Chapter preprint of the second edition of the Handbook of Satisfiability (2021)
Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh (Eds.)
Published by IOS Press. The print edition can be ordered at
https://www.iospress.nl/book/handbook-of-satisfiability-2/
The published electronic chapters (Version of Record) are available at
https://ebooks.iospress.nl/ISBN/978-1-64368-161-0/

Chapter 24

Maximum Satisfiability

Fahiem Bacchus, Matti Järvisalo, and Ruben Martins

Maximum satisfiability (MaxSAT) is an optimization version of SAT that is solved by finding an optimal truth assignment instead of just a satisfying one. In MaxSAT the objective function to be optimized is specified by a set of weighted soft clauses: the objective value of a truth assignment is the sum of the weights of the soft clauses it satisfies. In addition, the MaxSAT problem can have hard clauses that the truth assignment must satisfy. Many optimization problems can be naturally encoded into MaxSAT and this, along with significant performance improvements in MaxSAT solvers, has led to MaxSAT being used in a number of different application areas. This chapter provides a detailed overview of the approaches to MaxSAT solving that have in recent years been most successful in solving real-world optimization problems. Further recent developments in MaxSAT research are also overviewed, including encodings, applications, preprocessing, incomplete solving, algorithm portfolios, partitioning-based solving, and parallel solving.

24.1. Introduction

Computational problems that arise from real-world application scenarios often involve constraints that specify the feasible solutions and some notion of cost or goodness associated with the different solutions. Whereas satisfiability (SAT) provides no direct way of distinguishing solutions in terms of their cost or goodness, maximum satisfiability (MaxSAT for short) generalizes SAT with a over the solutions. This cost function is specified by a set of soft clauses with weights: the cost of a solution is the sum of the weights of the soft clauses it fails to satisfy. (Equivalently, the objective value of a solution is the sum of the weights of the soft clauses it satisfies.)

MaxSAT was for most of its history used mainly as a theoretical problem in the analysis of the computational complexity of optimization problems. Within the last decade, however, with the rise of increasingly effective MaxSAT solving techniques and solver implementations, MaxSAT has changed from a theoretical formalism into a viable constraint optimization paradigm. The approach of encoding various NP-hard optimization problems into MaxSAT and then utilizing

MaxSAT solvers has proved to be a viable solution technique capable of efficiently solving problems from a variety of domains. These domains include combinatorics, planning and scheduling, verification, security and software management, data analysis and machine learning, knowledge representation, and bioinformatics, among others. Many problems from these domains have natural propositional encodings on which MaxSAT is particularly effective, often more effective than other constraint optimization paradigms such as integer programming (see, e.g., [BJ17] and the results of [BJM14] in comparison with those of [PFL14]). Section 24.3.2 discusses some of these applications of MaxSAT and provides references to the relevant literature.

A series of MaxSAT evaluations [ALMP08, HLdGS08, ALMP11, ABJM17, BJM18, BJM19], running yearly since 2006 has supported the development of increasingly robust and efficient MaxSAT solvers containing new algorithmic insights. The evaluations provide a yearly snapshot of the current state of the art in MaxSAT solving, and also provide standard benchmark sets for the scientific evaluation of new algorithmic ideas for MaxSAT. The evaluation series is focused on complete MaxSAT solving, i.e., approaches that provide optimal solutions. Recently, however, the evaluations have also added a track for incomplete solvers, with the motivation of developing approaches which can rapidly find good—though not necessarily optimal—solutions.

Complementing Chapter 23, where various branch-and-bound approaches to MaxSAT solving are described (e.g., [HLO08, LMMP10]), the focus of this chapter is on recent algorithmic advances in MaxSAT solving. In particular, we overview MaxSAT solving approaches that utilize iterative SAT solver calls, namely, the model-improving, the core-guided, and the implicit hitting set approaches to MaxSAT solving, which at the time this chapter was written constitute the main algorithmic approaches for solving large MaxSAT instances arising from real-world domains.

The model-improving approach is based on querying a SAT solver for a solution of lower cost until the solution cost can no longer be improved. Cardinality constraints (Section 24.4.2.1) are used to force the SAT solver to find a lower cost solution. The core-guided and implicit hitting set approaches iteratively query a SAT solver for unsatisfiable cores , which are subsets of soft clauses of which at least one must be falsified by any solution. In the core-guided approaches, these cores are used to construct cardinality constraints that allow the SAT solver to falsify one, but no more than one, soft clause from each core. If the problem remains unsatisfiable, a new core will be found and in the next call the SAT solver will be additionally allowed to falsify one, but no more than one, soft clause of this new core. Eventually, no more cores will be found, i.e., the problem will become satisfiable and the solution returned by the SAT solver will falsify one soft clause from each core. Since every solution must falsify at least one soft clause from every core the returned solution will be optimal.

In the implicit hitting set approach cardinality constraints are not used, which in practice makes the SAT solver calls faster. Rather a minimum-cost hitting set of the accumulated set of cores is computed. The cost of this hitting set provides a lower bound on the cost of optimal solutions, as every solution must incur the cost of at least one soft clause from every core (i.e., it must falsify at least one soft

clause from every core). The SAT solver is then asked to find a solution satisfying all of the soft clauses not in the computed hitting set. If it succeeds that solution must be optimal as it will have a cost equal to the lower bound; and if it does not it will find another core, augmenting the set of cores, and a new hitting set can be computed.

These two approaches rely on assumption-based SAT solving to extract cores when no satisfying solution can be found. Assumption-based SAT solving will be described in Section 24.4.1.2; for further details various aspects of modern SAT solvers we refer the interested reader to Chapter 4.

Beyond overviewing these recent algorithmic advances, we also overview some of the recent successful applications of MaxSAT solving, and briefly discuss some further advances in the field, including preprocessing, parallel solving, and incomplete solving.

In the rest of this chapter, we first formalize the MaxSAT problem. Then using this formalism we discuss some encoding techniques that can be used when one wants to encode different problems into MaxSAT, and give an overview of a range of different application problems that have been solved by encoding to MaxSAT and using MaxSAT solvers (Section 24.3). With this motivation of the usefulness of MaxSAT, we turn to the bulk of the chapter: a presentation of recent algorithmic techniques for solving MaxSAT, preceded by some additional needed background (Section 24.4). Finally, we close the chapter with a discussion of other recent developments (Section 24.5) and a summary and outlook on MaxSAT.

24.2. The MaxSAT Formalism

As with SAT, in MaxSAT we deal exclusively with propositional formulas expressed in conjunctive normal form (CNF): each propositional variable and its negation are **literals**; a clause is a disjunction of literals; and a CNF formula is a conjunction of clauses. Since duplicated disjuncts and conjuncts can be discarded a clause can also be viewed as being a set of literals, and a CNF formula as being a set of clauses.

Given a CNF formula F we use $vars(F)$ to denote the set of propositional variables appearing in F. A truth assignment π for F assigns to a truth value (*true* or *false*) to each propositional variable $p \in vars(F)$. For a literal l, π makes l *true* ($\pi \models l$) if l is the variable p and $\pi(p) = true$ or if l is $\neg p$ and $\pi(p) = false$. For a clause c, π satisfies c ($\pi \models c$) if π makes at least one of the literals l in c *true*; and π **satisfies** F (or is a model of F) ($\pi \models F$), if it satisfies every clause of F. Given a set of propositional variables $A \subseteq vars(F)$, and a truth assignment π for F we denote the **restriction** of π to A by $\pi|_A$; $\pi|_A$ is a new truth assignment identical to π but only assigning truth values to the variables of A.

A MaxSAT formula F is a CNF formula that is partitioned into hard and soft clauses: $F = hard(F) \cup soft(F)$. The hard clauses must be satisfied, while the soft clauses can be falsified at a cost. In particular, each soft clause $c \in soft(F)$ has an associated positive integral weight $wt(c)$ specifying the cost of falsifying that clause.[1] For example, $F = \{(x, y), (\neg x, r, z), (\neg x)_2, (\neg y)_5\}$ is a

[1] It is possible to generalize the framework to accommodate rational valued weights, and

MaxSAT formula where the first two clauses are hard and the third and fourth clauses are soft with weights 2 and 5 respectively. We will use subscripted clauses to indicate that they are soft, where the subscript is the soft clause's weight.

A **feasible solution** of F is a truth assignment to the variables of F, $vars(F)$, that satisfies $hard(F)$. The cost of a feasible solution π of F is the sum of the weights of the soft clauses it falsifies: $cost(\pi, F) = \sum_{\{c|c \in soft(F) \wedge \pi \not\models c\}} wt(c)$. When F is clear from the context we often write just $cost(\pi)$ to denote $cost(\pi, F)$.

An **optimal solution** π of F is a feasible solution with minimum cost: $cost(\pi, F) \leq cost(\pi', F)$ for all feasible solutions π' of F. An optimal solution could equivalently be defined as a feasible solution *satisfying a maximum weight of soft clauses*. We will also use $cost(F)$ to denote the cost of an optimal solution of F, and if H is a set of soft clauses $cost(H)$ to denote the sum of the weights of its clauses $\sum_{c \in H} wt(c)$.

Definition 1 (The MaxSAT problem)**.** Given a MaxSAT formula F find one of its optimal solutions.

Note that F has feasible solutions if and only if $hard(F)$ is satisfiable. Thus F will have some optimal solutions if and only if $hard(F)$ is satisfiable. *In the rest of this chapter, we will assume that $hard(F)$ is satisfiable.* In practice, one can check the satisfiability of $hard(F)$ with a SAT solver prior to searching for an optimal solution. In fact, such a check is the first step of many MaxSAT algorithms.

Another useful concept is that of a **core**. In SAT, a core is defined to be an unsatisfiable subset of the input formula. In MaxSAT, since the hard clauses must always be satisfied, it is more useful to define a core as follows.

Definition 2 (MaxSAT core)**.** Given a MaxSAT formula F, any subset K of $soft(F)$ such that $K \cup hard(F)$ is unsatisfiable, is a **core** of F. That is, K is a core if and only if every feasible solution of F falsifies at least one soft clause of K.

Historically, restricted forms of MaxSAT have been defined. In particular, a *partial* MaxSAT formula F is one in which $hard(F)$ is non-empty and a *unweighted* MaxSAT formula F is one in which all the soft clauses have the same weight (which, without loss of generality, can be assumed to be one). Thus four different types of MaxSAT formulas were traditionally recognized: (1) unweighted MaxSAT, (2) weighted MaxSAT, (3) unweighted partial MaxSAT, and (4) weighted partial MaxSAT. The definition given above corresponds to the most general case (4). In recent years the importance of these distinctions has faded as most MaxSAT solvers can handle the most general input. It is still the case, however, that whether or not the instance is weighted or unweighted has an impact on which algorithms perform better. Hence, the yearly MaxSAT evaluations use different tracks for weighted and unweighted instances.

some MaxSAT solvers (e.g., the implicit hitting set solvers described in Section 24.4.6) can solve problems containing soft clauses with floating point weights.

24.3. Encodings and Applications

Significant performance improvements in MaxSAT algorithms over the years have led to an increasing number of applications of MaxSAT. To provide the reader with a better sense of the practical usefulness of MaxSAT, we first review some of these applications as well as some of the encoding techniques often applied to translate optimization problems into MaxSAT.

24.3.1. Encodings

Boolean optimization problems that can be expressed with linear constraints and a single optimization function are well-suited to be encoded and solved by MaxSAT solvers. However, it can be the case where the problem requires solving a multi-objective optimization function, or that the problem requires soft constraints to be modeled as a group of soft clauses instead of individual soft clauses. In those cases, MaxSAT can still be used with the encodings described in Sections 24.3.1.1 and 24.3.1.2, respectively. Moreover, problems that are originally encoded in different declarative optimization languages and approaches, such as MaxCSP, MinSAT, and pseudo-Boolean constraints can also be translated into MaxSAT, as overviewed in Section 24.3.1.3.

24.3.1.1. Boolean Multilevel Optimization

The class of Boolean optimization problems that can be represented using multi-objective functions with a predefined hierarchy of importance over the objective functions is known as *Boolean multilevel optimization (BMO)* [MAGL11].

Definition 3 (Boolean multilevel optimization)**.** Boolean multilevel optimization (BMO) problems [MAGL11] consist of a set of sets of soft clauses $C = \{C_1, \ldots, C_m\}$, where $\{C_1, \ldots, C_m\}$ forms a partition of C, and each clause $c \in C_i$ has the same weight w_i. A MaxSAT formula is an instance of BMO if and only if the following condition holds:

$$w_i > \sum_{i+1 \leq j \leq m} w_j \dot{|} C_j | \qquad \text{with} \quad i = 1, \ldots, m-1.$$

In words, in BMO problems it is more valuable to satisfy a single soft clause in the set C_i than to falsify all soft clauses in the subsequent sets $C_{i+1}, \ldots, C_m$.

Example 1. Consider the instance

$$\begin{aligned} F = &\{(\neg x_1, x_2), (\neg x_1, x_4), (\neg x_1, \neg x_5), (\neg x_3, x_2, x_4), \\ &(\neg x_1)_1, (\neg x_2)_1, (\neg x_3)_1, (\neg x_4)_1, (\neg x_5)_1\}. \end{aligned}$$

Assume that instead of all soft clauses having the same priority, the user wants to give priority to satisfying some of the soft clauses over others. For example, say the user wants to solve a multilevel optimization problem where they have three objectives specified by $O_1 = \{(\neg x_1)\}$, $O_2 = \{(\neg x_2)\}$, and $O_3 = \{(\neg x_3), (\neg x_4), (\neg x_5)\}$,

with O_1 being strictly more important than O_2 and O_2 being strictly more important than O_3. This problem can be solved with MaxSAT by associating suitable weights to each soft clause.

In this example, considering the formula F and the ordered optimization objectives O_1, O_2 and O_3 we would transform F into F^{BMO} by associating all soft clauses in O_3 with the weight 1. All soft clauses in O_2 with the weight equal to the sum of the weights of the soft clauses in O_3 plus 1 ($3+1=4$). Soft clauses in O_1 will have the weight 8 which corresponds to the sum of the weights of the soft clauses in O_2 and O_3 plus 1 ($3+4+1=8$). The intuition behind these weights is that the cost of falsifying a single soft clause in O_1 is greater than falsifying all soft clauses in O_2 and O_3. Therefore, these costs capture the lexicographical order between the objective functions: $O_1 > O_2 > O_3$. We arrive at the instance

$$\begin{aligned} F^{\text{BMO}} =\{&(\neg x_1, x_2), (\neg x_1, x_4), (\neg x_1, \neg x_5), (\neg x_3, x_2, x_4), \\ &(\neg x_1)_8, (\neg x_2)_4, (\neg x_3)_1, (\neg x_4)_1, (\neg x_5)_1\}. \end{aligned}$$

F^{BMO} can be solved using MaxSAT and the optimal solution to this problem corresponds to the optimal solution to the multilevel optimization problem described by the ordered objectives O_1, O_2 and O_3.

24.3.1.2. Group MaxSAT

Some applications may require achieving more complex soft constraints that can only be represented with a set (or group) of clauses. In this case, the soft constraint is only satisfied if *all* of the clauses used to represent it are satisfied; i.e., the cost of the soft constraint will be incurred if any of its clauses are falsified. These kinds of soft clauses give rise to the *group MaxSAT* problem [HMM15]. In group MaxSAT there are k soft constraints where the i soft constraint is encoded by a group G_i of clauses. The goal in group MaxSAT is to find a solution that satisfies all hard clauses while maximizing the total weight of the satisfied soft constraints.

