read-once resolution refutation represented in Figure 3.

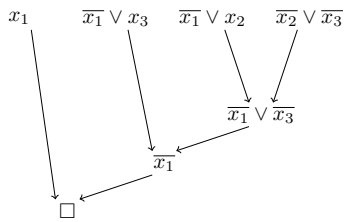

Fig. 3. Fixing unit propagation to get a read-once resolution refutation

The adaptation of any resolution refutation into a Max-SAT refutation is integrated into MS-Builder which can be seen as a core-based Max-SAT proof builder whose correctness is guaranteed by the correctness of the adaptation proposed in [28].

Example 4. Let $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$. The SAT oracle returns the resolution refutation in Figure 1 which is adapted into the read-once one in Figure 3 and to a Max-SAT one (by replacing resolutions by Max-SAT resolutions) and then applied to the formula. Now we have $\phi = \square \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$ and the SAT oracle returns that the current formula (without considering the empty clause) is satisfied by the assignment $I(x_1) = I(x_2) = I(x_3) = 0$. MS-Builder thus returns the proof described in Figure 5.

5 MS-Checker

In this section, we present our extendable Max-SAT checker, called MS-Checker, which requires two input files: a formula and a proof. The formula has to be given in the standard WCNF format, either in the old or new format [4]. The proof file must start with a sequence of Max-SAT transformation lines. A Max-SAT transformation line must start with 't' and must include the name of the inference rule (`msres` for Max-SAT resolution and `split` for the split rule) and its premise(s) (between '< >'). For the split rule, the variable to split on is specified as a parameter after its name. Then, the proof file must contain a line (starting with 'o') with the announced optimum cost of the formula. Finally, it must contain a line (starting with 'v') with a truth assignment satisfying the final formula (without the empty clauses).

```
c Formula of Example 1
1 -1 3 0
1 1 0
1 -1 2 0
1 -2 -3 0
```

Fig. 4. Formula file format

```
t msres < 1 -1 -2 | 1 -2 -3 >
t msres < 1 -1 3 | 1 -1 -3 >
t msres < 1 1 | 1 -1 >
o 1
v 000
```

Fig. 5. Proof file format

After reading the formula, MS-Checker verifies that the proposed inference rules are correct and that the premises are still in the formula then applies the transformation. Finally, it checks if the truth assignment satisfies the final formula without considering the empty clauses.

6 Experiments

We have implemented MS-Builder and MS-Checker in C++[1]. Resolution Refutations are computed using Booleforce [6] and Tracecheck [7,8]. We consider the benchmark of the unweighted partiel track of the 2020 Max-SAT Evaluation [4]. The experiments are performed on Dell PowerEdge M620 servers with Intel XeonSilver E5-2609 processors (clocked at 2.5⌣2.6 GHz) under Ubuntu 18.04. Each solving process is allocated a slot of 1 hour and at most 16 GB of memory per instance.

MS-Builder has succeeded to construct full proofs for 163 instance while MS-checker has succeeded to check 575 complete or partial proofs over 576 in total. The running time for building and checking instances are plotted respectively in Figures 6 and 7. Proof checking is obviously much easier than proof building except on rare formulas with an important number of clauses (in the input file or after applying some transformations) which can make difficult the linear operation of extracting a premise to the formula used in MS-Checker.

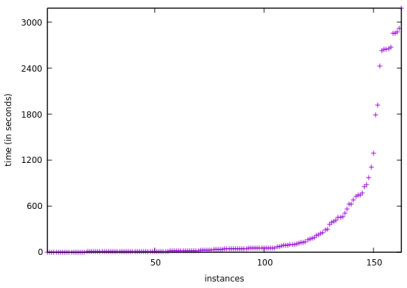

Fig. 6. Running time (in seconds) for building complete proofs

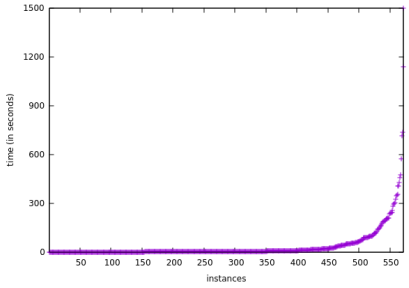

Fig. 7. Running time (in seconds) for checking proof

MS-Builder has also succeed to build at least half of the proofs (with respect to the number of empty clauses) of 302 instances over 463 instances for which the optimum cost is known. This is illustrated in Figure 8 which reports the percentage of empty clauses built per solved instance. The sizes of the computed proofs vary from few bytes to 1 Gigabyte as illustrated in Figure 9. Notice how empty (incomplete) proofs are computed for some very hard instances for which the timeout is not sufficient to even compute the first Max-SAT refutation. On the other hand, there are some instances, usually with an optimum cost of 1, which have very small proofs.

