1

Satisfiability Theories

After reading this chapter, you should be able to:

- Comprehend fundamental concepts of Boolean satisfiability (SAT), maximum satisfiability (MaxSAT), and MaxSAT extensions.
- Understand mainstream algorithms for solving SAT and MaxSAT problems.

1.1 Boolean Satisfiability (SAT)

A Boolean formula is a string that represents a Boolean function. A Boolean function is a function of the form: $B^k \rightarrow B$, where $B = \{True, False\}$ is a Boolean domain and k is the arity of the function. Usually, *True* and *False* are represented by 1 and 0, respectively. A propositional Boolean formula is a Boolean formula that only contains logic operations *and*, *or* and *not* (sometimes called *negation* or *complement*), symbolized as $\wedge$, $\vee$, and $\neg$, respectively. Some examples of propositional Boolean formulas are listed as follows:

$$(a \wedge b) \vee (a \wedge \neg c) \vee (b \wedge c \wedge d)$$
$$(a \vee b) \wedge (a \vee \neg b \vee \neg c) \wedge (c \vee \neg a)$$
$$(b \vee \neg c) \wedge (a \vee \neg c) \vee b$$

A propositional Boolean formula can be expressed in conjunctive normal form (CNF), also known as *product of sum (POS)* form. A formula in CNF consists of a conjunction (logic *and*) of one or more clauses. A clause is a disjunction (logic *or*) of one or more literals, and a literal is an occurrence of a Boolean variable or its negation. An example of a propositional Boolean formula in CNF is $\phi = (b \vee \neg c) \wedge (a \vee \neg c) \wedge b$, which can also be represented by a set of clauses as $\phi = \{b \vee \neg c, a \vee \neg c, b\}$, or an assemble of literal sets like $\phi = \{\{b, \neg c\}, \{a, \neg c\}, \{b\}\}$. Usually, a CNF formula is denoted by a set of clauses, i.e., conjunction is omitted.

Applied Satisfiability: Cryptography, Scheduling, and Coalitional Games,
First Edition. Xiaojuan Liao and Miyuki Koshimura.

The problem of determining whether there exists a variable assignment that makes a propositional Boolean formula evaluate to true is called a Boolean satisfiability (SAT) problem. In other words, the SAT problem tries to find a variable assignment to a CNF formula that satisfies all the clauses in a Boolean propositional formula. If such an assignment exists, the formula is *satisfiable*; otherwise, the formula is *unsatisfiable*.

Example 1.1 $\phi = \{a \vee b \vee \neg c \vee d, a \vee \neg b \vee \neg c, a \vee b, \neg d, d\}$ is a Boolean propositional formula in CNF. This formula contains five clauses: $(a \vee b \vee \neg c \vee d)$, $(a \vee \neg b \vee \neg c)$, $(a \vee b)$, $(\neg d)$, and (d). The first clause $(a \vee b \vee \neg c \vee d)$ contains four literals, i.e., a, b, $\neg c$, and d. This formula is unsatisfiable because the last two clauses are conflicting, which means they cannot be satisfied at the same time.

SAT problem is the first known nondeterministic polynomial time (NP)-complete problem proven by Cook [1]. The proof shows how every decision problem in the complexity class NP can be reduced to the SAT problem for CNF formulas. That the SAT problem is NP-complete problem briefly means that there is no known algorithm that efficiently solves all instances of SAT, and it is generally believed that no such algorithm can exist. A class of algorithms to efficiently solve a large enough subset of SAT instances is called SAT solver. Extending the capabilities of SAT solving algorithms is an ongoing area of progress. However, no current such methods can efficiently solve all SAT instances.

In the last century, showing that a certain problem is as hard as SAT was the end of the story and trying to solve it directly seemed to be hopeless. With the significant progress made in SAT solving, it is now widely accepted that being able to encode a problem into SAT is highly likely to lead to a practical solution. This "SAT Revolution" started at the end of the last century and continues to produce amazing new practical and theoretical results [2].

Examples of SAT applications include software verification [3, 4], bounded model checking [5, 6], artificial intelligence (AI) planning [7, 8], cryptographic attacks [9–13], scheduling [14–17], etc. These applications rely on the ability of SAT solvers to determine whether there exists an assignment that makes a given Boolean formula evaluate to true and return one satisfying assignment if the formula is satisfiable. It is noticeable that there are applications that require enumerating all the satisfying assignments. For example, in frequent itemset mining, it is necessary to generate all sets of items with high support from a given transaction database [18]. This opens up a new variant of SAT, called all solutions SAT (AllSAT). Solving an AllSAT problem aims to enumerate all satisfying assignments if a given CNF formula is satisfiable. Readers may refer to [19] for major techniques of ALLSAT solvers.

1.2 Maximum Satisfiability (MaxSAT)

Given a Boolean propositional formula, if it is unsatisfiable, SAT solvers only report that no solution exists, without any information on unsatisfiable instances. However, assignments violating a minimum number of constraints, or satisfying all the compulsory (*hard*) constraints and as many optional (*soft*) constraints as possible, can be considered as acceptable solutions in real-life scenarios. To cope with this limitation of SAT, maximum satisfiability (MaxSAT) and its extensions, such as partial MaxSAT and weighted MaxSAT, are becoming an alternative for representing and efficiently solving over-constrained problems [2].