Group MaxSAT formulas can be transformed into an equivalent MaxSAT formulas by using specific encodings. One of these encodings is known as the $\top-$encoding [HMM15]. We refer the reader interested in other encodings of group MaxSAT to [HMM15].

Definition 4 ($\top-$encoding [HMM15])**.** A group MaxSAT instance F^{G} can be transformed into an equivalent MaxSAT instance F by adding all hard clauses from F^{G} to F. Let the i'th soft constraint, with weight w_i, be represented by the group of soft clauses $G_i = c_1, \ldots, c_k$. For each $c \in G_i$ we add the hard clause $(c \vee r_i)$ to F where r_i is a brand new variable. Finally, for each group G_i, we include the soft clause $(\neg r_i)_{w_i}$ to F.

Example 2. Consider the group MaxSAT instance

$$F^{\text{G}} =\{(\neg x_1, x_2), (\neg x_1, x_5), (\neg x_1, \neg x_4), (\neg x_3, x_2, x_4)\},$$

where

$$G_1 = (\{(\neg x_3), (\neg x_4), (\neg x_5)\})_2,$$
$$G_2 = (\{(\neg x_2), (\neg x_1)\})_3.$$

This formula can be transformed into an equivalent MaxSAT formula F by using the $\top-$encoding, resulting in

$$\begin{aligned} F = \{&(\neg x_1, x_2), (\neg x_1, x_5), (\neg x_1, \neg x_4), (\neg x_3, x_2, x_4), \\ &(\neg x_3, r_1), (\neg x_4, r_1), (\neg x_5, r_1), (\neg x_2, r_2), (\neg x_1, r_2) \\ &(\neg r_1)_2, (\neg r_2)_3\}. \end{aligned}$$

Note that the number of soft clauses of the transformed MaxSAT formula corresponds to the number of groups in the original group MaxSAT instance.

24.3.1.3. Encoding other Constraint Optimization Languages to MaxSAT

The algorithmic advances of MaxSAT make it appealing to encode problems from other domains into MaxSAT and solve them with MaxSAT solvers.

For example, other optimization problems such as the maximum constraint satisfaction problem (MaxCSP) [ACLM12] and minimum satisfiability (MinSAT) [ZLMA12, LZMS12] can be reduced to MaxSAT and solved with MaxSAT solvers.

Even decision problems such as SAT can be reduced to Horn MaxSAT by using the dual-rail encoding [IMM17]. This reduction coupled with MaxSAT resolution (Section 24.4.1.1.2) constitutes a proof system that has polynomial-time refutations for pigeonhole formulas. We refer the interested reader to [MIM17] for more examples of reductions of decision and optimization problems into Horn MaxSAT.

A common reduction is to transform a pseudo-Boolean optimization problem to MaxSAT. The interested reader is referred to Chapter 28 for more details on pseudo-Boolean optimization.

Definition 5 (Pseudo-Boolean optimization, PBO)**.** Pseudo-Boolean optimization problems are of the following form:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j l_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} l_j \leq b_i \quad \forall i = 1, \ldots, m, \end{aligned}$$

where $n, m \in \mathbb{N}$ and $a_{ij}, b_i, c_j \in \mathbb{Z}$.

Example 3. Consider the PBO problem instance F^{PBO}

$$\begin{aligned} \text{minimize} \quad & x_1 + x_2 + x_3 + x_4 + x_5 \\ \text{subject to} \quad & \neg x_1 + x_2 \geq 1 \\ & \neg x_1 + x_5 \geq 1 \\ & \neg x_1 + \neg x_4 \geq 1 \\ & \neg x_3 + x_2 + x_4 \geq 1 \end{aligned}$$

The formula F^{PBO} can be encoded to an equivalent MaxSAT formula F. The constraints are encoded as hard clauses by using appropriate cardinality and pseudo-Boolean encodings 24.4.2.1. The objective function can be encoded as soft clauses: for each literal $c_j x_j$ in the objective function, create a corresponding unit soft clause $(\neg x_j)_{c_j}$.

F^{PBO} is equivalent to the following MaxSAT formula

$$\begin{aligned} F = \{&(\neg x_1, x_2), (\neg x_1, x_5), (\neg x_1, \neg x_4), (\neg x_3, x_2, x_4), \\ &(\neg x_3)_1, (\neg x_4)_1, (\neg x_5)_1, (\neg x_2)_1, (\neg x_1)_1\}. \end{aligned}$$

Notice that this is the same formula as the one presented in Example 1.

Note, however, that this example has a particularly simple translation to MaxSAT since all of the linear inequalities can be directly represented as clauses. In general, a linear constraint of the form $a_1 x_1 + a_2 x_2 + \cdots + a_k x_k \geq t$ would have to be translated to CNF using a pseudo-Boolean constraint encoding (Section 24.4.2.1) which would add multiple clauses to the resultant MaxSAT formula.

24.3.2. Applications

With significant performance improvements in MaxSAT algorithms over the years there has been a corresponding increase in the number and diversity of applications of MaxSAT. Some examples of domains where MaxSAT has been successfully used include planning, scheduling and configuration problems, AI and data analysis problems, combinatorial problems, verification and security, and bioinformatics. We will briefly outline some of these applications and provide references to further details.

24.3.2.1. Planning, Scheduling, and Configuration

Planning problems [ZB12, MBM16] consist of finding a sequence of actions that transform an initial to a final state. These problems can be encoded to MaxSAT to efficiently find solutions that minimize the cost of the actions required to achieve the final state. MaxSAT has also been used to synthesize linear temporal logic (LTL) specifications for robot motion planning and control of autonomous systems in scenarios where specifications may be unrealizable and these can be split into soft and hard specifications [DGT18].

Scheduling problems are widespread and arise in various contexts. MaxSAT has been applied to solving different types of scheduling problems, including course timetabling [DM17, MNRS17, AN14], staff scheduling [DMW17, BGSV15, CHB17], and even wedding seating arrangements [MS17]. The course timetabling problem consists of creating a schedule such that it satisfies various constraints with respect to teachers, rooms, times, lectures and students. These scheduling problems are often solved manually by a school administrator but can be encoded into MaxSAT to lever the algorithmic advances for MaxSAT.

MaxSAT has also been used to solve configuration problems such as finding a pre-production vehicle configuration [MKTR16], solving the package upgradeability problem where the goal is to find new packages to be installed in the current system according to some user preference [ALM09, ABL+10b, IJM14], and even for seating arrangements at weddings [MS17].

24.3.2.2. AI and Data Analysis Problems

MaxSAT has been used in a variety of data-oriented AI problems, starting with computing most probable explanations (MPE) [Par02] in Bayesian networks, where the goal is to compute the most likely state of a Bayesian network given observations on a subset of the random variables. MaxSAT has been shown to yield competitive approaches to learning different classes of probabilistic graphical models, including Bayesian network structures with restricted treewidth [BJM14], causal discovery in very generic model spaces [HSJ17], as well as causal structure estimation from time series data [HPJ+17]. MaxSAT has also been proposed as an alternative approach to inferring problem-specific cutting planes in a state-of-the-art integer programming approach to Bayesian network structure learning [SMJ15]. Further recent data-oriented applications of MaxSAT include learning explainable decision sets [IPNM18] and interpretable classification rules [MM18], constrained correlation clustering [BJ13, BJ17], and neighborhood-preserving low-dimensional visualization of high-dimensional datasets [BJB+14].

Other AI applications, specifically within the area of knowledge representation and reasoning, consist of applying MaxSAT to understand the dynamics of argumentation frameworks [WNJ17, NWJ16a, NWJ16b] and in model-based diagnosis [MJIM15]. Given a model of a system and an input-output observation that is not consistent with the expected behavior, the goal of model-based diagnosis is to identify a subset of components that when removed make the system consistent with the observation. MaxSAT can be used to find the smallest set of components that need to be removed [MJIM15]. In the realm of abstract argumentation, MaxSAT enables efficiently reasoning about different notions of *enforcement*, dealing with smallest changes required to argumentation frameworks in light of knowledge on extensions [WNJ17, NWJ16a] as well as *synthesis* of argumentation frameworks based on negative and positive examples of extensions [NWJ16b].

24.3.2.3. Combinatorial Problems

MaxSAT can also be used to encode and solve many combinatorial problems, such as the Max-Clique problem [LQ10, FLQ+14, LJX15] where given a group of vertices, the maximal clique is the largest subset of vertices in which each point is directly connected to every other vertex in the subset.

Other combinatorial problems that have been encoded into MaxSAT include the Steiner tree problem [dOS15], tree-width computation [BJ14] and finding solutions for the maximum quartet consistency problem [MM10].

24.3.2.4. Verification and Security

Another domain with many successful MaxSAT applications is verification and security of software and hardware. For instance, MaxSAT can be used to design and debug circuits [SMV+07, CSVMS09, CSMV10, AIMT13, XRS03]. Other examples of applications of MaxSAT to hardware consist in solving the hardware-software partitioning problem which decides during the design phase which parts should be implemented either in hardware or software [TDFDGDSJ+17].

MaxSAT also has many applications in security. For instance, solving the user authorization query problem [WQL09], reconstructing AES key schedule images [LZK16], detecting hardware Trojans [SA18], and maximizing the development assurance level to improve the rigor of the development of software and hardware on aircraft [BDS11]. Another application in security is to find malware in Android applications by maximizing the similarity between a malicious application and the application that is being scanned [FBM+17].

MaxSAT has also been used to minimize cascading style sheets (CSS) in order to reduce the size of the code transmitted over the web [HLH19]. Another application is to use MaxSAT for quality of service (QoS) of cloud services where the goal is to minimize the composition of distributed web services while satisfying the user preferences [WJH18, BACF17].

When analyzing software, program analysis tools usually trade precision for scalability and their output often contains undesirable output due to approximations. MaxSAT can be used in a user-guided program analysis setting where the hard clauses capture the soundness of the analysis and soft clauses capture the degrees of approximation and user's preferences [MZNN15a, SZGN17]. Encoding user-guided program analysis into MaxSAT can lead to very large formulas which can be solved by specialized MaxSAT solving algorithms [MZNN15b, ZMNN16].

MaxSAT can also be used for fault localization where the goal is to pinpoint the localization of software bugs [ZWM11, JM11]. In particular, MaxSAT can be used to reduce the error log reported to the programmer and decrease the amount of code that needs to be reviewed. The problem of state-space reduction for non-deterministic visibly pushdown automata can also be encoded into MaxSAT, which can be used to improve the performance of software model-checkers [HST17].

24.3.2.5. Bioinformatics

MaxSAT has applications in many interdisciplinary applications such as bioinformatics. One of these applications is to solve the haplotype inference problem which aims at finding the minimum number of haplotypes which explains a given set of genotypes [GML11, GMLO11].

Other applications within this domain consist of finding generalized Ising models [HKD+16], finding the maximum similarity between RNA sequences [Mar17], modeling biological networks representing the interactions between genes [GL12], and even applications in cancer therapy to find faulty areas of the gene regulatory network [LK12].

24.3.2.6. Further Applications

Yet another application of MaxSAT is to improve other constraint solving algorithms. For instance, MaxSAT can be used to restore satisfiability of constraint satisfaction problems (CSPs). By encoding a CSP instance as a MaxSAT instance, one can identify the smallest set of tuples to be removed from the CSP instance to restore satisfiability [LM11]. MaxSAT has also been used to enumerate minimal correction sets (MCS) [MLM13] with the goal of removing a minimal number of clauses such that an unsatisfiable SAT formula becomes satisfiable.

Propagation complete encodings have the property of being able to identify inconsistencies via unit propagation, thus allowing the encoding to be more efficiently solved by SAT solvers. However, this trade-off usually comes at the expense of larger SAT encodings. MaxSAT can be used to compute approximately propagation complete encodings [ER18] which allows conflicts to be detected earlier when using a SAT solver.

24.4. Modern MaxSAT Algorithms

In this section, we provide an overview of the currently most important algorithmic approaches to MaxSAT, focusing on SAT-based approaches: the model-improving approach (based on querying a SAT solver for solutions of increasing quality), the core-guided approach (in its several variants, based on extracting MaxSAT cores and compiling each core into the instance until a satisfying assignment is found), and the implicit hitting set approach (based on accumulating a set of MaxSAT cores and computing a minimum-cost hitting set of the cores until a satisfying assignment for the clauses not in the hitting set is found).

We will not cover branch-and-bound approaches to MaxSAT (e.g., [HLO08, LMMP10]). The interested reader is referred to Chapter 23 for an overview of such approaches. The main feature of the approaches we will present here is that they scale up to much larger problems than branch-and-bound approaches. There do exist small MaxSAT instances with a few hundred variables (e.g., finding largest cuts in random graphs) that branch-and-bound algorithms can solve very quickly (less than ten seconds) while the modern SAT-based MaxSAT algorithms overviewed in this chapter have considerable problem solving (i.e., instances remain unsolved after thousands of seconds). However, branch-and-bound approaches are typically ineffective on instances with more than a thousand variables. Problems from various application areas, including ones described in Section 24.3.2, are usually much larger, typically involving at least ten thousand variables and sometimes as many as millions of variables and clauses.

24.4.1. Background

While details on the MaxSAT formalism were already presented in Section 24.2, in order to properly explain current algorithms for solving MaxSAT we need some further background.

24.4.1.1. Transformations

MaxSAT algorithms rely on certain transformations that can be applied to a MaxSAT formula. There are a number of sound ways to transform a MaxSAT formula F into a new MaxSAT formula F' such that from an optimal solution to F' we can easily obtain an optimal solution for F. Such transformations include simplifying transformations used in preprocessing, and transformations used simply to make it more algorithmically convenient to solve the MaxSAT problem. Preprocessing is discussed in more detail in Section 24.5.1. In this section we present some more basic transformations that are useful for algorithmic convenience (both for MaxSAT solving and for further preprocessing).

24.4.1.1.1. The Blocking Variable Transformation. The blocking variable transformation is used on the input formula by a number of MaxSAT solvers. It transforms an input MaxSAT formula F into an equivalent formula F^b that has only unit soft clauses. Having a single literal ℓ whose truth value encodes the truth value of the soft clause c, allows us to use ℓ as an assumption to the SAT solver (Section 24.4.1.2) and to use ℓ as an input to cardinality constraints used when encoding the cost bound k.

In the F^b transformation, the hard clauses F are unchanged. Each soft clause c_i of F is transformed by adding the negation of a brand new variable b_i,[2] adding the resultant extended clause $c_i \vee \neg b_i$ as a new hard clause, and creating a new unit soft clause (b_i) with weight equal to c_i's weight (c_i's weight is moved to the new unit soft clause).

Definition 6 (Blocking variable transformation)**.** Given a MaxSAT instance F, F^b is a new MaxSAT instance with:

1. $hard(F^b) = hard(F) \cup \{(c_i \vee \neg b_i) \mid c_i \in soft(F)\}$, where each b_i is a new variable called a **blocking** or a **relaxation** variable.
2. $soft(F^b) = \{(b_i) \mid \ b_i$ was added in step 1$\}$
3. $wt((b_i)) = wt(c_i)$ (each new soft clause (b_i) gets the weight of the original $c_i \in soft(F)$).

Example 4. If $F = \{(x, \neg y, z), (\neg y, z), (z, y)_{10}, (\neg z)_5\}$, then $F^b = \{(x, \neg y, z), (\neg y, z), (z, y, \neg b_1), (\neg z, \neg b_2), (b_1)_{10}, (b_2)_5\}$.[3]

F^b has the following useful properties.