[1] The source code is available on https://pageperso.lis-lab.fr/matthieu.py/en/software.html.

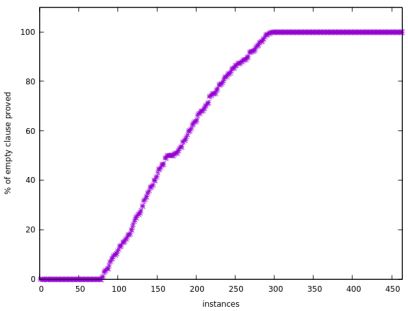

Fig. 8. Percentage of proved □ per instance

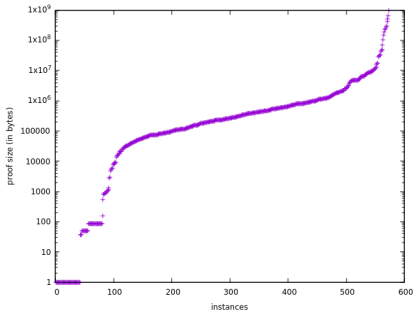

Fig. 9. Size of proof per instance (in logarithmic scale)

Finally, we can observe in Table 1 that read-once resolution refutations and resolution refutations which are read-once after fixing unit propagation appear very often. However, there are many instances such that the last resolution refutation met are the largest and the hardest (i.e. unrestricted) and that is why the last resolution refutations are often the most difficult to adapt and the timeout often stops on these resolution refutations.

Type of resolution refutation	Number	Percentage
read-once	169,239	83.7 %
read-once after UP-fixing	24,556	12.1 %
tree-like regular	2,879	1.4 %
tree-like	1,795	0.9 %
unrestricted	3,799	1.9 %

Table 1. Encountered types of resolution refutations in the whole benchmark

7 Conclusion

In this paper, we proposed two tools, MS-Builder and MS-Checker, to respectively generate and check Max-SAT proofs. MS-Builder builds proofs by iteratively calling a SAT oracle and adapting the obtained SAT refutations into Max-SAT refutations. MS-Builder has succeeded in building a substantial amount of proofs for unweighted partial instances of the 2020 Max-SAT Evaluation. However, unrestricted resolution refutations are usually hard to adapt due to the exponential overhead caused by duplicating parts of the proofs.

As future work, it would be interesting to include more advanced techniques such as core reduction or minimization [2,25] in order to improve the efficiency of these tools. Furthermore, It would be relevant to study the possibility of extending the UP-fixing mechanism to non-unit clauses. Finally, we are also working on extending our tools to build and check proofs for weighted partial Max-SAT formulas.