The MaxSAT problem for a CNF formula is the problem of finding a variable assignment that maximizes the number of satisfied clauses. MaxSAT is often used to mean MinUNSAT because finding an assignment that maximizes the number of satisfied clauses is equivalent to finding an assignment that minimizes the number of unsatisfied clauses. MaxSAT is useful to measure the extent of unsatisfiability of a CNF formula.

Three extensions of MaxSAT are more well suited for representing and solving over-constrained problems: partial MaxSAT, weighted MaxSAT, and weighted partial MaxSAT (WPM).

In a partial MaxSAT instance, each clause is labeled either *hard* or *soft*. The hard clauses must be obligatorily satisfied, while the soft clauses can be unsatisfied. The goal of solving the partial MaxSAT instance is to satisfy all hard clauses and the maximal number of soft clause. The partial MaxSAT problem is easily extended to a SAT problem if all the clauses are hard, and a MaxSAT problem if all the clauses are soft.

Example 1.2 Given a partial MaxSAT instance $\phi = \{[a], [\neg a \vee \neg b], (b \vee c)\}$, the first and second clause, enclosed by "[ ]," are hard clauses, and the third clause, enclosed by "( )," is a soft clause. To satisfy the hard clauses, a and b are forced to be 1 and 0, respectively. Based on the assignment of a and b, the true assignment of c has to be 1 so that the soft clause can be satisfied.

A weighted MaxSAT instance is expressed in weighted CNF, where each clause is assigned a positive integer. The problem is to find a truth assignment that maximizes the sum of weights of satisfied clauses. In a special case, if the weights of all the clauses in weighted MaxSAT are equal to one, the problem is regarded as a MaxSAT problem.

Example 1.3 A weighted MaxSAT instance $\phi = \{(a, 2), (\neg a \vee \neg b, 3), (b \vee c, 4)\}$ has three weighed clauses holding weights 2, 3, 4, respectively. All of the three

clauses can be satisfied by assigning 1, 0, 1 to a, b, c, respectively, which leads to the maximal sum of weights of satisfied clauses.

The WPM is the combination of partial MaxSAT and WPM. A WPM instance distinguishes hard and soft clauses, where each soft clause is assigned a positive integer. Solving the WPM instance is to satisfy all hard clauses and maximize the sum of weights of satisfied soft clauses. The definition of WPM can be easily extended to partial MaxSAT, where all soft clauses have weight 1, and weighted MaxSAT, where no clauses are hard.

Example 1.4 Given a WPM instance $\phi = \{[a], [\neg a \vee \neg b], (b \vee c, 2), (b \vee \neg c, 7)\}$, the first and second clause, enclosed by "[ ]," are hard clauses, while the third and fourth clauses, enclosed by "( )," are soft clauses. To satisfy the hard clauses, a and b are forced to be 1 and 0, respectively. Based on the assignment of a and b, the true assignment of c is preferred to be 0 so that the weight 7 can be earned, which is larger than the other case that the gain is merely 2.

Many important problems can be naturally expressed as MaxSAT, including academic problems such as Max-Cut or Max-Clique, as well as problems from many industrial domains. Concrete examples include the following domains: routing problems [20], hardware debugging [21–23], software debugging [24, 25], scheduling [17, 26–28], planning [29–32], coalitional games [33–37], etc. Additionally, many problems originally formulated in other optimization frameworks can be easily reformulated as MaxSAT, such as the Pseudo-Boolean Optimization framework [38], the Weighted Constraint Satisfaction Problem (WCSP) framework [39], and the MaxSMT framework [40]. Readers may refer [41] for more traditional applications of MaxSAT.

1.3 Satisfiability Algorithms

The last two decades have witnessed significant progress in the development of theoretical, logical and algorithmic aspects of SAT and MaxSAT solving techniques. Moreover, the SAT competition[1] (starting from 2002), MaxSAT evaluation[2] (starting from 2006), and the international conference on theory and applications of satisfiability testing[3] (first held in 1996) jointly play as a driving force for motivating the development of novel SAT and MaxSAT technologies.

1 http://satcompetition.org.
2 https://maxsat-evaluations.github.io.
3 www.satisfiability.org.

1.3.1 SAT Algorithms

Algorithms for solving the SAT problem are mainly based on incomplete algorithms and complete algorithms. Incomplete algorithms (e.g., GSAT [42], WalkSAT [43]) can get a satisfying assignment quickly, but do not prove unsatisfiability if the formula is unsatisfiable. By contrast, complete algorithms (e.g., DPLL algorithm and its variants Chaff [44] and Generic seaRch Algorithm for the Satisfiability Problem, GRASP [45]) guarantee to find satisfying assignment or prove unsatisfiability.

Incomplete algorithms are usually based on stochastic local search, which iteratively improve an assignment of the variables until all constraints are satisfied. Stochastic local search algorithms run efficiently especially on large-scale instances. The primary challenge is the cycling problem where a candidate solution may be revisited [46]. Random walk, restarting strategies, and the configuration check [46] are usually used to tackle this problem. For an in-depth understanding of most recent incomplete algorithms, we recommend readers to consult a comprehensive survey on intelligent optimization algorithms [47].