1. F^b has only unit soft clauses, so there is a single literal denoting the truth and falsity of each soft clause.
2. If π^b is a feasible solution for F^b then $\pi^b|_{vars(F)}$ is a feasible solution for F with $cost(\pi^b, F^b) \geq cost(\pi^b|_{vars(F)}, F)$.
3. If π is a feasible solution for F, then we can extend π to π^b, a feasible solution for F^b, with $cost(\pi, F) = cost(\pi^b, F^b)$.

To illustrate property 2, $\pi^b = \{x, \neg y, z, \neg b_1, \neg b_2\}$ (listing the literals made *true* by π^b) is a feasible solution for F^b from Example 4, and $cost(\pi^b, F^b) = 15$. Restricting π^b to $vars(F)$ we obtain $\pi^b|_{vars(F)} = \{x, \neg y, z\}$, which can be seen to be a feasible solution to F with lower cost $cost(\pi^b|_{vars(F)}, F) = 5$.

To illustrate property 3, $\pi = \{x, \neg y, z\}$ is a feasible solution of F with $cost(\pi, F) = 5$, and we can extend it to be a feasible solution of F^b by setting each b_i to *true* iff $\pi \models c_i$. In this case, we obtain $\pi^b = \{x, \neg y, z, b_1, \neg b_2\}$ with equal cost $cost(\pi^b, F^b) = 5$.

Proposition 1. *Properties 2 and 3 hold for the blocking variable transformation.*

[2] The idea of adding a new variable to a clause so that the new variable encodes the clause's truth value appears in [GW93], but may have been used even earlier.

[3] For some MaxSAT algorithms it is possible in certain cases to avoid adding a new variable to soft clauses that are already unit [BSJ15b]. This can significantly reduce the number of new variables in the formula F^b making it more efficient to solve.

Proof. To see that property 2 is true, observe the following. If π^b is a feasible solution of F^b it must satisfy all hard clauses of F^b. Hence, π^b satisfies every hard clause of F since $hard(F) \subseteq hard(F^b)$. Since none of the hard clauses of F contain any new b_i variables we also have that $\pi^b|_{vars(F)}$ satisfies every hard clause of F, and thus $\pi^b|_{vars(F)}$ is a feasible solution for F. Consider $cost(\pi^b|_{vars(F)}, F)$. This is the sum of the weights of all soft clauses falsified by $\pi^b|_{vars(F)}$. Let c_i be one of the soft clauses falsified by $\pi^b|_{vars(F)}$ so that weight $wt(c_i)$ is a term in the sum $cost(\pi^b|_{vars(F)}, F)$. Since π^b satisfies the hard clause $(c_i \vee \neg b_i)$ of F^b and $\pi^b|_{vars(F)}$ falsifies c_i, we must have that $\pi^b \models \neg b_i$. In other words, π^b must falsify the soft clause $(b_i) \in F^b$, which contributes the term $wt((b_i))$ $(= wt(c_i))$ to $cost(\pi^b, F^b)$. That is, we must have that $cost(\pi^b, F^b) \geq cost(\pi^b|_{vars(F)}, F)$.

To see that property 3 is true, let π be a feasible solution for F. We extend π to π^b by setting $\pi^b \models b_i$ if $\pi \models c_i$ and $\pi^b \models \neg b_i$ if $\pi \not\models c_i$. As a result π^b will satisfy all hard clauses of F^b. Also, $cost(\pi^b, F^b)$ will include the term $wt((b_i))$ if and only if $cost(\pi, F)$ includes the term $wt(c_i)$ $(= wt(b_i))$. Hence $cost(\pi, F) = cost(\pi^b, F^b)$ □

Furthermore, properties 2 and 3 imply that the blocking variable transformation MaxSAT equivalence in the following sense.

Proposition 2 (MaxSAT equivalence of F^b)**.** *If π^b is an optimal solution of F^b then $\pi^b|_{vars(F)}$ is an optimal solution of F.*

Proof. The proposition follows from properties 2 and 3 on the blocking variable transformation. In particular (2) tells us that the cost of an optimal solution to F^b is at least as large as the cost of an optimal solution to F; and (3) tells us that that is at least as small. □

This proposition tells us that we can solve a MaxSAT formula F by solving F^b and then simply discarding the truth assignments to the newly introduced variables b_i.

Historically, the idea of adding "relaxation" or "blocking" variables to the soft clauses has been used in all MaxSAT algorithms based on solving a sequence of SAT instances. In the presentation of these algorithms, typically the relaxation variables are introduced at various stages of the algorithm's execution rather than initially. In practice, however, MaxSAT solvers generally introduce the relaxation variables at the beginning, so as to exploit SAT solving under assumptions to extract cores, or to impose bounds on how many soft clauses can be falsified via cardinality or pseudo-boolean constraints. In general, it is simpler and more uniform to explain these different algorithms as they run in practice—i.e., as they run on F^b rather than as they run on F.

Also typically the positive form of the relaxation variables is added to the clause. So the soft clause $(x, y)_{10}$ would become the hard clause (x, y, b_1), rather than $(x, y, \neg b_1)$ as we have done here. By adding the negation of the relaxation variable we avoid a potentially confusing polarity flip. In particular, now when b_i is *true* c_i must be satisfied, rather that having $\neg b_i$ imply that c_i must be satisfied.

24.4.1.1.2. Soft Clause Cloning and MaxSAT Resolution. Another transformation that is sometimes used when solving a MaxSAT formula is to split a soft clause c_{m+n} into two soft clauses c_m and c_n. For example, we can replace the single soft clause $(x, \neg y)_{12}$ with the two soft clauses $(x, \neg y)_5$ and $(x, \neg y)_7$. In the literature, this process is called soft clause cloning. It is not difficult to see that the cost of any feasible solution π remains unchanged by soft clause cloning (and hence the optimal solutions are unchanged). This transformation can also be performed in reverse, we can combine the weights of all copies of a soft clause into a single copy. Soft clause cloning is used in many (but not all) MaxSAT algorithms to deal with weighted instances.

Lifting the classical resolution rule to MaxSAT gives rise to a more complex transformation [LH05, BLM07]. The Larrosa and Heras [LH05] transformation is as follows. Given two clauses of the form $(x, A)_{w_1}$ and $(\neg x, B)_{w_2}$ and letting $m = \min(w_1, w_2)$ we can apply a MaxSAT resolution step as follows.

1. Replace the two original clauses $(x, A)_{w_1}$ and $(\neg x, B)_{w_2}$ by their residues: $(x, A)_{w_1 - m}$ and $(\neg x, B)_{w_2 - m}$.
2. Add the resolvant $(A, B)_m$.
3. Add the two compensation clauses $(x, A, \neg B)_m$ and $(\neg x, B, \neg A)_m$.

There are a few things to note. First, any soft clause with weight zero can be removed from the formula—such clauses cannot affect the cost of any feasible solution. Hence, at least one of the residue clauses will vanish. Second, the compensation clauses are not clauses (they contain conjunctions of literals $\neg A$ or $\neg B$). Hence, they have to be converted into a set of clauses, where the size of that set will depend on whether or not new variables are introduced. And third, by treating hard clauses as being soft clauses with infinite weight (e.g, $(x, y) = (x, y)_\infty$), we can apply MaxSAT resolution to combinations of soft and hard clauses.

Since the MaxSAT resolution rule of Larrosa and Heras can generate non-clausal formula it cannot form a proof system. In any proof system for CNF each rule of inference must yield a valid new CNF. Bonet et al. [BLM07] addressed this issue and gave a version of MaxSAT resolution that does yield a new CNF. They also showed that their version of MaxSAT resolution forms a complete proof system: for any MaxSAT instance F there exists a MaxSAT resolution proof that F has optimal cost $cost(F)$. MaxSAT resolution has been used as a theoretical underpinning for some MaxSAT algorithms, e.g., PMRes [NB14] and MiniMaxSat [HLO08], but directly implementing MaxSAT resolution to solve MaxSAT does not seem to be practical [MM11].

24.4.1.2. Incremental and Assumption-based SAT Solving for Extracting Cores

An essential technique used in MaxSAT solving is assumption-based SAT solving [ES03] implemented in most modern SAT solvers. When given an input formula F a SAT solver can produce either a satisfying truth assignment or conclude that F is unsatisfiable. Assumption-based SAT solving extends the capacity of the SAT solver by asking it to solve F subject to a set of assumptions A which must be a set of literals. Now the SAT solver must either find a truth assignment satisfying $F \wedge \bigwedge_{\ell \in A} \ell$ (i.e., a truth assignment satisfying F that also makes all of the literals

in A *true*), or it must conclude that $F \wedge A$ is unsatisfiable. Furthermore, and most critical to MaxSAT solvers, when $F \wedge A$ are unsatisfiable the SAT solver is able to return a clause c such that

1. c contains only negated literals of A, i.e., $A \models \neg c$ (c is a conflict clause over A), and
2. $F \models c$.

The conflict clause c is thus a subset of A that is sufficient to show that no model of F satisfies all of A. In particular any model of F must falsify at least one of the literals of A whose negation is contained in c.

It should be noted that the conflict clause c returned by the SAT solver need not be minimal. That is, there could be another clause $c' \not\subseteq c$ satisfying the above two conditions. Hence, if F by itself is unsatisfiable the SAT solver might return any subset of negated literals of A (including possibly the empty clause).

Assumptions can be used to extract cores when the MaxSAT formula contains unit soft clauses (e.g., after the F^b transformation has been applied). In particular, if $(b_1)_{w_1}, \ldots, (b_m)_{w_m}$ are the soft clauses of a MaxSAT formula F then using $\{b_1, \ldots, b_m\}$ as assumptions in a SAT solve of $\mathit{hard}(F)$ will either result in a model of $\mathit{hard}(F)$ that satisfies all of the soft clauses of F, i.e., a zero-cost model, or it will result in a conflict clause c such that $F \models c$ and $c \subseteq \{\neg b_1, \ldots, \neg b_m\}$. That is, the conflict clause c asserts that at least one of the negated b_i literals in it must be *true* in any feasible model. In other words, c specifies a set of soft clauses at least one of which must be falsified by any feasible model, i.e., a core of F.

Assumptions also play a useful role in *incremental* SAT solving. In many approaches to MaxSAT solving a SAT solver is called on a sequence of problems that are closely related to each other. Each of problems could be solved by invoking a new instance of the SAT solver. However, then information computed during one SAT solver invocation (e.g., learnt clauses) cannot easily be exploited in subsequent solver invocations. The idea of incremental SAT solving is to use only one instance of the SAT solver for all of the problems so that all information computed can be retained for the next problem. In this case, we can monotonically add clauses to the SAT solver or use different sets of assumptions to specify each new problem to be solved. With assumptions we can add or remove certain clauses by adding to those clauses a new literal ℓ. When we assume $\neg\ell$ these clauses become active (added to the problem), and when we assume ℓ these clauses become inactive (removed from the problem).

24.4.2. Approaches based on Sequences of Satisfiability Queries

MaxSAT with weighted soft clauses is known to be in the complexity class $\mathrm{FP}^{\mathrm{NP}}$ [Pap94]. This means that it can be solved with a polynomial number of calls to an NP-oracle. In particular, a polynomial number of calls to a SAT solver can be used to solve the MaxSAT instance F, with each SAT call determining if F can be satisfied by a feasible solution with cost less than or equal to k, for various values of k. With integer weights, if F can be satisfied with cost k and F cannot be satisfied with cost $k-1$, then the feasible model achieving cost k must be an optimal solution.

Given the effectiveness of modern SAT solvers, the approach of solving MaxSAT via a sequence of calls to a SAT solver has proved to be both popular and effective. A number of different techniques have been developed for effectively encoding sequences of the following decision problem $\mathcal{D}(F, k)$, for various values of k, into CNF.

Definition 7 (The decision problem $\mathcal{D}(F, k)$)**.** Let $\mathcal{D}(F, k)$ be the decision problem "does a feasible solution of F with cost less than or equal to k exist?"

In particular, we want to encode $\mathcal{D}(F, k)$, into a CNF formula that can be effectively solved by a SAT solver. The key to many of these encoding techniques is the successful exploitation of information returned by the SAT solver when it either determines $\mathcal{D}(F, k)$ to be unsatisfiable or satisfiable. In particular, when $\mathcal{D}(F, k)$ is satisfiable the cost of the model found can be used to determine the next value of k to test; and when the problem is unsatisfiable the returned **core** can be used to add additional constraints to the SAT solver so as to effectively encode $\mathcal{D}(F, k')$ for some value k' larger than k.

MaxSAT solvers using a sequence of SAT queries have been designed in a number of different ways. In this section, we will survey the main MaxSAT algorithms using this approach. First, however we introduce some important features of the CNF encoding of the decision problem "does a feasible solution of F with cost less than or equal to k exist", since these features affect both the design and effectiveness of the MaxSAT algorithms discussed in this section.

24.4.2.1. Cardinality and Pseudo-Boolean Constraints

Pseudo-Boolean constraints are used to help encode $\mathcal{D}(F, k)$. A pseudo-Boolean constraint over the literals $\{\ell_1, \ldots, \ell_m\}$ has the form $a_i\ell_1 + \cdots + a_m\ell_m \leq k$ where the a_i and k are integers and the sum is evaluated by regarding $\ell_i = \mathit{true}$ as 1 and $\ell_i = \mathit{false}$ as 0.[4] When all of the a_i are 1, the pseudo-Boolean constraint is called a cardinality constraint. Cardinality constraints limit the number of literals ℓ_i that can be *true*.

There is an extensive literature on different methods for encoding pseudo-Boolean constraints and cardinality constraints, e.g. [War98, FG10, ES06, BB03, Sin05, ANOR09, OLH+13, BBR09, MPS14, HMS12, JMM15]. The details of these encodings and their different properties will not be discussed here, but we will point out some key features of these encodings that are particularly relevant to their use in MaxSAT solvers.

The first feature of these encodings is their *size*, i.e., the number of clauses and new variables needed in the CNF encoding. The main parameters influencing size are m the number of literals whose sum is being taken, k the size of the upper bound, and $\max(a_i)$ the greatest coefficient in the sum.

For cardinality constraints the commonly used CNF encodings (e.g., the totalizer encoding [BB03]) the size of the encoding grows as $O(mk)$. So in the worst case when $k = m$ this can be as large as $O(m^2)$. Smaller CNF encodings for cardinality constraints exist. For instance, the modulo totalizer encoding [OLH+13]

[4] The other relations ($=$, $\geq$, $<$, and $>$) can also be represented.

requires $O(m^{\frac{3}{2}})$ clauses, whereas the cardinality network encoding [ANOR09] requires $O(mlog_2(k))$ clauses. However, there are many MaxSAT problems with over 100,000 soft clauses. After the F^b transformation this would yield over 100,000 soft unit clauses. So we can see that naively constructing a cardinality constraint summing all of these literal might require as much as 10^{10} clauses. So MaxSAT solvers must exploit cardinality constraints in more clever ways.

For pseudo-Boolean constraints, the typical encodings are much larger. Some encodings do not necessarily depend on k but rather on m and $\log_2(\max(a_i))$ (the number of bits required to represent the coefficients in binary). There are pseudo-Boolean encodings of size as small as $O(m \log_2(\max(a_i))$ (linear in m if the size of the coefficients is bounded) [War98], and others that are of larger size $O(m^3 \log_2(m) \log_2(\max(a_i)))$ [BBR09], $O(m^2 \log_2(m) \log_2(\max(a_i)))$ [MPS14]. Other encodings may depend on k (e.g, the sequential weight counter encoding [HMS12]) and have size $O(mk)$, or depend on the number of distinct weights (which is the case for e.g. the generalized totalizer encoding [JMM15]). The tradeoff between these encodings is that the larger encodings achieve better propagation, as we discuss next.