References

1. Abramé, A., Habet, D.: Ahmaxsat: Description and Evaluation of a Branch and Bound Max-SAT Solver. Journal on Satisfiability, Boolean Modeling and Computation **9**, 89–128 (2015)
2. Alexey Ignatiev, A.M., Marques-Silva, J.: RC2: an efficient maxsat solver. Journal on Satisfiability, Boolean Modeling and Computation. pp. 53–64 (2019)
3. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. Technical Communications of The Twenty-eighth International Conference on Logic Programming (ICLP'12) **17** (01 2012)
4. Bacchus, F., Järvisalo, M., Martins, R.: MaxSAT Evaluation (2020), https://maxsat-evaluations.github.io/2020/
5. Ben-sasson, E., Impagliazzo, R., Wigderson, A.: Near optimal separation of tree-like and general resolution. Combinatorica **24**, 585–603 (2004)
6. Biere, A.: Booleforce, http://fmv.jku.at/booleforce/
7. Biere, A.: TraceCheck, http://fmv.jku.at/tracecheck/
8. Biere, A.: PicoSAT Essentials. Journal on Satisfiability, Boolean Modeling and Computation **4**(2-4), 75–97 (2008)
9. Bonet, M.L., Levy, J.: Equivalence Between Systems Stronger Than Resolution. In: Theory and Applications of Satisfiability Testing – SAT 2020. Lecture Notes in Computer Science, vol. 12178 (2020)
10. Bonet, M.L., Levy, J., Manyà, F.: A complete calculus for MAX-SAT. In: Theory and Applications of Satisfiability Testing - SAT 2006. vol. 4121, pp. 240–251 (08 2006)
11. Bonet, M.L., Levy, J., Manyàb, F.: Resolution for Max-SAT. In: Artificial Intelligence. vol. 171, pp. 606–618 (2007)
12. D'Almeida, D., Grégoire, É.: Model-based diagnosis with default information implemented through MAX-SAT technology. In: IEEE 13th International Conference on Information Reuse & Integration. pp. 33–36. IEEE (2012)
13. Davies, J., Bacchus, F.: Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. In: Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming. p. 225–239. CP'11 (2011)
14. de Givry, S., Larrosa, J., Meseguer, P., Schiex, T.: Solving max-sat as weighted csp. In: Principles and Practice of Constraint Programming – CP 2003. pp. 363–376 (2003)
15. Guerra, J., Lynce, I.: Reasoning over Biological Networks Using Maximum Satisfiability. In: Principles and Practice of Constraint Programming - CP 2012,. Lecture Notes in Computer Science, vol. 7514, pp. 941–956 (2012)
16. Hertel, A., Urquhart, A.: Algorithms and complexity results for input and unit resolution. Journal of Satisfiability, Boolean Modeling and Computation **6** (2009)
17. Iwama, K., Miyano, E.: Intractability of read-once resolution. In: Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference (1995)
18. Küegel, A.: Improved exact solver for the weighted max-sat problem. In: POS-10. Pragmatics of SAT. EPiC Series in Computing, vol. 8, pp. 15–27. EasyChair (2012)
19. Larrosa, J., Heras, F.: Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In: IJCAI International Joint Conference on Artificial Intelligence - IJCAI 2005. pp. 193–198 (01 2005)
20. Larrosa, J., Rollon, E.: Augmenting the Power of (Partial) MaxSat Resolution with Extension. In: Proceedings of the AAAI Conference on Artificial Intelligence (2020)

21. Larrosa, J., Rollon, E.: Towards a better understanding of (partial weighted) maxsat proof systems. In: Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12178, pp. 218–232. Springer (2020)
22. Li, C.M., Manyà, F., Soler, J.R.: A Clause Tableau Calculus for MaxSAT. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. p. 766–772. IJCAI'16 (2016)
23. Li, C.M., Manyà, F., Planes, J.: New inference rules for Max-SAT. Journal of Artificial Intelligence Research (JAIR) **30**, 321–359 (2007)
24. Loveland, D.: A linear format for resolution. Symposium on Automatic Demonstration pp. 147–162 (1970)
25. Marques-Silva, J.: Minimal unsatisfiability: Models, algorithms and applications (invited paper). In: 2010 40th IEEE International Symposium on Multiple-Valued Logic. pp. 9–14 (2010)
26. Martins, R., Manquinho, V.M., Lynce, I.: Open-WBO: A Modular MaxSAT Solver. In: Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference. Lecture Notes in Computer Science, vol. 8561, pp. 438–445 (2014)
27. Narodytska, N., Bacchus, F.: Maximum Satisfiability Using Core-Guided MaxSAT Resolution. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence. pp. 2717–2723 (2014)
28. Py, M., Cherif, M.S., Habet, D.: Towards bridging the gap between sat and max-sat refutations. In: 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI). pp. 137–144 (2020)
29. Robinson, J.A.: A Machine-Oriented Logic Based on the Resolution Principle. Journal of the Association for Computing Machinery **12**, 23–41 (1965)
30. Urquhart, A.: The complexity of propositional proofs. Bull. Symbolic Logic **1**, 425–467 (1995)
31. Urquhart, A.: A near-optimal separation of regular and general resolution. SIAM Journal on Computing **40**, 107–121 (2011)
32. Xu, H., Rutenbar, R.A., Sakallah, K.A.: sub-SAT: a formulation for relaxed Boolean satisfiability with applications in routing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, **22**, 814–820 (2003)