Complete algorithms are based on the modern improvements or variants of the DPLL approach. DPLL [48, 49], named after the four collective authors, i.e., Davis, Putnam, Logemann, and Loveland, is a complete, backtracking-based algorithmic framework that is the basis of many of the most successful modern complete SAT solvers.

The backtracking algorithm takes a CNF formula ϕ as input. Its objective is to determine whether ϕ is satisfiable, and if so to find a satisfying assignment. The algorithm maintains a truth assignment and runs recursively. Initially, the assignment is empty with all variables unassigned. At each recursive step, the algorithm selects a literal l and explores two potential assignments: true and false. This is done by making recursive calls to investigate both assignments to l. The algorithm continues this process until it either finds a satisfying assignment or it exhaustively search through the entire space and concludes that ϕ is unsatisfiable [50].

In the worst case, the backtracking algorithm has to explore the entire search space by setting each literal first true and then false. To enhance the search efficiency, based on the backtracking algorithm, DPLL is enhanced by *unit propagation* and *pure-literal elimination* rules.

If a clause contains only a single unassigned literal, it is called a unit clause. Given a CNF formula, the unit-clause rule checks if the formula contains a unit clause. If so, it removes all clauses that contain the literal from the formula and removes the negation of the literal from the other clauses. Unit propagation is the iterated application of the unit-clause rule, which repeatedly applies the unit-clause rule until either it derives an empty clause or there are no more unit clauses left.

Example 1.5 Consider the following propositional formula in CNF:

$$((a \vee b \vee c \vee \neg d) \wedge (\neg a \vee c) \wedge (\neg c \vee d) \wedge (a))$$

In this formula, (a) is a unit clause and must be assigned true, then it can be removed from the formula. The first clause $(a \vee b \vee c \vee \neg d)$ contains a, thus this clause is always satisfied and can be removed from the formula safely.

Consider the second clause $(\neg a \vee c)$, which contains $\neg a$. Now that we have known that $\neg a$ is false, we can remove $\neg a$ from the clause since it has nothing to do with the truth value of the clause.

After unit propagation, the formula can be simplified as

$$((c) \wedge (\neg c \vee d))$$

Here, we find the simplified formula contains a new unit clause (c). Continue with the unit propagation. We can remove c from the formula and remove $\neg c$ from the clause $(\neg c \vee d)$. Thereby, the formula is simplified as

$$(d)$$

Apparently, (d) is a unit clause and can be removed from the formula. Eventually, the formula becomes empty, and the original formula is satisfiable.

If a literal appears in some clause but its negation does not appear in the formula, it is called a pure literal. Pure-literal elimination removes all clauses containing a pure literal because these clauses can be always satisfied by making the pure literal true without falsifying any other clauses.

Example 1.6 Consider the following propositional formula in CNF:

$$((\neg a \vee b) \wedge (a \vee c \vee \neg b) \wedge (c \vee \neg d) \wedge (d \vee \neg e))$$

In the formula, c and $\neg e$ are pure literals. We can always satisfy the clauses $(a \vee c \vee \neg b)$, $(c \vee \neg d)$, and $(d \vee \neg e)$ by assigning c to true and e to false. After the pure-literal elimination, the formula can be simplified as

$$(\neg a \vee b)$$

This simplified formula contains pure literals $\neg a$ and b. Therefore, we can remove this clause from the formula. Eventually, the formula is empty, and the original formula is satisfiable.

The DPLL procedure is described in Algorithm 1.1. Functions *UnitPropagate*(ϕ) and *PureLiteralEliminate*(ϕ) take a CNF formula ϕ as input and return the simplified formula of applying unit propagation and pure-literal elimination, respectively. By picking an unassigned variable from ϕ, which is accomplished

by *SelcetVariable*(ϕ), and assigning truth values to it, the resulting formula is simplified iteratively. This process continues until ϕ is empty or ϕ contains an empty clause, indicating that ϕ is satisfiable or unsatisfiable, respectively.

Algorithm 1.1 DPLL recursive procedure

Input: A CNF formula ϕ.
Output: SAT if ϕ is satisfiable; UNSAT, otherwise.

```
1:  function DPLL(φ)
2:      while φ contains a unit clause do
3:          φ ← UnitPropagate(φ)                          ▷ unit propagation
4:      end while
5:      while φ contains a pure literal do
6:          φ ← PureLiteralEliminate(φ)                   ▷ pure-literal elimination
7:      end while
8:      if φ is empty then
9:          return SAT                                    ▷ φ is satisfiable.
10:     end if
11:     if φ contains an empty clause then
12:         return UNSAT                                  ▷ φ is unsatisfiable.
13:     end if
14:     x ← SelectVariable(φ)                             ▷ Select an unassigned literal x
15:     return DPLL(φ_{x←true}) ∨ DPLL(φ_{x←false})
16: end function
```

The DPLL algorithm has two primary limitations. First, upon encountering a conflict, where a variable assignment renders a clause unsatisfiable, the algorithm identifies only the present variable assignment as the source of the conflict, lacking the ability to deduce specific combinations of variable assignments that may also trigger the conflict. Second, when a conflict occurs, the algorithm employs the chronological backtracking approach, jumping back to the most recent variable assignment that led to the conflict and assigning the opposite truth value to that variable. This strategy may result in a considerable amount of time, as it wastes time exploring variable assignments in a search space that is guaranteed to lead to conflicts.