The second relevant feature of these encoding is their propagation power. This is the extent to which unit propagation can find entailed literals. Cardinality constraint encodings typically allow unit propagation to achieve *arc consistency* [Dec03, BB03]. That is, unit propagation on the encoding is able to find all literals entailed by the cardinality constraint. Since SAT solvers learn clauses based on the set of unit propagated literals, this degree of propagation power enhances the effectiveness of the SAT solver. The watchdog encoding [BBR09] was the first polynomial size pseudo-Boolean encoding that maintains arc consistency. Since then, several other polynomial arc consistent encodings have been proposed [MPS14, HMS12]. Encodings that achieve arc consistent are typically very large. Nevertheless, if the encoding is not prohibitively large then it can have better SAT solving performance due to greater propagation power.

Finally, the third relevant feature is whether or not the encoding can be made incremental. That is, given a CNF encoding of $a_i\ell_1 + \cdots + a_m\ell_m \leq k$ can we incrementally add additional clauses to encode $a_i\ell_1 + \cdots + a_m\ell_m + a_{m+1}\ell_{m+1} \cdots a_{m+j}\ell_{m+j} \leq k'$, where we are summing over more literals and $k' \geq k$. For cardinality constraints incrementality was achieved in [MML14], and for pseudo-Boolean constraints [MJML14a, PRB18] reported on incremental ways of changing the left-hand size k to k'. In both cases making these encodings incremental yielded significant speedups for the underlying MaxSAT solver.

A critical element supporting incremental SAT solving (Section 24.4.1.2) is that some cardinality constraint encodings (e.g., the totalizer encoding [BB03]) include a set of "output" variables that encode the sum of the input literals in unary. That is, the encoding has a set of "output" variables o_i such that for any i, o_i is *true* if and only if the input literals sum to at least i. This allows using incremental SAT solving when changes are made to the the left-hand side of the constraint simply by making different assumptions. For example, given the totalizer encoding of the cardinality constraint $\sum_{j=1}^{m} \ell_j \leq k$ we can incrementally impose the constraint $\sum_{j=1}^{m} \ell_j < i$ for any $i \leq k$, by simply assuming $\neg o_i$ during the SAT call.

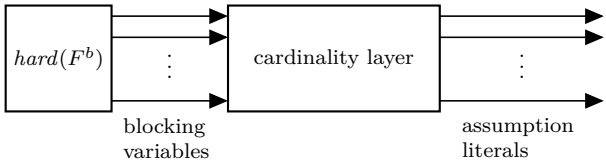

Figure 24.1: Structure of CNF for solving $\mathcal{D}(F, k)$. Note some blocking variable inputs to the cardinality layer might be passed directly through to the outputs (i.e., used directly as assumption literals).

24.4.3. General Structure of CNF Encodings of $\mathcal{D}(F, k)$

Figure 24.1 shows the general structure of the CNF formulas typically used for encoding the decision problem $\mathcal{D}(F, k)$. As already mentioned, the MaxSAT algorithms discussed in this section are all based on solving a sequence of $\mathcal{D}(F, k)$ decision problems for various values of k using a SAT solver. The encoding typically used consists of the hard clauses of the blocking variable transformation, $hard(F^b)$, conjoined with clauses of a cardinality layer. The clauses of the cardinality layer encode various cardinality constraints (or pseudo-Boolean Constraints) over the *blocking* variables of F^b, and sometimes over other internal variables of the cardinality layer.

The soft clauses of F^b are all of the form (b_i), i.e., positive instances of the newly introduced blocking variables. Typically, the cardinality layer imposes constraints on which and how many of the literals $\neg b_i$ can be *true*. That is, it imposes constraints on the set of soft clauses of F^b that the SAT solver is allowed to falsify. In addition, various literals from the cardinality layer (including perhaps some of blocking variables) can be assumed to be *true* in assumption-based SAT solving.

The SAT solver, for which all clauses are of course hard, has all of the clauses of $hard(F^b)$. Hence, all satisfying models found by the SAT solver must be feasible models of F when restricted to $vars(F)$. (Since these models are feasible solutions of F^b this follows from the properties of F^b described in Section 24.4.1.1.1). Furthermore, the SAT solver must find a model that satisfies all of the clauses of the cardinality layer and makes its current set of assumptions *true*. Of course, if no such model exists the SAT solver can return a conflict clause over its assumptions, as described in Section 24.4.1.2.

24.4.4. Model-Improving Algorithms

The first MaxSAT algorithm we present is the model-improving Linear SAT/UNSAT (LSU) algorithm. LSU follows almost directly from a proof that MaxSAT is in the complexity class FP^{NP}. That is, it uses a SAT solver to solve the sequence of decision problems $\mathcal{D}(F, k_0)$, $\mathcal{D}(F, k_1)$, ..., $\mathcal{D}(F, k_u)$, where $k_0 = \infty$ (i.e., find a feasible solution irrespective of cost), k_i is the cost of the model found when solving $\mathcal{D}(F, k_{i-1})$ minus one, and $\mathcal{D}(F, k_u)$ is the first decision problem in the sequence that was unsatisfiable (UNSAT). The feasible solution returned by

Algorithm 1: Linear SAT/UNSAT algorithm for MaxSAT [BP10]

```
1  LSU(F)
2  F^b = bv_transform(F); card_layer = {}; Blits = {¬b_i | (b_i) ∈ soft(F^b)}
3  bestmodel = {}; cost = ∞; sat? = true
4  while sat? do
       /* SAT solver returns boolean SAT/UNSAT status                    */
       /* if SAT returns π, the model found                              */
5      (sat?, π) = SATSolve (hard(F^b) ∪ card_layer)
6      if sat? then
7          bestmodel = π
8          cost = cost(π)
9          card_layer = CNF(Σ_{b_i ∈ Blits} wt((b_i)) × ¬b_i < cost)
10     else
11         return (bestmodel|_{vars(F)}, cost)
```

the last satisfiable instance $\mathcal{D}(F, k_{u-1})$ when restricted to the variables of F is an optimal solution of F.

Algorithm 1 gives the details. First any feasible model π is found (line 5) with card_layer equal to the empty set of clauses. Then using either a cardinality (when $wt((b_i)) = 1$ for all i) or pseudo-Boolean constraint (when the soft clauses have varying weights) in the cardinality layer (line 9), the SAT solver is asked to find a feasible solution with cost strictly less than $cost(\pi)$.[5] This is repeated until the SAT solver returns UNSAT, in which case the last feasible solution returned is an optimal solution.

The approach of solving a sequence of decision problems $\mathcal{D}(F, k_i)$ with decreasing bounds k_i has a long history. The earliest reported use of this approach in SAT solving appears to be [ARMS02]. The approach was also used in the Minisat+ solver described in [ES06], and in the Sat4j MaxSAT solver [BP10] where the name LSU was first used and an algorithm similar to Algorithm 1 was presented. These three systems actually found optimal models with respect to pseudo-Boolean objective functions. However, pseudo-Boolean objective functions can easily represent MaxSAT when all of the soft clauses are unit (as they are in the transformed formula F^b).

This approach continues to be used in the QMaxSAT [KZFH12] and Pacose solvers (described in [BJM18]). QMaxSAT includes a number of different cardinality and pseudo-Boolean encodings, and carefully chooses which encoding to use based on the characteristics of the input problem. Pacose builds on QMaxSAT by implementing a dynamic construction of one of its pseudo-Boolean encodings [PRB18].

Recently, the LSU algorithm has seen increasing use in incomplete MaxSAT solving in which the aim is to return the lowest-cost feasible solution within a given time bound. As can be seen from Algorithm 1 its chief benefit in this context is that each completed call to the SAT solver returns an improved solution. Hence, at any point the best model found so far can be returned.

Overall, however, the downside of this algorithm is that the cardinality or

[5] Finding a low cost initial feasible model can potentially reduce the size of the cardinality constraint encoding (see Section 24.4.2.1).

pseudo-Boolean constraint needs to be over m literals, where m is the number of soft clauses. As pointed out in Section 24.4.2.1, this can lead to very large encodings when the MaxSAT formula has tens or hundreds of thousands of soft clauses. Hence, solving the resulting SAT problem can become infeasible.

24.4.5. Core-Guided Algorithms

In contrast to LSU, core-guided algorithms work from UNSAT to SAT. That is, they solve a sequence of decision problems $\mathcal{D}(F, k_i)$ where k_i is increasing ($k_i > k_{i-1}$). When k_i is less than the cost of an optimal solution, $\mathcal{D}(F, k_i)$ will be unsatisfiable. Essentially, core-guided algorithms increase k_i until the current $\mathcal{D}(F, k_i)$ is satisfiable, but do not do this as explicitly as done in the LSU approach.

As with LSU these algorithms utilize information from the SAT solver returned from each UNSAT result, to construct the next decision problem to solve. The key insight, originally coming from Fu and Malik [FM06] is that cores can be extracted from the SAT solver when $\mathcal{D}(F, k)$ is unsatisfiable. Every feasible solution must falsify at least one soft clause of every core. Hence, the cores provide us information about the subset of soft clauses that the SAT solver can consider for falsification when solving $\mathcal{D}(F, k)$.

Empirically, many MaxSAT instances have optimal solutions falsifying very few soft clauses (relative to the total number of soft clauses). Hence MaxSAT cores often contain only a small subset of the soft clauses. So instead of having large cardinality constraints over all of the soft clauses, core-based algorithms can often utilize much smaller cardinality constraints over only those soft clauses appearing in the found cores, as these are the only ones that the SAT solver needs to consider falsifying. The other soft clauses can be required to be satisfied by using assumptions, rather than via cardinality constraints.

24.4.5.1. Fu-Malik

Fu and Malik presented the first MaxSAT algorithm exploiting cores [FM06]. Algorithm 2 gives the details. This algorithm only works on unweighted MaxSAT instances (i.e., all soft clauses have weight 1).

All blocking variables are always assumed to be *true* (line 2 and 5). Their main role is simply to allow the SAT solver to extract cores via its assumption mechanism. In the initial call, card_layer is empty. Hence, the initial call is asking the SAT solver to find a feasible model satisfying all soft clauses (line 5). In particular, every soft clause c_i of F has been encoded in F^b as the hard clause $(c_i \vee \neg b_i)$. Hence, when we assume b_i this clause is reduced to c_i, and the SAT solver must satisfy it.

Unless there is a zero-cost feasible solution, this call with result in the SAT solver declaring unsatisfiability. Furthermore, the SAT solver will also identify a subset of negated assumptions forming a core, κ. That is, at least one of the literals $\neg b_i$ in κ must be *true*. The algorithm then modifies the hard clause corresponding to each of the soft clauses in the core; for $\neg b_i$ this is the hard clause $(c_i \vee \neg b_i)$. It adds a brand new relaxation variable r_i^{cost} to each such clause by modifying the c_i part of the clause (line 8). (Note that "cost" is always incremented in the loop, line 10, so different variables are added in each iteration).

Algorithm 2: Fu-Malik algorithm for unweighted MaxSAT [FM06]

```
1  FuMalik(F)
2  F^b = bv_transform(F); card_layer = {}; assumptions = {b_i | (b_i) ∈ soft(F^b)}
3  F' = hard(F^b); cost = 0; sat? = false
4  while not sat? do
       /* SAT solver takes as input the CNF to solve and a set of literals to assume  */
       /* Returns boolean SAT/UNSAT status                                            */
       /* If SAT returns π, the model found                                           */
       /* If UNSAT returns κ, a core (subset of negated assumptions).                 */
5      (sat?, π, κ) = SATSolve (F' ∪ card_layer, assumptions)
6      if not sat? then
7          for ¬b_i ∈ κ do
8              c_i = c_i ∨ r_i^cost
               /* Modifying c_i modifies the clause (c_i ∨ b_i) ∈ F'              */
               /* Each r_i^cost is a brand new variable.                          */
9          card_layer = card_layer ∪ CNF(Σ_{i|¬b_i∈κ} r_i^cost = 1)
10         cost = cost + 1
11     else
12         return (π|_vars(F), cost)
```

Finally, it adds a cardinality constraint asserting that one and only one of the newly added relaxation variables can be made *true*. Making r_i^{cost} *true* immediately satisfies (or relaxes) the clause $(c_i \vee r_i^{\text{cost}} \vee \neg b_i)$, allowing the SAT solver to set the assumption b_i to *true* with impunity.

This algorithm does not quite follow the general structure described in Section 24.4.3. In particular, each new call to the SAT solver involves changing $hard(F^b)$. Since the change involves adding brand new variables to existing clauses (line 8), it is not easy to use the SAT solver incrementally in this algorithm.

Example 5. Consider $F = \{(x, \neg y),\ (y, z),\ (y, \neg z),\ (\neg x)_1,\ (\neg y)_1\}$. In this formula both x and y are forced, and thus both soft clauses must be falsified. Any feasible solution has cost 2, and thus all feasible solutions are optimal solutions.

$F^b = \{(x, \neg y),\ (y, z),\ (y, \neg z),\ (\neg x, \neg b_1),\ (\neg y, \neg b_2),\ (b_1)_1,\ (b_2)_1\}$. The first iteration of the main loop of Algorithm 2 would call the SAT solver on the CNF $\{(x, \neg y),\ (y, z),\ (y, \neg z),\ (\neg x, \neg b_1),\ (\neg y, \neg b_2)\}$ with the assumptions $\{b_1, b_2\}$. A possible core is $\kappa = \{\neg b_1, \neg b_2\}$. (Note that the cores returned by the SAT solver do not have to be minimal.)

The second iteration would then be a call to the SAT solver on the CNF $\{(x, \neg y),\ (y, z),\ (y, \neg z),\ (\neg x, \mathbf{r_1^0}, \neg b_1),\ (\neg y, \mathbf{r_2^0}, \neg b_2)\} \cup \text{CNF}\big(r_1^0 + r_2^0 = 1\big)$ with the same assumptions $\{b_1, b_2\}$. Again $\kappa = \{\neg b_1, \neg b_2\}$ is a possible core.

The third iteration would then be a call to the SAT solver on the CNF $\{(x, \neg y),\ (y, z),\ (y, \neg z),\ (\neg x, \mathbf{r_1^0}, \mathbf{r_1^1}, \neg b_1),\ (\neg y, \mathbf{r_2^0}, \mathbf{r_2^1}, \neg b_2)\} \cup \text{CNF}\big(r_1^0 + r_2^0 = 1\big) \cup \text{CNF}\big(r_1^1 + r_2^1 = 1\big)$ with the same assumptions $\{b_1, b_2\}$. This time the formula is satisfiable. For example, the truth assignment $\{x, y, z, r_1^0, r_2^1, \neg r_2^0, \neg r_1^1, b_1, b_2\}$ satisfies all of the clauses. At this point cost = 2.

Each core yields a subset of soft clauses of which at least one must be falsified. Adding the relaxation variables and the cardinality constraint gives the SAT

Algorithm 3: WPM1/WMSU1 algorithm for MaxSAT [ABL09, MMP09]

```
1  WPM1/WMSU1(F)
2  F^b = bv_transform(F); card_layer = {}; assumptions = {b_i | (b_i) ∈ soft(F^b)}
3  F' = hard(F^b); cost = 0; iter = 0; sat? = false
4  m = max({i | b_i ∈ assumptions}) + 1
5  while not sat? do
       /* SAT solver interface is the same as in Algorithm 2                    */
6      (sat?, π, κ) = SATSolve (F' ∪ card_layer, assumptions)
7      if not sat? then
8          wt_min = min_{b_i|¬b_i∈κ} wt((b_i))
9          cost = cost + wt_min
10         for ¬b_i ∈ κ do
11             if wt((b_i)) > wt_min then
                   /* Add new copy of clause c_i with new blocking variable b_m */
12                 c_m = c_i; m = m + 1; F' = F' ∪ {(c_m ∨ ¬b_m)}
13                 assumptions = assumptions ∪ {b_m}
14                 wt((b_m)) = wt((b_i)) − wt_min   /* New copy gets residual weight */
15             wt((b_i)) = wt_min                   /* Old copy gets weight wt_min */
16             c_i = c_i ∨ r_i^iter                 /* Add new relaxation variable to old copy */
                                                    /* Modifies clause (c_i ∨ ¬b_i) ∈ F' */
17         card_layer = card_layer ∪ CNF(Σ_{i|¬b_i∈κ} r_i^iter = 1)
18         iter = iter + 1
19     else
20         return (π|_vars(F), cost)
```

solver the freedom to falsify one (and only one) of these soft clauses by setting its relaxation variable to *true*. Each iteration finds a core subject to all of the previously derived relaxations. That is, at least one clause from the i'th core κ_i must be falsified even when we falsify a soft clause from each previously derived core. Hence, at stage k we have proved that the optimal cost is at least k. Note that the cores allow us to restrict the set of soft clauses that need to be considered for falsification. That is, only those soft clauses appearing in the found cores need be considered for falsification. This makes the size of the cardinality constraint encodings much smaller.

The example shows that one feature of this algorithm is that it can add multiple relaxation variables to the same clause if that clause appears in more than one core. This is a by-product of using an "$= 1$" cardinality constraint. In the example, we need to use one relaxation variable from each cardinality constraint in order to satisfy the two clauses $(\neg x, \neg b_1, \mathbf{r_1^0}, \mathbf{r_1^1})$ and $(\neg y, \neg b_2, \mathbf{r_2^0}, \mathbf{r_2^1})$; we cannot use two relaxation variables from the same cardinality constraint. These multiple relaxation variables in the soft clause can be a source of significant inefficiency as they introduce symmetries that can slow the SAT solver down [ABL13].

24.4.5.2. WPM1/WMSU1—Fu-Malik for Weighted MaxSAT

A restriction of the Fu-Malik algorithm is that it only works with unweighted MaxSAT instances. Soft clause cloning is a technique [ABL09, MMP09] that allows many unweighted algorithms to be extended to the general weighted case.

This technique was first used in the WPM1 [ABL09] and WMSU1 [MMP09] algorithms which appeared concurrently and independently in 2009. Both algorithms are essentially identical and extend Fu-Malik to deal with weighted soft clauses. Details are presented as Algorithm 3.

When a new core κ is found after we have already found and relaxed some previous cores, we know that at least one of the soft clauses in κ must be falsified. If these soft clauses have different weights, all that can be concluded is that the lower bound on the cost of an optimal solution must be incremented by $wt_{\min} = \min_{\{\neg b_i \in \kappa\}} wt((b_i))$, i.e., by the minimum weight among the soft clauses in κ. Clause cloning takes place at this stage, converting κ into a core containing clauses of weight $wt_{\min}$ by splitting every clause in κ into two copies: one with weight $wt_{\min}$ and one with the residual weight. The copies with the residual weight are added to the set of soft clauses, while the copies with weight $wt_{\min}$ are processed as an unweighted core.

In Algorithm 3 we again use the blocking variables introduced into F^b as assumptions so that we can extract cores. In processing each core we add a new relaxation variable to the hard clause corresponding to each of the soft clauses in the core (line 16). And for those soft clauses with weight greater than the minimum weight in the core ($wt_{\min}$), we create a new copy of the clause (line 12), give it a brand new blocking variable b_m, and add it to the formula F' (line 12). The old soft clause (b_i), whose hard clause $(c_i \vee r_i^{\text{iter}} \vee \neg b_i)$ contains the new relaxation variable, gets weight $wt_{\min}$. The new soft clause (b_m), whose hard clause $(c_m \vee \neg b_m)$ is a copy of the original clause with a new blocking variable, gets the remaining weight $wt((b_i)) - wt_{\min}$ (line 14). Note that, as in Algorithm 2, the unit soft clauses of F^b, including the new soft clause (b_m), only appear in the set of assumptions. So we also need to add b_m to the set of assumptions (line 13). This means that the SAT solver is not allowed to falsify the original soft clause c_i until this clause has appeared in cores whose sum of $wt_{\min}$ weights is equal to $wt(c_i)$. Only at that point will all copies of c_i have a relaxation variable allowing the SAT solver to falsify c_i. Furthermore, to falsify c_i the SAT solver will have to use a relaxation variable from each copy (by setting that variable to *true*). This will then "use up" all of the cardinality constraints c_i appeared in. That is, once c_i is *false*, all of the cardinality constraints it appears in will have one relaxation variable set, and will thus disallow any other relaxation variable in the constraint from being used.

Example 6. Consider

$$\begin{aligned} F &= \{(\neg x, \neg q), (\neg y, \neg q), (\neg z, \neg q), (x)_1, (y)_1, (z)_1, (q)_2\}, \\ F^b &= \{(\neg x, \neg q), (\neg y, \neg q), (\neg z, \neg q), (x, \neg b_1), (y, \neg b_2), (z, \neg b_3), (q, \neg b_4), \\ &\quad wt((b_1)) = wt((b_2)) = wt((b_3)) = 1, wt((b_4)) = 2\}. \end{aligned}$$

On this input, Algorithm 3 could perform the following sequence of iterations (different iterations are possible depending on what cores are returned by the SAT solver).

1. Core $\kappa_1 = \{\neg b_1, \neg b_4\}$, then $wt_{\min} = 1$, cost $= 0 + wt_{\min} = 1$, and

$$\begin{aligned} F' = \; & \{(x, \neg q), (y, \neg q), (z, \neg q), (x, \mathbf{r_1^0}, \neg b_1), (y, \neg b_2), (z, \neg b_3), \\ & \mathbf{r_4^0}, \neg b_4), (\mathbf{q}, \neg \mathbf{b_5})\} \cup \mathbf{CNF}\big(\mathbf{r_1^0} + \mathbf{r_4^0} = \mathbf{1}\big), \\ & \text{assumptions} = \{b_1, b_2, b_3, b_4, \mathbf{b_5}\}, \\ & wt((b_1)) = wt((b_2)) = wt((b_3)) = 1, \mathbf{wt}((\mathbf{b_4})) = \mathbf{1}, \mathbf{wt}((\mathbf{b_5})) = \mathbf{1}. \end{aligned}$$

2. Core $\kappa_2 = \{\neg b_1, \neg b_2, \neg b_4, \neg b_5\}$, then $wt_{\min} = 1$, cost $= 1 + wt_{\min} = 2$, and

$$\begin{aligned} F' = \; & \{(x, \neg q), (y, \neg q), (z, \neg q), (x, r_1^0, \mathbf{r_1^1}, \neg b_1), (y, \mathbf{r_2^1}, \neg b_2), (z, \neg b_3), \\ & (q, r_4^0, \mathbf{r_4^1}, \neg b_4), (\mathbf{q}, \mathbf{r_5^1}, \neg \mathbf{b_5})\} \\ & \cup \mathrm{CNF}\big(r_1^0 + r_4^0 = 1\big) \cup \mathbf{CNF}\big(\mathbf{r_1^1} + \mathbf{r_2^1} + \mathbf{r_4^1} + \mathbf{r_5^1} = \mathbf{1}\big), \\ & \text{assumptions and weights unchanged.} \end{aligned}$$

3. Now the formula is satisfied by the truth assignment $\pi = \{x,\ y,\ z,\ \neg q,\ r_4^0,\ r_5^1,\ \neg r_1^0,\ \neg r_1^1,\ \neg r_2^1,\ \neg r_4^1,\ b_1,\ b_2,\ b_3,\ b_4,\ b_5\}$, and $\pi|_{vars(F)} = \{x, y, z, \neg q\}$ is an optimal solution for F.

Fu-Malik and its weighted version are mostly of historical interest as the first application of cores and clause cloning, respectively. To date no one has developed implementations of these algorithms that perform as efficiently as more recent algorithms. It is of course always possible that new insight could make these algorithms competitive.

24.4.5.3. MSU3 for Unweighted MaxSAT

MSU3 [MP07] is a much simpler algorithm than Fu-Malik (Section 24.4.5.1), and, when implemented with incremental SAT solving [MJML14b], it has proved to be much more efficient. In fact, at the time this chapter was written, the implementation of MSU3 in the Open-WBO system [MML14] was one of the most effective solvers for the unweighted MaxSAT problem. MSU3, like the LSU algorithm (Section 24.4.4) and unlike the Fu-Malik algorithm, uses a simple encoding that transparently captures the decision problem $\mathcal{D}(F, k)$. Like Fu-Malik, MSU3 starts with $k = 0$ and increases k until $\mathcal{D}(F, k)$ becomes SAT exploiting the cores found for each UNSAT decision problem instance $\mathcal{D}(F, k)$. In particular, in the unweighted case, $\mathcal{D}(F, k)$ is asking the SAT solver to find a feasible solution that falsifies at most k soft clauses. The SAT solver need not consider all soft clauses for falsification. Rather, it needs to consider only those soft clauses that have appeared in one of the cores obtained from the previous SAT solver calls. This makes the cardinality constraint used in the encoding of $\mathcal{D}(F, k)$ much smaller and easier for the SAT solver to handle.

Algorithm 4 gives the details. It starts with an empty card_layer and assumptions that force all soft clauses to be satisfied (line 2). Hence, the SAT solver is being asked to find a feasible solution with zero cost. Unless a solution is found, the SAT solver will return a core specifying a set of soft clauses one of which must be falsified. The literals b_i forcing the satisfaction of these soft clauses (b_i) are then removed from the assumptions (line 9), and added to the cardinality constraint (line 8). Finally, the cost is incremented and a new cardinality constraint is constructed specifying that at most this new "cost" of soft clauses can be falsified.

Algorithm 4: MSU3 algorithm for unweighted MaxSAT [MP07]

```
1  MSU3(F)
2  F^b = bv_transform(F); card_layer = {}; assumptions = {b_i | (b_i) ∈ soft(F^b)}
3  inCard = {}
4  F' = hard(F^b); cost = 0; sat? = false
5  while not sat? do
       /* SAT solver interface is the same as in Algorithm 2                    */
6      (sat?, π, κ) = SATSolve (F' ∪ card_layer, assumptions)
7      if not sat? then
8          inCard = inCard ∪ κ
9          assumptions = assumptions \ {b_i | ¬b_i ∈ κ}
10         cost = cost + 1
11         card_layer = CNF( Σ_{¬b_i ∈ inCard} ¬b_i ≤ cost )
12     else
13         return (π|_{vars(F)}, cost)
```

Note that the initial cardinality constraint is only over the soft clauses that have appeared in the found core κ. In subsequent iterations, the core κ will specify a subset of soft clauses one of which must be falsified *even though the cardinality constraint already allowed some number of soft clauses to be falsified.* Hence, the bound on the cardinality constraint is always incremented by one, and it is always over the soft clauses in the union of the cores that have been found so far. Note also that by removing the literal b_i from the assumptions, we no longer require the SAT solver to satisfy the soft clause (b_i). However, by adding $\neg b_i$ to the cardinality constraint we still require the SAT solver to not falsify more than *cost* of these soft clauses. Note also that κ can be empty. In this case, the SAT solver has proved that the current bound on the number of soft clauses that can be falsified in the current cardinality constraint is too low. In this case, the only change to card_layer will be to increase the right-hand side bound "cost".

If we consider the formula being solved by the SAT solver on line 6, we see that only card_layer changes between SAT solver invocations. Furthermore, card_layer changes in two ways: (a) the right-hand side cost bound is incremented, and (b) potentially additional literals are added to the left-hand side sum. Martins et al. [MJML14b] show how, using a totalizer encoding of the cardinality constraint [BB03], both changes can be made by only adding new clauses to the SAT solver and by changing the assumed totalizer output variable to change the right-hand side bound (Section 24.4.2.1). Hence, the same SAT solver instance can be used throughout the algorithm giving rise to significant performance improvements over non-incremental implementations.

24.4.5.3.1. The PM2 Algorithm. Algorithm 4 (MSU3) uses a single cardinality constraint over all of the soft clauses that have appeared in some core. However, in some cases, this cardinality constraint can be decomposed into smaller cardinality constraints each over a disjoint set of soft clauses. This is the insight used in the PM2 algorithm [ABL09, ABL13]. In particular, PM2 operates just like MSU3 except that in each iteration card_layer will contain a set of cardinality constraints rather than just one larger cardinality constraint.

Consider the following sequence of cores (discovered by the SAT solver in this order): $\kappa_1 = \{\neg b_1, \neg b_2, \neg b_3\}$, $\kappa_2 = \{\neg b_4, \neg b_5, \neg b_6\}$, $\kappa_3 = \{\neg b_7, \neg b_8\}$, $\kappa_4 = \{\neg b_1, \neg b_6\}$. MSU3 would construct the following sequence of cardinality constraints (setting card_layer to the i'th constraint in the i'th iteration):

(1) $\neg b_1 + \neg b_2 + \neg b_3 \leq 1$,
(2) $\neg b_1 + \cdots + \neg b_6 \leq 2$,
(3) $\neg b_1 + \cdots + \neg b_8 \leq 3$,
(4) $\neg b_1 + \cdots + \neg b_8 \leq 4$.

However, the first three cores are over disjoint sets of soft clauses. PM2 would instead use a set of cardinality constraints in card_layer during each iteration:

(1) $\neg b_1 + \cdots + \neg b_3 \leq 1$,
(2) $\neg b_1 + \neg b_2 + \neg b_3 \leq 1$ **and** $\neg b_4 + \neg b_5 + \neg b_6 \leq 1$,
(3) $\neg b_1 + \neg b_2 + \neg b_3 \leq 1$ **and** $\neg b_4 + \neg b_5 + \neg b_6 \leq 1$ **and** $\neg b_7 + b_8 \leq 1$.

However, κ_4 is over soft clauses contained in κ_1 and κ_2. So as to ensure that its cardinality constraints are each over disjoint sets of soft clauses, PM2 would now merge the soft clauses in κ_1, κ_2 and κ_4 and construct one cardinality constraint over the union of their soft clauses with a bound of 3 (since 3 cores have been discovered over this subset of soft clauses). Hence, at iteration 4, PM2 would use the following set of cardinality constraints in card_layer: (4) $\neg b_1 + \cdots + \neg b_6 \leq 3$ **and** $\neg b_7 + \neg b_8 \leq 1$.

By splitting up the cores into multiple cardinality constraints PM2 is potentially giving the SAT solver an easier problem to solve. In particular, there are fewer settings of the b_i variables satisfying the two cardinality constraints $\neg b_1 + \cdots + \neg b_6 \leq 3$ **and** $\neg b_7 + \neg b_8 \leq 1$ than there are settings satisfying the single cardinality constraint $\neg b_1 + \cdots + \neg b_8 \leq 4$.

PM2 also added $\geq$ cardinality constraints (At-Least) to the SAT solver. However, these constraints are logically redundant—the SAT solver has already inferred that at least this number of soft clauses must be falsified. Potentially, these At-Least constraints might slow down the SAT solver.