Conflict-driven clause learning (CDCL) [44, 45], improved from the DPLL algorithm, incorporates conflict analysis and clause learning techniques to avoid repeatedly exploring the same impossible assignments to literals, thus pruning the search space effectively and expediting the solving process. Main improvements of the CDCL are *clause learning from conflicts* and *nonchronological backtracking*, which are summarized as follows:

- Clause learning from conflicts: When detecting a conflict, i.e., a clause becomes unsatisfiable, CDCL analyzes the conflict to identify the cause and adds a learnt clause (the negation of the assignments that led to the conflict) to the original formula. This learnt clause is essential for avoiding the same conflict in the future, as it blocks the problematic assignment.
- Nonchronological backtracking: If a conflict arises, CDCL jumps back to the appropriate decision level where the first-assigned variable involved in the conflict and assigns the opposite truth value to it. The backtracking level may not necessarily be the level where the most recently assigned variable locates. This allows the algorithm to flexibly select the decision level for backtracking and intelligently avoid conflict repetition by skipping over potentially unnecessary variable assignments.

Given the ability to discover early that branches of the search space does not have to be explored, the CDCL algorithm is extremely efficient for solving large-scale SAT problems. Currently, the CDCL algorithm stands out as one of the most efficient methods for solving SAT problems, laying the foundation of many modern SAT solvers (e.g., MiniSAT [51], Glucose [52], MergeSat [53], and IsaSAT [54]). For more details on the CDCL procedure, we recommend readers to refer to [50]. For a comprehensive study and analysis of the latest developments and new approaches of SAT solvers, we suggest referring to [55].

1.3.2 MaxSAT Algorithms

Generally speaking, there are two approaches for MaxSAT solving techniques: heuristic and approximation algorithms that find near-optimal solutions and exact algorithms that compute optimal solutions.

Heuristic local search algorithms are the foundation of early practical works to find near-optimal solutions. Whereas many exact MaxSAT solvers use local search algorithms to rapidly compute an initial assignment of variables, these algorithms do not guarantee the quality of their output solutions. By contrast, approximation algorithms output solutions not as fast as heuristic algorithms, but they provide a guarantee about the quality of their solutions. The approximation of a MaxSAT solution is usually measured by a factor that is bounded by a constant α or a slowly growing function of the input size. Given a constant α, an algorithm is α-approximation for a maximization problem if it provides a feasible solution in polynomial time, which is at least α times the optimum, considering all the possible instances of the problem [56]. A number of improvements have been achieved on the performance guarantee, from 1/2, proposed in 1974 [57], to 0.7584, proposed in 1995 [58]. Later on, a limit on approximability was proved by Håstad [59] that unless NP=P, no approximation algorithm for MaxSAT can

achieve a performance guarantee better than 7/8. This theory was proved again in [60], showing that the constant 7/8 is tight. From a theoretical and practical point of view, semidefinite programming has been shown quite promising for approximating MaxSAT solutions. Readers may refer to [61] to learn more about how to approximate MaxSAT with semidefinite programming.

Exact algorithms can be classified into two approaches. The one follows a branch and bound (BB) algorithm and applies several techniques tailored to MaxSAT. Another one makes use of a state-of-the-art SAT solver as an inference engine, referred to as a SAT-based approach.

Many contemporary exact MaxSAT solvers follow a BB algorithm [62–69], which ensures the minimal number of unsatisfied clauses in a MaxSAT problem. Given a MaxSAT instance ϕ, BB explores a search tree that represents the space of all possible assignments for ϕ in a depth-first manner. At every node, BB compares the upper bound (*UB*) with the lower bound (*LB*). *UB* is the best solution (i.e., the minimum number of falsified clauses) found so far for a complete assignment, and *LB* is the sum of the number of clauses which are falsified by the current partial assignment plus an underestimation of the number of clauses that will become unsatisfied if the current partial assignment is completed. If $LB \geq UB$, the algorithm prunes the subtree below the current node and backtracks chronologically to a higher level in the search tree. If $LB < UB$, the algorithm tries to find a better solution by extending the current partial assignment by assigning one more variable. The value of *UB* after the search of entire tree is the optimal number of unsatisfied clauses in ϕ.

BB algorithms usually perform effective in solving small-scale MaxSAT problem instances with a few hundred variables. However, they are ineffective on instances with more than a thousand variables [70]. Currently, the most important algorithms for solving MaxSAT problems are based on iterative calls to a SAT solver, commonly known as the SAT-based approaches. These approaches can be further classified into three types: the model-improving, the core-guided, and the implicit hitting set approaches.

Model-improving approaches (e.g., SAT4j [71], QMaxSAT [72], Pacose [73], and MergeSat [74]) are based on querying a SAT solver for solutions of increasing quality. Given a MaxSAT instance ϕ with a collection of n soft clauses $\{C_1, \ldots, C_n\}$, a new variable b_i is added to each soft clause C_i ($1 \leq i \leq n$), where b_i is called a *blocking variable*. Solving the MaxSAT problem for ϕ is to minimize the number of blocking variables that evaluate to true, called *true blocking variables*, in ϕ' with soft clauses $\{C_1 \vee b_i, \ldots, C_n \vee b_n\}$. The minimal satisfied assignment is searched by iterative calls to a SAT solver, which are summarized as follows [72]. First run the SAT solver on ϕ' without any constraints to get an initial model and count the number k of true blocking variables in the model, then add a constraint to limit the number of true blocking variables to a smaller k (called a cardinality constraint),

and run the solver again. If the problem is unsatisfied, k is the optimal solution. Otherwise, the process is repeated with the constraint that limits the number of true blocking variables to a smaller integer. This process terminates when the problem becomes unsatisfied.