24.4.5.4. OLL for Weighted MaxSAT

Another algorithm that is very efficient for MaxSAT is the OLL algorithm. OLL is originally an algorithm for optimization in Answer Set Programming [AKMS12] and was adapted for MaxSAT by Morgado et al. [MDM14]. The OLL MaxSAT algorithm was most recently (at the time this chapter was written) implemented in the RC2 solver.

The key idea of OLL is to utilize what are called "soft cardinality constraints." Consider an unweighted instance for which a core $\kappa = \{\neg b_1, \ldots \neg b_n\}$ has been found (i.e., for each $\neg b_i \in \kappa$, (b_i) is a soft clause with $wt((b_i)) = 1$). The formula can be relaxed by removing the b_i from the assumptions and adding a cardinality constraint $\sum_{\neg b_i \in \kappa} \neg b_i < j$. This allows the SAT solver to falsify up to $j-1$ of these soft clauses. Now define new summation output variables o_j for $j = 1, \ldots n-1$ such that o_j is *true* if and only if $\sum_{\neg b_i \in \kappa} \neg b_i \geq j$. That is, o_j is *true* if at least j of the soft clauses (b_1), ..., (b_n) are falsified; and *false* if at most $j-1$ are falsified.

Algorithm 5: OLL algorithm for MaxSAT [MDM14]

```
1  OLL(F)
2  F^b = bv_transform(F); card_layer = {}; assumptions = {b_i | (b_i) ∈ soft(F^b)}
3  F' = hard(F^b); cost = 0; iter = 0; sat? = false
4  while not sat? do
       /* SAT solver interface is the same as in Algorithm 2                    */
5      (sat?, π, κ) = SATSolve (F' ∪ card_layer, assumptions)
6      if not sat? then
7          wt_min = min_{ℓ|ℓ∈κ} wt((¬ℓ))
8          cost = cost + wt_min
9          for ℓ ∈ κ do                                  /* (¬ℓ) is a soft clause */
10             wt((¬ℓ)) = wt((¬ℓ)) − wt_min
11             if wt((¬ℓ)) == 0 then
12                 assumptions = assumptions \ {¬ℓ}
13         for o^i_j ∈ κ do                              /* Summation output variables */
14             if wt((¬o^i_j)) == 0 ∧ o^i_{j+1} exists then
                   /* i'th summation has another output variable                */
15                 assumptions = assumptions ∪ {¬o^i_{j+1}}
           /* Encode clauses that make output variables equal to value of sum   */
16         card_layer = card_layer ∪ CNF(Σ_{ℓ∈κ} ℓ ≥ j ≡ o^iter_j for o^iter_2, ..., o^iter_{|κ|})
17         for j = 2, ..., |κ| do
18             wt((¬o^iter_j)) = wt_min
19         assumptions = assumptions ∪ {¬o^iter_2}
20         iter = iter + 1
21     else
22         return (π|_{vars(F)}, cost)
```

We will refer to the o_j as being *output* variables of the cardinality constraint, and the literals being summed (the left-hand side) as the *input* variables.

Instead of adding the cardinality constraint $\sum_{\neg b_i \in \kappa} \neg b_i < j$, we can add the clauses specifying that for each output variable o_j the equivalence $o_j \equiv \sum_{\neg b_i \in \kappa} \neg b_j \geq j$ holds, and call the SAT solver under the assumption $\neg o_j$ (Section 24.4.2.1). This technique is used by [MJML14b] to make MSU3 incremental. However, the use of soft cardinality constraints takes this idea further. Instead of treating the literals $\neg o_j$ as hard assumptions, we can regard them to be new soft clauses $(\neg o_j)_1$. Since $o_j \rightarrow o_{j-1}$, we observe that any truth assignment falsifying the soft clause $(\neg o_j)_1$ must also falsify the j soft clauses $\{(\neg o_1)_1, \ldots, (\neg o_j)_1\}$. Hence any feasible solution falsifying $(\neg o_j)_1$ incurs cost j, and this is exactly the same as the costs incurred by falsifying j of the original soft clauses (b_i) (o_j is *true* implies j of the $\neg b_i$ are *true*). That is, replacing the soft clauses $(b_i)_1$ by the soft clauses $(\neg o_j)_1$ corresponding to the sums of the $\neg b_i$ is a cost-preserving transformation of the MaxSAT formula.

The main advantage of treating the $(\neg o_i)$ as new soft clauses is that cores can now be found over these variables. For example, the SAT solver could compute a core $\kappa = \{\neg b_2, \neg b_4, o_2^2\}$ where $(b_2)_1$ and $(b_4)_1$ are soft clauses of F^b, and o_2^2 is the 2nd output of the second cardinality constraint. This core asserts that either one of $(b_2)_1$ or $(b_4)_1$ must be falsified *or* more than two soft clause inputs of the second cardinality constraint must be falsified. This is clearly a more general

and potentially more powerful type of core than cores strictly over the original soft clauses (b_i). Furthermore, the outputs of one cardinality constraint (the right-hand side variables) can end up being used as the inputs (the left-hand side literals) to additional cardinality constraints, building up new output variables representing quite complicated constraints on the set of original soft clauses that can be falsified.

Algorithm 5 gives the details of the weighted case.[6] One main feature of this algorithm is that the totalizer encoding [BB03] of cardinality constraints already provides the summation output variables o_i used on line 16 (see Section 24.4.2.1).[7] Another feature of the algorithm is that through its use of assumptions it can perform clause cloning without having to actually duplicate the clauses as was needed in WPM1/WMSU1 (Section 24.4.5.2); this is described below.

The algorithm starts as normal with an empty card_layer and assumptions that force the satisfaction of all soft clauses. Whenever a core is found, the minimum weight over the soft clauses in the core is subtracted from the weight of every soft clause in the core. Note that with summation output literals $\neg o^i_j$ being assumed (lines 15 and 19) the core can contain either negated blocking variables $\neg b_i$ or positive output variables o^i_j. In both cases the corresponding soft clause contains the negation, (b_i) or $(\neg o^i_j)$. Line 10 reduces the weight of these soft clauses by $wt_{\min}$. If the weight drops to zero, then that particular soft clause has been fully absorbed into the card_layer (i.e., its weight has been fully accounted for in the cost of the cores found so far). In that case we can remove it from the assumptions; otherwise, we continue to assume it but now with reduced weight.

The summation output variables appearing in the core require special treatment if their weight drops to zero. If the weight of o^i_j drops to zero, the i-th cardinality constraint has had j of its inputs made *true*. This corresponds to j soft clauses being falsified, and the weight of these j falsifications has been fully absorbed into card_layer. Now we must disallow any further falsifications by assuming $\neg o^i_{j+1}$ (line 15). If further falsifications are needed, the SAT solver will subsequently generate cores containing o^i_{j+1}; eventually the weight of a $(j+1)$'th falsification, i.e., $wt((\neg o^i_{j+1}))$, will be absorbed into the card_layer and we will then assume o^i_{j+2}. Note that each of the soft clauses $(\neg o^i_j)$ has a weight equal to the minimum weight of the soft clauses in the i'th core (line 18). This is in general different from $wt_{\min}$ of the *iter*'th core currently being processed. If all of the summation output variables have been absorbed (i.e., there is no o^i_{j+1} output variable) then no further assumptions need be added to the SAT solver—the possible falsification of all of the soft clause inputs to this cardinality constraint has been fully accounted for by the algorithm.

After updating the weights of the soft clauses a new cardinality constraint is generated over the literals in the core (line 16) with a corresponding new set of summation output variables o^{iter}_j. We do not need o^{iter}_1 because we know that at

[6] The original publication [MDM14] did not detail the general weighted case, so this algorithm is based on the RC2 implementation.

[7] Algorithms that use the cardinality constraint $\text{CNF}\big(\sum_{\neg b_i \in \text{inCard}} \neg b_i \leq \mathit{cost}\big)$ with a specific bound, such as MSU3, often use the totalizer encoding that also supplies these summation output variables o_i. To impose the bound of $\leq \mathit{cost}$, these algorithms add $\neg o_{\mathit{cost}+1}$ to the set of SAT solver assumptions.

least one of the literals in κ must be *true*, κ specifies a conflict clause satisfied by all feasible solutions (see Section 24.4.1.2), and so o_1^{iter} is already *true*. These new summation output variables will count up how many literals $\neg\ell$ of κ are *true*, and thus how many of the corresponding soft clauses (ℓ) are *false*. Since each of the summation output variables gets weight $wt_{\min}$ (line 18), each input clause (ℓ) contributes $wt_{\min}$ if falsified. This is how clause cloning is implemented. If the soft clause (ℓ) (for $\neg\ell \in \kappa$) has weight $wt(\ell) > wt_{\min}$ we will have ℓ as an assumption with new weight $wt(\ell) - wt_{\min}$, and ℓ as an input to the cardinality constraint with weight $wt_{\min}$. These two uses of the literal ℓ correspond to the two clause copies used in WPM1, and there is no longer a need to add clause copies to the SAT solver.[8]

There are a number of options with respect to assuming the new output variables. In Algorithm 5 the assumptions contain at most one output literal from each cardinality constraint at any time: initially we add $\neg o_2^i$ (line 19), and whenever $wt((\neg o_j^i)$ drops to zero we remove it and add the next output literal $\neg o_{j+1}^i$. However, since $\neg o_j^i \rightarrow \neg o_{j+1}^i$ (equivalently $o_{j+1}^i \rightarrow o_j^i$) it is logically sound to assume all of the output literals $\neg o_j^i$ at once [ADR15]. In the RC2 implementation (see [BJM18]) only $wt((\neg o_2^{\text{iter}}))$ is initialized to $wt_{\min}$ (as in line 18). Then, whenever weight is removed from $(\neg o_j^i)$, an equivalent weight is added to $(\neg o_{j+1}^i)$. Thus by the time $(\neg o_2^{\text{iter}})$ has had its weight reduced to zero, $(\neg o_3^{\text{iter}})$ will have had a total of $wt_{\min}$ weight added to it. That is, instead of starting $(\neg o_3^{\text{iter}})$ with weight $wt_{\min}$ as is done on line 18, weight $wt_{\min}$ is moved to this soft clause incrementally. Hence, by the time $(\neg o_j^i)$, ..., $(\neg o_2^i)$ all have weight zero, $wt_{\min} \cdot j$ weight must have been accumulated into card_layer.

With multiple output literals o_i^j in the assumptions, the cores (which are not minimal) can contain multiple output literals from the same cardinality constraint. Furthermore, the weight of o_j^i might become zero even when o_{j-1}^i still has positive weight. Hence, solvers such as RC2 might remove o_j^i from the assumptions when its weight gets reduced to zero, and then add it back in when weight is later shifted in from o_{j-1}^i. The impact of this on the efficiency of the algorithm has not been reported to date.

Finally, it can be noted that the cardinality constraint built on line 16 of Algorithm 5 need not be built all at once. Instead, solvers like RC2 build this cardinality constraint incrementally, adding output variables o_j^{iter} only as these are needed.

24.4.5.5. Other Soft Cardinality Constraint Algorithms

Besides OLL other published algorithms have used the notion of dynamically creating new soft clauses representing summations or disjunctions of the original soft clauses. These include, e.g., PMRes [NB14], Maxino [ADR15], and WPM3 [ADG15], all of which have very good performance. We briefly describe their main features, leaving the interested reader to consult the references for more detailed information.

[8] This can yield a significant improvement in efficiency when the core contains hundreds or thousands of soft clauses.

All of these algorithms use the same main loop as Algorithm 5 except that they process cores differently. That is, initially all soft clauses are assumed to be *true* and card_layer is empty. Then, for every core κ that is found, $wt_{\min}$ is determined and subtracted from every soft clause in κ (line 10). If the remaining weight is greater than zero, a "copy" of the soft clause is left in the assumptions; otherwise, the clause is removed (line 12). This leaves κ with soft clauses all with the same weight $wt_{\min}$.

After this, the core is processed, updating card_layer and assumptions. In Algorithm 5 this is done on lines 13 to 19. It is this processing and these lines that are changed by the different algorithms we discuss here.

First, it is useful to describe the properties processing a core must satisfy. Core processing has to update the formula so that the SAT solver is now allowed to falsify *one* of the weight $wt_{\min}$ soft clauses in κ: $wt_{\min}$ has been added to cost (line 8) so the cost of a single falsification has already been accounted for. Furthermore, the update must preserve the condition that if a truth assignment falsifies j soft clauses of κ, then that truth assignment must incur a cost of $(j-1)\cdot wt_{\min}$ in the new formula.

In OLL this is accomplished by defining the summation output variables o_j^{iter} for $j = 2$ to $|\kappa|$ with totalizer clauses (added to card_layer) ensuring that $o_j^{\text{iter}} \equiv \sum_{\ell\in\kappa} \ell \geq j$. Then the soft clauses $(\neg o_j^{\text{iter}})_{wt_{\min}}$ are added by adding $\neg o_j^{\text{iter}}$ to the assumptions and setting $wt((\neg o_j^{\text{iter}})) = wt_{\min}$ (line 18). These soft clauses are added incrementally to the assumptions (line 15), with initially only $\neg o_2^{\text{iterm}}$ in the assumptions (line 19). Thus, since $\neg o_1^{\text{iter}}$ is never assumed nor even defined, the SAT solver is allowed to falsify any single soft clause in κ. Furthermore, if a truth assignment falsifies j soft clauses in κ, the variables o_2^{iter}, ..., o_j^{iter} will all be made *true*, and thus $j-1$ soft clauses $(\neg o_2^{\text{iter}})_{wt_{\min}}$, ..., $(\neg o_j^{\text{iter}})_{wt_{\min}}$ will be falsified incurring cost $(j-1)\cdot wt_{\min}$.

PMRes. The PMRes algorithm processes cores differently. In particular, PMRes does not use summation output variables. Instead it operates as follows. Let $\kappa = \{\neg\ell_1, \ldots, \neg\ell_m\}$ and $w = wt_{\min}$. Thus the soft clauses are $(\ell_1)_w$, ..., $(\ell_m)_w$. PMRes will define $m-1$ variables d_2, ..., d_m such that $d_i \equiv \ell_i \vee (\ell_1 \wedge \cdots \wedge \ell_{i-1})$, adding the clauses defining this equivalence to card_layer ([NB14] presents a compact way of encoding these equivalences). That is, each d_i is *true* if either the soft clause $(\ell_i)_w$ is satisfied or if all prior soft clauses in the sequence $(\ell_1)_w$, ..., $(\ell_{i-1})_w$ are satisfied. It will then add the soft clauses $(d_2)_w$, ..., $(d_m)_w$ by adding d_2, ..., d_m to the assumptions and setting $wt((d_1)) = w$, ..., $wt((d_m)) = w$.

In the new formula, the SAT solver is now allowed to falsify any single soft clause in κ. If exactly one $(\ell_i)_w$ is falsified, then all of the d_j assumptions will be *true*: for $j \neq i$, ℓ_j is *true*, making d_j *true*, and for $j = i$, the conjunction $(\ell_1 \wedge \cdots \wedge \ell_{i-1})$ is *true*, making d_j *true*. Furthermore, if a truth assignment falsifies j of the soft clauses $(\ell_i)_w$, then it will make d_i true for all i corresponding to satisfied soft clauses; it will make d_i *true* for i equal to the index of the first falsified soft since all previous soft clauses are satisfied; and it will make d_i *false* for i equal to the index of all $j-1$ subsequent falsified soft clauses, since in this case all previous soft clauses are not satisfied. Hence, the truth assignment will falsify exactly $j-1$ of the soft clauses $(d_i)_w$ in the new formula incurring cost $(j-1)\cdot wt_{\min}$.