Model-improving solvers start by adding a blocking variables to each soft clause and then iteratively call a SAT solver by decreasing the number of true blocking variables until the formula becomes unsatisfiable. When a MaxSAT problem has a significantly large number of soft clauses, solving this problem becomes infeasible due to the resulting extensive encodings.

In contrast to model improving approaches, core-guided approaches (e.g., Fu–Malik algorithm [75], WPM1 [76], WMSU1 [77], OLL [78], and Maxino [79]) are unsatisfiability based, working from unsatisfiable (UNSAT) to SAT. Given a MaxSAT instance ϕ, the following process is iterated until ϕ is satisfiable: First run a SAT solver on ϕ. If ϕ is unsatisfiable, extract an unsatisfiable subset $US = \{C_1, \ldots, C_m\}$ from ϕ and introduce m new blocking variables b_i $(1 \leq i \leq m)$. The unsatisfiable subset of clauses is called a *core*. Then, replace C_i with $C_i \vee b_i$ $(1 \leq i \leq m)$ and add a constraint $\sum_{i=1}^{m} b_i = 1$ to build a new formula ϕ'. The process is called *relaxation*. If ϕ' becomes satisfiable, the iteration terminates. The number of iterations indicates the number of falsified clauses in the original ϕ.

Core-based algorithms have proved to be effective on industrial problems, typically suitable for solving instances that have optimal solutions falsifying very few soft clauses (relative to the total number of soft clauses). A drawback of core-based algorithms is that the relaxation requires adding new cardinality constraints to the formula at each step, which makes the problem harder to solve at each iteration. Narodytska and Bacchus [80] proposed an alternative approach for solving WPM problems, which also builds a sequence of new SAT formulas, but avoids using cardinality constraints. The new core-guided solver outperforms the contemporary solvers by a large margin in the total number of industrial problem instances.

Similar to the purely SAT-based core-guided approaches, implicit hitting set solvers (e.g., [81–84]) extract cores in an iterative fashion. Given a set of cores, a hitting set is a set of soft clauses that includes at least one soft clause from each core. A hitting set is optimal (of minimum cost) if and only if the sum of the weights of the soft clauses in it is smallest among all hitting sets of the set of cores. In contrast to core-based algorithms that transform the input instance using core compilation steps, the input MaxSAT instance of implicit hitting set solvers is not altered during search. Each SAT solver call is made on the original hard clauses together with a subset of the original soft clauses. Thus, the MaxSAT instance does not get larger in size as the search progresses. As a result, the cores found during search remain relatively small compared to the core-guided approaches [70].

Although SAT and MaxSAT problems have a long history, the solving techniques have evolved and improved over the years. Many advanced techniques have been introduced to aid in problem solving, such as parallel computing [74, 85–88], reinforcement learning [88, 89], machine learning [90–92], and quantum computing [93–95]. These new techniques, in turn, encourage SAT and MaxSAT applications in more real-world scenarios.

1.4 Chapter Summary

This section introduced the fundamental formal concepts of SAT, MaxSAT, partial MaxSAT, weighted MaxSAT, and WPM. Mainstream algorithms for solving SAT and MaxSAT were summarized.

From a practical view, many existing applications treat SAT and MaxSAT solvers as black boxes. In other words, once a real-world problem is translated into a CNF formula, the problem can be solved by utilizing an off-the-shelf solver without any interactions between the solver and the application. However, understanding the satisfiability algorithms is beneficial for selecting appropriate solvers, analyzing performance, and further optimizing the efficiency of solving specific problems.

References

1 S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

2 A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability* - Second Edition, IOS Press, 2021.

3 E. M. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, pages 168–176, 2004.

4 D. Jackson, I. Schechter, and I. Shlyakhter. Alcoa: the alloy constraint analyzer. In *Proceedings of International Conference on Software Engineering*, pages 730–733, 2000.

5 A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207, 1999.

6 M. Sheeran, S. Singh, and G. Stalmarck. Checking safety properties using induction and a SAT solver. In *Proceedings of the 3rd International Conference on Formal Methods in Computer-Aided Design*, pages 108–125, 2000.

7 J. Rintanen, K. Heljanko, and I. Niemelä. Planning as satisfiability: Parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

8 B. Selman and H. Kautz. Planning as satisfiability. In *Proceedings of European Conference on Artificial Intelligence*, pages 359–363, 1992.

9 A. A. Kamal. Applications of SAT solvers to AES key recovery from decayed key schedule images. In *Proceedings of the 4th International Conference on Emerging Security Information Systems and Technologies*, pages 216–220, 2010.

10 C. Patsakis. RSA private key reconstruction from random bits using SAT solvers. *IACR Cryptology ePrint Archive*, 2013.