PMRes has proved to be a very effective algorithm, with the Eva solver implementing it was one of the best performers in the 2014 MaxSAT evaluation. An open issue with PMRes is that the order of the soft clauses in κ has an influence on the algorithm's performance. Why this is the case is not well understood. A better understanding of this issue might lead to improved ordering schemes and better performance.

Maxino. The Maxino algorithm uses summation output variables to define the new soft clauses like OLL. Its main innovation is to use a simple technique to partition the core κ into a sequence of smaller subsets that are pairwise disjoint and whose union is κ. The algorithm then constructs a sequence of cardinality constraints, each cardinality constraint summing the falsified soft clauses of one partition. It links the cardinality constraints together by including the 1 output (the output variable indicating that the sum of the inputs is ≥ 1) from the previous cardinality constraint as an input of the subsequent cardinality constraint.

Each cardinality constraint allows at most one of its inputs to be *true* (at most one of its corresponding soft clauses to be *false*). In particular, its outputs indicating that that the sum is ≥ 2, ≥ 3, ..., are all assumed to be *false* and each is given weight $wt_{\min}$ (i.e., the negation of these output variables become new soft clauses as in OLL[9]). Its 1 output indicating that the sum is ≥ 1 is not made into a soft clause, instead the 1 output is fed into the subsequent cardinality constraint.

With these new assumptions and clauses capturing the new sequence of cardinality constraints, the SAT solver is now allowed to falsify any single soft clause in κ. Say that the single falsified soft clause is an input to the i'th cardinality constraint in the sequence of cardinality constraints. The first to $i(-1)$'th cardinality constraints will all have only *false* inputs and thus will set all of their outputs to *false*, satisfying the assumptions. The i'th cardinality constraint has one *true* input, the input corresponding to the falsified soft clause, and will set its 1 output to *true* and all other outputs to *false*. This again satisfies all of the assumptions since the 1-output has not been added to the assumptions. The $(i+1)$'th cardinality constraint will have a *true* input, the input corresponding to the 1 output of the i'th cardinality constraint, and will set its 1 output to *true* and all other outputs to *false*. Again this will satisfy all of the assumptions since the 1 output has not been added to the assumptions. This pattern will continue for the subsequent cardinality constraints and all assumptions will be satisfied.

Furthermore, if j soft clauses in κ are falsified by some truth assignment, then the sequence of cardinality constraints will have j *true* input variables. The first cardinality constraint in the sequence with $i > 0$ *true* inputs, will set its i, $i-1$, ..., 1 outputs to *true* incurring cost $(i-1) \cdot wt_{\min}$: the 1 output is not a soft clause and hence does not incur any cost while the other outputs all falsify soft clauses with weight $wt_{\min}$. Every subsequent cardinality constraint with k *true* input variables will also get an additional *true* input variable from the previous cardinality constraint's 1 output. Thus, it will set its $k+1$, k, ..., 1 outputs to *true*

[9]However, all output variables are assumed to be *false*, whereas in OLL only the ≥ 2 output is assumed to be *false*. As already explained, OLL incrementally assumes the output variables of the cardinality constraint whereas Maxino assumes them all at once.

incurring cost $k \cdot wt_{\min}$: again the 1 output does not incur any cost. Summing over all the cardinality constraints it can be seen that a total cost of $(j-1) \cdot wt_{\min}$ will be incurred.

The interested reader may consult [ADR15] for further details. However, we note that in [ADR15] the totalizer encoding is presented in a non-standard way. In particular, for a totalizer with inputs $\ell_1, \ldots, \ell_n$ and outputs $o_i \equiv \sum_{j=1}^{n} \ell_j \geq i$ $(1 \leq i \leq n)$, [ADR15] uses the non-standard but equivalent encoding $\text{CNF}(\neg\ell_1 + \cdots + \neg\ell_n + o_1 + \cdots o_n \geq n)$ conjoined with $\bigwedge_{i=1}^{n} o_i \rightarrow o_i$. That is, if k inputs ℓ_i are *true* (i.e. at most $k-n$ of the $\neg\ell_i$ are *true*), then at least k outputs must be *true* with the *true* outputs clustered at the lowest indices.

Maxino is another very effective algorithm being one of the best performing algorithms in the 2018 MaxSAT evaluation. An open issue with Maxino is understanding more precisely how partitioning the core into smaller subsets helps the solver, and how this clustering should be done in order to maximize solver performance.

WPM3. The WPM3 algorithm, like Maxino and OLL, uses summation output variables to define the new soft clauses, but it differs in some ways. First, the cardinality constraints it uses are always summations of falsified input soft clauses. The new "meta" soft clauses arising from the output of previous cardinality constraints are not used as inputs to new cardinality constraints as in Maxino, OLL, and PMRes. Second, WPM3 utilizes a sub-optimization step to find the best bound for each core.

When processing a new core κ, WPM, like OLL, distinguishes between the soft clauses that are input soft clauses and those that are output variables of previously constructed cardinality constraints. In particular, if $o_i^j \in \kappa$ is the i'th output of the j'th cardinality constraint ($(\neg o_i^j)$ is a soft clause), then WPM3 replaces o_i^j with the union of the inputs of the j'th cardinality constraint. Since this replacement is done before the new cardinality constraint over the literals in the core is constructed, every cardinality constraint will have only input soft clauses as input (i.e., literals indicating the falsification of some input soft clause). After building this larger core κ, WPM3 constructs a new cardinality constraint C with the literals in κ as inputs and a corresponding set of output variables o_i^C counting the number of falsified soft clauses in κ.

Now instead of assuming the 2 output of the new cardinality constraint C to limit the number of falsified soft clauses to one, WPM3 does a sub-optimization to determine if this limit must be higher. This can be done in various ways, but WPM3 applies the LSU (Algorithm 1) over the subproblem consisting of $hard(F_b)$ along with only the soft clauses that are inputs to C. Say that this sub-optimization concludes that k of the soft clause inputs of C must be falsified. Then WPM3 will add the output $\neg o_{k+1}^C$ to the set of assumptions with weight $wt_{\min}$ and will add $(k-1) \cdot wt_{\min}$ to the cost. In other words, after concluding that k soft clauses inputs to C must be falsified, WPM3 will construct a problem in which the SAT solver can falsify no more than k of these soft clauses without incurring an additional cost.

24.4.5.5.1. Remarks. The algorithms mentioned in this section, all of which use the technique of adding new soft clauses defined in terms of previous soft clauses, seem to have similar levels of excellent performance. It is not clear if any single approach is generally the best. In particular, much of the demonstrated performance differences seem to arise from the quality of implementation and additional techniques used in the implementation. We close this section by mentioning some of the key techniques used to improve the base algorithm.

An important technique for all core-guided algorithms is weight stratification [ABL09]. This is the technique of partitioning the soft clauses into k sets so that the soft clauses in set i have higher weight than those in set $i-1$. Then the problem is solved ignoring all soft clauses except those in the k'th set. Once a solution is found for these higher-weight soft clauses, the $k-1$'th set of soft clauses is added and the problem solved again. This is repeated until the problem has been solved with all soft clauses.

The reason this technique is useful for core-guided algorithms is that it generates cores with higher minimum weight. This moves the cost towards the optimum more rapidly and reduces the amount of clause cloning that needs to occur. In particular, even with clause cloning optimized to be simply reusing a literal (as described earlier), excessive cloning will still slow the solver down by forcing it to generate more cores. For example, if an instance contains a single soft clause of weight 100 that must be falsified in any optimal solution, along with many other soft clauses of weight 1, then the solver would have to generate at least 100 cores if each of these cores has $wt_{\min}$ of 1. One hundred such cores would be required to reduce the weight 100 soft clause down to weight zero: before its weight is reduced to zero it remains in the assumptions and must continue to be satisfied by the SAT solver. Furthermore, every core generated augments the complexity of card_layer, which starts to slow the SAT solver down. Even with stratification, however, core-guided algorithms become significantly less efficient as the number of distinct soft clause weights increases.

The WPM3 technique of finding an optimal lower bound for each cardinality constraint via a sub-optimization also seems to be quite effective. For example, RC2 implementation of OLL (see [BJM18]) employed this technique (its "exhaust_core" method). As pointed out earlier, each added cardinality constraint slows down the SAT solver. The sub-optimization technique allows the solver to increase the cost towards the optimum without adding additional cardinality constraints; optimization is done by using the output variables of the already added cardinality constraint as SAT solver assumptions (recall Section 24.4.5.5).

The final technique that we will mention here is that of core minimization. There are a number of algorithms for reducing the size of a core. For example, one simple technique is as follows. Given a core $\kappa = \{\ell_1, \ldots, \ell_n\}$, we can ask the SAT solver if it can satisfy $\neg\ell_1, \ldots, \neg\ell_{n-1}$ using assumptions. In the negative case, the SAT solver it will provide a new core that must be a strict subset of κ. In the positive case, the test can be performed leaving out a different literal ℓ_i, $i \neq n$. Depending on the size of κ and the complexity of the current SAT formula (which is growing as we process more cores and add more cardinality constraints), this technique can be effective in reducing the size of the core after which the reduced core can be processed. Core reduction if applied judiciously

can speed up the solver.

Besides the algorithms already mentioned, it should also be noted that a number of other core-guided MaxSAT algorithms have been proposed, e.g., [IMM+14, MHM12, MM08, ABL10a]. These algorithms have mostly been subsumed by the algorithms discussed, but some ideas and insights might still be available by examining these earlier works more carefully.

24.4.6. The Implicit Hitting Set Approach

Implicit hitting set (IHS) MaxSAT solvers [DB11, DB13a, DB13b, Dav13, SBJ16] instantiate the implicit hitting set paradigm [DCB10, Kar10] in the context of MaxSAT. In particular, similarly to the purely SAT-based core-guided approach, IHS solvers extract cores in an iterative fashion. However, in contrast to the core-guided approach, in the IHS approach hitting sets over the iteratively accumulated set of cores are computed. If a minimum-cost hitting set (MCHS)[10] of the accumulated cores, when removed from the input formula, makes the formula satisfiable, then any satisfying truth assignment must be an optimal solution [DB11]. In contrast to solvers implementing the purely SAT-based core-guided approach, IHS solvers do not transform the input instance using core compilation steps; rather, the input MaxSAT instance 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 (selected through the use of assumptions). 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 (consisting of no more than a few hundred clauses on current typical MaxSAT benchmarks).

To obtain a MCHS HS, current IHS solvers for MaxSAT have employed integer programming (IP) solvers on the following MCHS IP formulation:

$$\begin{aligned} \text{minimize} \quad & \sum_{(b_i) \in soft(F^b)} wt((b_i)) \cdot \neg b_i & \\ \text{subject to} \quad & \sum_{\neg b_i \in \kappa} \neg b_i \geq 1 & \forall \kappa \in \mathcal{K}, \\ & \neg b_i \in \{0, 1\} & \forall (b_i) \in soft(F^b), \end{aligned}$$

where $\mathcal{K}$ is the accumulated set of cores, the b_i are the blocking variables for the input soft clauses c_i created by the blocking variable transformation (Section 24.4.1.1.1), and $\neg b_i = 1$ when b_i is *false* and c_i is falsified.[11] In the original

[10] Given a set of cores, a hitting set of the cores is a set of soft clauses that includes a soft clause from each core. A hitting set is optimal (of minimum cost) iff the sum of the weights of the soft clauses in it is smallest among all hitting sets of the set of cores.

[11] Other techniques for computing minimum-cost hitting sets besides using IP solvers can be used. These include branch-and-bound methods [Kle11] and SAT based methods, e.g., [IPLM15]. In certain contexts these methods can be quite effective, e.g., when recomputing a minimum-cost hitting set after adding a single new set. However, these alternatives to IP solvers either cannot handle the weighted case or are not very efficient on the weighted case. In addition, IP solvers support adding other types of constraints that can be extracted from the input formula [DB13a], and bounding techniques that can be used to harden some of the soft clauses [BHJS17]; both of these ideas can yield significant efficiency improvements.

Algorithm 6: IHS algorithm for MaxSAT [DB13b]

```
1  IHS(F)
2  F^b = bv_transform(F); F' = hard(F^b); K = {}; HS = {}
   /* Initialize upper and lower bounds (UB and LB). LB is initialized to zero. UB is
      initialized to a feasible solution. A feasible solution must exist since by assumption
      hard(F) is satisfiable. Remember the solution yielding the UB cost. SAT solver
      interface is the same as in Algorithm 2                                          */
3  (sat?, π, κ) = SATSolve (F', {})
4  UB = cost(π); LB = 0; best_sol = π
5  while true do
6      (sat?, π, κ) = SATSolve (F', {b_i | (b_i) ∈ (soft(F^b) \ HS)})
7      if not sat? then
8          K = K ∪ {κ}
9          (HS, MCHS?) = ComputeHittingSet(K)
10         if MCHS? then
11             LB = max(LB, cost(HS))
12             if LB ≥ UB then
13                 return (best_sol, cost(best_sol))
14     else
15         if cost(π) < UB then
16             UB = cost(π)
17             best_sol = π
18             if LB ≥ UB then
19                 return (best_sol, cost(best_sol))
```

IHS algorithm for MaxSAT [DB11] each iteration of the algorithm involved computing a MCHS HS of the current set of cores $\mathcal{K}$ (which is initially empty) and then testing the satisfiability of $hard(F^b)$ subject to the assumptions $\{b_i \mid (b_i) \in \big(soft(F^b) \setminus HS\big)\}$. That is, a feasible solution satisfying all soft clauses except those in the MCHS HS must be found. If this formula is satisfiable, the set HS must hit *all* cores of the instance [Rei87], and the optimality of HS implies the optimality of the solution [DB11], terminating the MaxSAT search.

One benefit of the IHS approach is the possibility of tightly integrating upper and lower bound information into search. In fact, starting with [DB13b], more recent versions of IHS MaxSAT solvers have used a different approach that is better cast as an algorithm computing an upper and lower bound and stopping when these two bounds meet. This more modern version of the IHS algorithm is shown in Algorithm 6.

The algorithm starts by applying the blocking variable transformation and initializing the set of cores $\mathcal{K}$ and the hitting set HS to be empty. It also initializes the lower bound (LB) to be zero and finds a feasible solution whose cost is used to initialize the upper bound (UB). Note that by calling the SAT solver with an empty set of assumptions (line 3) the SAT solver is free to falsify any of the soft clauses. Hence, this call will return SAT if any feasible solution exists, and since by assumption the input set of hard clauses, $hard(F)$ are satisfiable, this call must return SAT (Section 24.4.1.1.1). The feasible solution is stored in best_sol so that an optimal solution can be returned on termination.

Then the main loop is entered where the SAT solver is asked to find a solution satisfying all soft clauses except those in the current hitting set (line 6). If the

solver reports unsatisfiability, the found core κ is added to the set of cores $\mathcal{K}$, and a new hitting set is computed. If that hitting set is an MCHS, the lower bound can be updated to be the maximum of its previous value and the cost of the MCHS HS (line 11). In particular, every truth assignment π must falsify at least one soft clause in every core. Thus the set of soft clauses falsified by π forms a hitting set of $\mathcal{K}$. The cost of π is equal to the sum of the weights of these falsified clauses; that is, it is equal to the cost of the hitting set it generates. Thus $cost(\pi)$ must be at least as high as the cost of any MCHS. Since this is true for all truth assignments including the optimal solutions when HS is an MCHS, its cost must be a valid lower bound on $cost(F)$.