11 X. Liao, H. Zhang, M. Koshimura, H. Fujita, and R. Hasegawa. Using MaxSAT to correct errors in AES key schedule images. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013*, Herndon, VA, USA, November 4–6, 2013, pages 284–291. IEEE Computer Society, 2013.

12 X. Liao, H. Zhang, and M. Koshimura. Reconstructing AES key schedule images with SAT and MaxSAT. *IEICE Transactions on Information and Systems*, 99-D(1):141–150, 2016.

13 I. Zimerman, E. Nachmani, and L. Wolf. Recovering AES keys with a deep cold boot attack. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, 18–24 July 2021, Virtual Event, volume 139 of *Proceedings of Machine Learning Research*, pages 12955–12966. PMLR, 2021.

14 J. M. Crawford and A. B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *12th AAAI National Conference on Artificial Intelligence*, pages 1092–1097, 445 Burgess Drive Menlo Park, CA, United States, Oct. 1994. American Association for Artificial Intelligence.

15 M. Koshimura, H. Nabeshima, H. Fujita, and R. Hasegawa. Solving open job-shop scheduling problems by SAT encoding. *IEICE Transactions on Information and Systems*, E93-D(8):2316–2318, 2010.

16 R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and E. Rollon. Employee scheduling with SAT-based pseudo-Boolean constraint solving. *IEEE Access*, 9:142095–142104, 2021.

17 X. Liao, H. Zhang, M. Koshimura, R. Huang, and F. Li. Solving restricted preemptive scheduling on parallel machines with SAT and PMS. *Journal of Universal Computer Science*, 29(8):911–937, 2023.

18 I. O. Dlala, S. Jabbour, L. Saïs, and B. B. Yaghlane. A comparative study of SAT-based itemsets mining. In M. Bramer and M. Petridis, editors, *Research and Development in Intelligent Systems XXXIII - Incorporating Applications and Innovations in Intelligent Systems XXIV. Proceedings of AI-2016, The 36th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Cambridge, UK, December 13–15, 2016, pages 37–52. Springer, 2016.

19 T. Toda and T. Soh. Implementing efficient all solutions SAT solvers. *ACM Journal of Experimental Algorithmics*, 21(1):1.12:1–1.12:44, 2016.

20 H. Xu, R. Rutenbar, and K. Sakallah. Sub-SAT: A formulation for relaxed Boolean satisfiability with applications in routing. In *Proceedings of International Symposium on Physical Design*, pages 182–187, 2002.

21 S. Safarpour, H. Mangassarian, A. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *Proceedings of the 7th Conference on Formal Methods in Computer-Aided Design*, pages 13–19, 2007.

22 Y. Chen, S. Safarpour, A. Veneris, and J. Marques-Silva. Spatial and temporal design debug using partial MaxSAT. In *Proceedings of ACM Great Lakes Symposium on VLSI*, pages 345–350, 2009.

23 H. Mangassarian, A. G. Veneris, S. Safarpour, F. N. Najm, and M. S. Abadir. Maximum circuit activity estimation using pseudo-Boolean satisfiability. In *Proceedings of Conference on Design, Automation and Test in Europe*, pages 1538–1543, 2007.

24 M. Jose and R. Majumdar. Bug-assist: Assisting fault localization in ANSI-C programs. In *Proceedings of International Conference on Computer Aided verification*, pages 504–509, 2011.

25 M. Jose and R. Majumdar. Cause clue clauses: Error localization using maximum satisfiability. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 437–446, 2011.

26 E. Demirović, N. Musliu, and F. Winter. Modeling and solving staff scheduling with partial weighted MaxSAT. *Annals of Operations Research*, 275(1):79–99, 2019.

27 X. Liao, H. Zhang, M. Koshimura, R. Huang, W. Yu, and F. Li. Modeling and solving scheduling in overloaded situations with weighted partial MaxSAT. *Mathematical Problems in Engineering*, 2021:1–17, 2021.

28 A. Lemos, P. T. Monteiro, and I. Lynce. Introducing *UniCorT*: An iterative university course timetabling tool with MaxSAT. *Journal of Scheduling*, 25(4):371–390, 2022.

29 M. C. Cooper, S. Cussat-Blanc, M. de Roquemaurel, and P. Régnier. Soft arc consistency applied to optimal planning. In *Proceedings of International Conference on Principles and Practice of Constraint Programming*, pages 680–684, 2006.

30 F. Juma, E. I. Hsu, and S. A. McIlraith. Preference-based planning via MaxSAT. In *Proceedings of Canadian Conference on AI*, pages 109–120, 2012.

31 N. Robinson, C. Gretton, D. N. Pham, and A. Sattar. Partial weighted MaxSAT for optimal planning. In *Proceedings of Pacific Rim International Conference on Artificial Intelligence*, pages 231–243, 2010.

32 L. Zhang and F. Bacchus. MaxSAT heuristics for cost optimal planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 1846–1852, 2012.

33 X. Liao, M. Koshimura, H. Fujita, and R. Hasegawa. Solving the coalition structure generation problem with MaxSAT. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012*, Athens, Greece, November 7–9, 2012, pages 910–915. IEEE Computer Society, 2012.

34 X. Liao, M. Koshimura, H. Fujita, and R. Hasegawa. MaxSAT encoding for MC-net-based coalition structure generation problem with externalities. *IEICE Transactions on Information and Systems*, 97-D(7):1781–1789, 2014.

35 X. Liao, M. Koshimura, H. Fujita, and R. Hasegawa. Extending MaxSAT to solve the coalition structure generation problem with externalities based on agent relations. *IEICE Transactions on Information and Systems*, 97-D(7):1812–1821, 2014.

36 X. Liao and M. Koshimura. A comparative analysis and improvement of MaxSAT encodings for coalition structure generation under MC-nets. *Journal of Logic and Computation*, 29(6):913–931, 2019.

37 X. Liao, M. Koshimura, K. Nomoto, S. Ueda, Y. Sakurai, and M. Yokoo. Improved WPM encoding for coalition structure generation under MC-nets. *Constraints*, 24(1):25–55, 2019.

38 F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Generic ILP versus specialized 0-1 ILP: An update. In *Proceedings of International Conference on Computer-Aided Design*, pages 450–457, 2002.

39 J. Larrosa and T. Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Inteligence*, 159(1-2):1–26, 2004.

40 R. Nieuwenhuis and A. Oliveras. On SAT modulo theories and optimization problems. In *Proceedings of International Conference on Theory and Applications of Satisfiability Testing*, pages 156–169, 2006.

41 A. Morgado, F. Heras, M. H. Liffiton, J. Planes, and J. Marques-Silva Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

42 D. Mitchell, B. Selman, and H. Leveque. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, 1992.

43 B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In B. Hayes-Roth and R. E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence*, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1, pages 337–343. AAAI Press / The MIT Press, 1994.

44 M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001*, Las Vegas, NV, USA, June 18–22, 2001, pages 530–535. ACM, 2001.

45 J. P. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

46 S. Cai, K. Su, and A. Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9-10):1672–1696, 2011.

47 L. Yang, X. Wang, H. Ding, Y. Yang, X. Zhao, and L. Pang. A survey of intelligent optimization algorithms for solving satisfiability problems. *Journal of Intelligent Fuzzy Systems*, 45(1):445–461, 2023.

48 M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

49 M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

50 S. Buss and J. Nordström. Proof complexity and SAT solving. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability* - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 233–350. IOS Press, 2021.

51 N. Eén and N. Sörensson. An extensible sat-solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 502–518, Italy, May 2003. Springer.

52 G. Audemard, J.-M. Lagniez, and L. Simon. Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *Theory and Applications of Satisfiability Testing –SAT 2013*, pages 309–317, Berlin, Heidelberg, 2013. Springer-Verlag.

53 N. Manthey. The MergeSAT solver. In C.-M. Li and F. Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference*, Barcelona, Spain, July 5–9, 2021*, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 2021.

54 M. Fleury and P. Lammich. A more pragmatic CDCL for IsaSAT and targetting LLVM (Short Paper). In B. Pientka and C. Tinelli, editors, *Automated Deduction - CADE 29 - 29th International Conference on Automated Deduction*, Rome, Italy, July 1–4, 2023, *Proceedings*, volume 14132 of *Lecture Notes in Computer Science*, pages 207–219. Springer, 2023.

55 S. Alouneh, S. Abed, M. H. Al Shayeji, and R. Mesleh. A comprehensive study and analysis on SAT-solvers: Advances, usages and achievements. *Artificial Intelligence Review*, 52(4):2575–2601, 2019.

56 C. M. Li and F. Manyà. MaxSAT, hard and soft constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability* - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 903–927. IOS Press, 2021.

57 D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

58 M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

59 J. Håstad. Some optimal inapproximability results. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, pages 1–10, 1997.

60 H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 406–415, 1997.

61 M. F. Anjos. Semidefinite optimization approaches for satisfiability and maximum-satisfiability problems. *Journal on Satisfiability, Boolean Modeling and Computation*, 1(1):1–47, 2006.

62 F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: An efficient weighted Max-SAT solver. *The Journal of Artificial Intelligence Research*, 31:1–32, 2008.

63 H. Lin, K. Su, and C. M. Li. Within-problem learning for efficient lower bound computation in Max-SAT solving. In *Proceedings of the 22th AAAI Conference on Artificial Intelligence*, pages 351–356, 2008.

64 K. Pipatsrisawat and A. Darwiche. Clone: Solving weighted Max-SAT in a reduced search space. In *Proceedings of 20th Australian Joint Conference on Artificial Intelligence*, pages 223–233, 2007.

65 T. Alsinet, F. Many, and J. Planes. An efficient solver for weighted Max-SAT. *Journal of Global Optimization*, 41(1):61–73, 2008.

66 J. Larrosa, F. Heras, and S. de Givry. A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, 172(2-3):204–233, 2008.

67 J. Davies, J. Cho, and F. Bacchus. Using learnt clauses in MaxSAT. In *Proceedings of International Conference on Principles and Practice of Constraint Programming*, pages 176–190, 2010.

68 A. Kügel. Improved exact solver for the weighted MAX-SAT problem. In D. Le Berre, editor, *POS-10. Pragmatics of SAT*, Edinburgh, UK, July 10, 2010, volume 8 of *EPiC Series in Computing*, pages 15–27. EasyChair, 2010.

69 A. Abramé and D. Habet. Ahmaxsat: Description and evaluation of a branch and bound Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 9(1):89–128, 2014.

70 F. Bacchus, M. Järvisalo, and R. Martins. Maximum satisfiabiliy. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability* - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 929–991. IOS Press, 2021.

71 D. L. Berre and A. Parrain. The SAT4J library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

72 M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:95–100, 2012.

73 T. Paxian, S. Reimer, and B. Becker. Pacose: An iterative SAT-based MaxSAT solver. *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, B-2018-2 of Department of Computer Science Series of Publications B(1):20, 2018.

74 N. Manthey. Parallel by default - MergeSat and MergeSat-Pcasso. *MaxSAT Evaluation 2023: Solver and Benchmark Descriptions*, B-2023-1 of Department of Computer Science Series of Publications B(1):34–36, 2023.

75 Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In A. Biere and C. P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference*, Seattle, WA, USA, August 12–15, 2006, *Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006.

76 C. Ansótegui, M. L. Bonet, and J. Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In *Proceedings of International conference on Theory and Applications of Satisfiability Testing*, pages 427–440, 2009.

77 V. M. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for weighted Boolean optimization. In O. Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009*, Swansea, UK, June 30 - July 3, 2009. *Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009.

78 A. Morgado, C. Dodaro, and J. Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In B. O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014*, Lyon, France, September 8–12, 2014. *Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014.

79 M. Alviano. Maxino. *MaxSAT Evaluation 2017: Solver and Benchmark Descriptions*, B-2017-2 of Department of Computer Science Series of Publications B(1):10–11, 2017.

80 N. Narodytska and F. Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In C. E. Brodley and P. Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, July 27–31, 2014, Québec City, Québec, Canada, pages 2717–2723. AAAI Press, 2014.

81 J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In J. H.-M. Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011*, Perugia, Italy, September 12–16, 2011. *Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011.

82 J. Davies and F. Bacchus. Exploiting the power of MIP solvers in MaxSAT. In M. Järvisalo and A. Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference*, Helsinki, Finland, July 8–12, 2013. *Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013.

83 J. Davies and F. Bacchus. Postponing optimization to speed up MAXSAT solving. In C. Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013*, Uppsala, Sweden, September 16–20, 2013. *Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013.

84 P. Saikko, J. Berg, and M. Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In N. Creignou and D. Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference*, Bordeaux, France, July 5–8, 2016*, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016.

85 R. Martins, V. M. Manquinho, and I. Lynce. Exploiting cardinality encodings in parallel maximum satisfiability. In *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011*, Boca Raton, FL, USA, November 7–9, 2011, pages 313–320. IEEE Computer Society, 2011.

86 M. Terra-Neves, I. Lynce, and V. M. Manquinho. Non-portfolio approaches for distributed maximum satisfiability. In *28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016*, San Jose, CA, USA, November 6–8, 2016, pages 436–443. IEEE Computer Society, 2016.

87 A. Biere. Lingeling essentials, A tutorial on design and implementation aspects of the SAT solver lingering. In D. Le Berre, editor, *POS-14. 5th Pragmatics of SAT Workshop, A Workshop of the SAT 2014 Conference, Part of FLoC 2014 During the Vienna Summer of Logic*, July 13, 2014, Vienna, Austria, volume 27 of *EPiC Series in Computing*, page 88. EasyChair, 2014.

88 H. Nabeshima and K. Inoue. Reproducible efficient parallel SAT solving. In L. Pulina and M. Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference*, Alghero, Italy, July 3–10, 2020*, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 123–138. Springer, 2020.

89 J. Zheng, K. He, J. Zhou, Y. Jin, C.-M. Li, and F. Manyà. BandMaxSAT: A local search MaxSAT solver with multi-armed bandit. *arXiv preprint arXiv:2201.05544*, 2022.

90 H. Wu. Improving Sat-solving with machine learning. In M. E. Caspersen, S. H. Edwards, T. Barnes, and D. D. Garcia, editors, *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2017*, Seattle, WA, USA, March 8–11, 2017, pages 787–788. ACM, 2017.

91 W. Guo, H.-L. Zhen, X. Li, W. Luo, M. Yuan, Y. Jin, and J. Yan. Machine learning methods in solving the Boolean satisfiability problem. *Machine Intelligence Research*, 20(5):640–655, 2023.

92 M. Liu, P. Huang, F. Jia, F. Zhang, Y. Sun, S. Cai, F. Ma, and J. Zhang. Can graph neural networks learn to solve the MaxSAT problem? (Student

Abstract). *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(13):16264–16265, Sept. 2023.

93 C. M. Varmantchaonala, J. L. E. K. Fendji, J. P. T. Njafa, and M. Atemkeng. Quantum hybrid algorithm for solving SAT problem. *Engineering Applications of Artificial Intelligence*, 121:106058, 2023.

94 S.-W. Lin, T.-F. Wang, Y.-R. Chen, Z. Hou, D. Sanán, and Y. S. Teo. A parallel and distributed quantum SAT solver based on entanglement and quantum teleportation. *CoRR*, abs/2308.03344, 2023.

95 A. Alasow and M. Perkowski. Quantum algorithm for maximum satisfiability. In *2022 IEEE 52nd International Symposium on Multiple-Valued Logic (ISMVL)*, pages 27–34, 2022.