On the other hand, if the SAT solver returns a satisfying truth assignment, we can check the cost of the returned solution π and update the upper bound if we have found a new lower cost solution. In both cases when either the lower or upper bound is updated we check if the bounds now meet. If they do, we return best_sol, since has now been proven to be an optimal solution.

ComputeHittingSet. The one part of Algorithm 6 that is incompletely specified is how ComputeHittingSet works. This subroutine must satisfy two properties: (a) it must return a hitting set of $\mathcal{K}$, and (b) it must always eventually return an MCHS of $\mathcal{K}$. That is, ComputeHittingSet is free to compute and return non-optimal hitting sets (in which case MCHS? is returned as *false*), but there must always exist a future call that will compute and return an MCHS. In fact, the ability to postpone computing MCHS (which is a computationally hard problem) has been shown to be essential for the effectiveness of the IHS approach [DB13b].

It can be shown that as long as ComputeHittingSet satisfies these two properties, Algorithm 6 will eventually terminate returning an optimal solution [BHJS17]. Nevertheless, there is still considerable flexibility in how ComputeHittingSet works, and although effective implementations have been found, deep insight has not yet been achieved into how to implement this function optimally. However, Saikko [Sai15] did provide useful empirical evidence about different techniques.

In current IHS solvers (MaxHS and LMHS [BJM18]) the following techniques are used to compute hitting sets.

1. The entire new core κ is added to the previously computed hitting set HS. Clearly, if HS was a hitting set of all previous cores in $\mathcal{K}$, $HS \cup \kappa$ will be a hitting set of $\mathcal{K} \cup \{\kappa\}$. Typically, this technique is used immediately after any other more expensive technique for computing a hitting set is employed, and it is used until the hitting set becomes so large that the SAT solver returns SAT. Saikko [Sai15] showed that employed in this way this technique is empirically effective.
2. A greedy algorithm can be used to compute a new hitting set. The standard greedy heuristic, where the soft clause that hits the most cores for the least weight is continually selected until all cores are hit, is typically used.
3. The IP solver can be used to compute a hitting set but it can be stopped as soon as it finds a feasible solution with a cost lower than the current upper bound. This technique can help limit the time consumed by the IP

solver.

4. Finally the IP solver is used when it is time to compute an MCHS

These techniques are listed in order of increasing computational cost. ComputeHittingSet implementations typically try to repeat the least expensive technique as many times as they can before moving on to a more expensive technique, with the aim of reducing the time required to accumulate a set of cores capable of raising the lower bound. In particular, when the hitting set returned by a technique immediately yields a solution, there is no longer any point in repeating that technique—no more cores can be found using it until some other cores have been found by other techniques. This "repeat the cheapest until nothing new can be computed" method has proved to be quite effective in practice, but other more sophisticated policies might well be even more effective.

Other Techniques. A number of additional techniques are employed by IHS solvers to speed up the algorithm. Here we mention some of the most important ones.

Seeding is the technique of examining the input MaxSAT instance to determine if any constraints can immediately be added to the IP solver. This technique is quite effective and is more fully described in [DB13a].

Core minimization (Section 24.4.5.5.1) is very effective for IHS solvers [DB13a] and is standardly employed. However, as with its use in core-guided algorithms, the resources expended on core minimization must be limited. For example, it would be too expensive to compute the smallest core. Nevertheless, core minimization tends to be easier to perform in IHS solvers than in core-guided solvers. As already mentioned, the cores in IHS solvers tend to be smaller to begin with, and since cardinality constraints are not being added to the formula the SAT solver calls used during minimization are more efficient. In IHS solvers, minimizing the cores results in tighter constraints for the IP solver, allowing it to raise the lower bound with fewer cores.

The final technique that we will mention here is the IP technique of *reduced cost fixing*. IP solvers standardly work by iteratively solving linear (LP) relaxations of the input IP and then refining those relaxations with cuts or by branching. In the context of IHS, the LP relaxation of the minimum-cost hitting set IP provides a lower bound on the value of the MCHS, which in turn is a lower bound on the cost of an optimal solution to the MaxSAT problem. Hence, as proposed in [BHJS17], the LP can be used to obtain a *reduced cost* for every soft clause by computing the reduced costs of the IP variables $\neg b_i$. This allows the solver to make use of the standard IP technique of reduced cost fixing [Wol98, DFJ54, CJP83, NW99]. Intuitively, the reduced cost of a soft clause specifies a minimum increase in the cost of the LP relaxation that would arise from falsifying (respectively, satisfying) a soft clause which was satisfied (respectively, falsified) in a computed optimal LP solution. If this cost increase makes the lower bound provided by the LP relaxation greater than the current best known MaxSAT upper bound, the clause (i.e., its blocking variable) can be fixed to its status as prescribed by the LP solution. Fixing the values of blocking variables subsequently simplifies both the minimum-cost hitting set computations as well as the SAT solver calls. This use of a proven IP technique to improve IHS solvers

is made possible by their hybrid use of both SAT and IP solvers. Potentially, other ideas from the rich field of IP solving could be discovered to be useful.

24.4.7. Solving MaxSAT Directly via Integer Programming

In addition to developing SAT-based approaches to MaxSAT, integer programming systems can be used to directly solve MaxSAT instances using the standard encoding of MaxSAT to integer programming [GW93]. In more detail, starting with the blocking variable transformation F^b, an IP 0/1 variable x can be defined for each propositional variable x with $\neg x$ translated as $(1 - x)$. Then every clause can be translated into a ≥ 1 linear constraint, and the objective function can be defined to be the minimization of $\sum_{(b_i) \in soft(F^b)} wt((b_i)) \cdot \neg b_i$ As shown in [AG13, DB13a], IP solvers can be quite effective on some MaxSAT instances using this standard IP encoding of MaxSAT. However, in general, their performance is not competitive with MaxSAT-specific algorithms on most instances.

24.5. Further Developments

Finally, we overview further recent developments in MaxSAT solving involving preprocessing, practical incomplete algorithms, algorithm portfolios, partitioning-based, and parallel approaches to MaxSAT solving.

24.5.1. Preprocessing

Preprocessing techniques for SAT provide significant performance improvements to SAT solving, and preprocessing is today a standard part of the SAT solving workflow (see also Chapter 9). In comparison, progress on MaxSAT preprocessing is more recent and preprocessing less routinely applied.

Nevertheless, essentially all major SAT preprocessing techniques can be naturally lifted to MaxSAT. The key to SAT-based preprocessing for MaxSAT is in introducing first blocking variables through the F^b transformation (cf. Section 24.4.1.1.1), turning each soft clause c into the hard clause $c \vee \neg b_C$. The clauses of $hard(F^b)$ resulting from this transformation can then be preprocessed using an SAT preprocessing with one minor but—in terms of correctness for MaxSAT–very important restriction: the elimination of the blocking variables by resolving on the variables (e.g., through bounded variable elimination) must be disallowed, in order to preserve the unsatisfiable core structure of the instance [BJM13, BMM13]. By lifting the notions of resolution asymmetric tautologies [JHB12] from SAT to MaxSAT, a generic proof of correctness for natural MaxSAT liftings of SAT preprocessing techniques was recently provided [BJ19].

An optimal solution to the original MaxSAT instance can then be obtained from any optimal solution to the instance after applying SAT preprocessing by applying the generic solution reconstruction technique for SAT preprocessing [JHB12] in linear time with respect to the number of preprocessing steps made.

After applying SAT preprocessing the clauses can contain more than one

blocking variable, and a blocking variable can occur in multiple clauses.[12] Hence the bijective mapping between blocking variables and soft clauses is lost after SAT preprocessing. For coupling SAT preprocessing with core-guided and IHS MaxSAT approaches, this is not problematic, but nevertheless requires attention. The blocking variables introduced before preprocessing can be directly used as blocking variables by the MaxSAT algorithms which employ blocking variables, instead of performing another F^b transformation step after preprocessing [BSJ15a].

Variables that can be directly used as blocking variables can also be detected from input MaxSAT instances and reused as assumptions throughout the MaxSAT solving process, thereby limiting the number of new variables added by the F^b transformation step, as proposed in [BSJ15b] under the notion of *group detection*. Group detection refers to detecting auxiliary variables introduced by MaxSAT encodings of more high-level finite-domain constraints. In more detail, assume that a set of clauses $\text{CNF}(\mathcal{C}) = \{C_1, \ldots, C_k\}$ is a conjunctive normal form encoding of a finite-domain constraint $\mathcal{C}$. Now assume that $\mathcal{C}$ is a soft constraint, with an associated weight $W_{\mathcal{C}}$. On the level of *group MaxSAT* [HMM15] this soft constraint can be represented as the soft group $\{C_1, \ldots, C_k\}$ with weight $W_{\mathcal{C}}$. For employing a standard MaxSAT solver, a natural way of encoding such a group-level MaxSAT representation is to introduce a single blocking variable $b_{\mathcal{C}}$ and add it to each of the clauses $C_1, \ldots, C_k$, thereby obtaining the hard clauses $C_1 \vee \neg b_{\mathcal{C}}, \ldots, C_k \vee \neg b_{\mathcal{C}}$ together with the soft clause $b_{\mathcal{C}}$ with weight $W_{\mathcal{C}}$. With this intuition, group detection refers to pattern matching in a MaxSAT instance literals that are directly re-usable as blocking variables. In particular, a literal l is re-usable as a blocking variable if the literal $\neg l$ only appears in the hard clauses of the MaxSAT instance. This captures the auxiliary variables arising from the already-described general approach of encoding high-level soft constraints in MaxSAT. In practice, in [BSJ15b] the authors showed that in the industrial and crafted benchmark instances of MaxSAT Evaluation 2014, variables re-usable directly as blocking variables were found in a majority of the instances using group detection, most often in significant numbers.

The impact of SAT preprocessing on the efficiency of MaxSAT algorithms in practice has been to date less studied, although some theoretical evidence on its potential is available; see e.g. [BMM13, KBSJ17]. However, it has been shown that, theoretically, SAT preprocessing has no effect on the best-case number of iterations of core-guided MaxSAT algorithms [BJ16], which suggests that its impact may mainly be in making the individual SAT solver calls within core-guided MaxSAT algorithms simpler.

Beyond SAT preprocessing, native MaxSAT-specific preprocessing techniques have also been proposed. A limited form of variable saturation in the context of MaxSAT resolution was proposed as a MaxSAT preprocessing technique in [ALM08]. More recently, the MaxPre MaxSAT preprocessor [KBSJ17] introduced several further preprocessing techniques native to MaxSAT, including label matching, binary core removal, and generalized form of subsumed label elimina-

[12] In this context, blocking variables are sometimes referred to as *labels*, following the *labeled CNF* framework [BM12] that explicitly distinguishes between the blocking variables and original clauses of each clause.

tion [BSJ16], the last of which allows for removing blocking variables (labels) in the following sense: A blocking variable b is subsumed by a set of blocking variables $\{b_1, \ldots, b_k\}$ if one of $b_1, \ldots, b_k$ occurs in a clause whenever b occurs in the clause, and $\sum_{i=1}^{k} wt(b_i) \leq wt(b)$. For $k = 1$, this gives the original proposed definition of subsumed label elimination [BSJ16], while in general the conditions correspond to the NP-complete hitting set problem. In practice, an approximate version of the generalized form of subsumed label elimination can be computed using a classical $\ln(n)$-approximation algorithm for the hitting set problem.

In terms of implementations of MaxSAT preprocessing, the SAT preprocessor Coprocessor [Man12] supports SAT preprocessing techniques for MaxSAT instances, while the MaxPre preprocessor [KBSJ17] offers combinations of SAT preprocessing and native MaxSAT preprocessing techniques.

24.5.2. Incomplete Algorithms for MaxSAT

Incomplete MaxSAT solving can primarily be divided into two categories: (i) local search MaxSAT solvers and (ii) approximation algorithms based on iterative calls to SAT solvers that may not guarantee optimality.

24.5.2.1. Local Search for MaxSAT

Local search solvers start by finding a random assignment to the MaxSAT formula. Since this assignment is unlikely to satisfy all clauses, they choose a clause that is unsatisfied and flip the assignment of a variable occurring in that clause. When compared to local search solvers for SAT, local search solvers for MaxSAT face additional challenges since they must find an assignment that satisfies all hard clauses while trying to maximize the sum of the weights of satisfied soft clauses. To overcome this challenge, local MaxSAT solvers distinguish between soft and hard clauses and apply weighting schemes that favor satisfying hard clauses [CLTS14, LCW+15, CJS15, CLLS16, LCSH17, LC18]. For more details on local search SAT solvers, we refer the interested reader to Chapter 6.

24.5.2.2. Model-Improving Approximation

Model-improving approximation algorithms do not guarantee the optimality of the final solution. An example of this approach is an enumeration of minimal correction subsets (MCSes) [MHJ+13, MPM15]. An MCS of an unsatisfiable set of clauses is a minimal subset that, if removed, makes the formula satisfiable. There has been a recent trend of using approximation approaches for incomplete MaxSAT. For instance, an approach based on bit-vector optimization was proposed for unweighted incomplete MaxSAT with promising results [Nad18].

LinSBPS [DS18] is an incomplete MaxSAT solver based on model-improving algorithms. For weighted instances, they simplify the weights of the formula and see it in *low resolution*, i.e., with all their weights divided by a large value. Since they use integer division, the weights of some soft clauses are set to zero, effectively removing them. Whenever the solver terminates with an "optimal" solution to the simplified formula, the resolution is increased by decreasing the divisor value. Additionally, they use solution-based saving [AG17, DCS18] to guide the search towards the best solution found so far.

Open-WBO-Inc [JKMR18] is another incomplete MaxSAT solver based on model-improving algorithms. For weighted instances, this solver also simplifies the weights of the formula by clustering them into k different weights. An alternative approach also employed by this solver is to consider a weighted instance as a multilevel optimization problem (even though it may not satisfy the BMO condition described in Section 24.3.1.1). This alternative approach reduces the solving of a weighted instance to solving a sequence of unweighted instances, which significantly improves the performance of model-improving algorithms and quickly converges to a final solution which corresponds to an upper bound on the optimal value of the original formula.

24.5.3. Algorithm Portfolios

Given the diversity of MaxSAT algorithms (some of which were overviewed in Section 24.4), a meta approach that aims at taking advantage of this diversity is to build an oracle to predict the most suitable MaxSAT algorithm for a particular instance. Inspired by the success of SATzilla [XHHL08] and other algorithm portfolios for SAT solving, Matos et al. [MPLM08] proposed the first algorithm portfolio for MaxSAT by employing linear regression using three kinds of features [NLH+04]: problem size features, balance features and local search probe features.

ISAC+ [AMS14] is another algorithm portfolio approach that extends the instance-specific algorithm configuration (ISAC [KMST10]) by clustering the benchmarks and tuning the parameters of existing solvers on each cluster. ISAC does not use regression-based analysis but instead computes a representative feature vector that captures properties of a particular instance in order to identify clusters of similar instances. A single solver is selected for each group based on the characteristics of those benchmarks. Given a new instance, its features are computed and it is assigned to the closest cluster and solved by the respective solver. The features used by ISAC+ consider the percentage of clauses that are soft, and the statistics of the distribution of weights (average, minimum, maximum, standard deviation). The remaining 32 features are a subset of the features used by SATZilla that are specific to SAT instances and are applied to MaxSAT instances by considering all soft clauses as hard. ISAC+ differs from ISAC since it also applies an algorithm selector to choose the best parameters of existing solvers for each cluster. Using a portfolio of different MaxSAT solvers, allowed ISAC+ to be one of the best approaches for MaxSAT in the 2013-2015 MaxSAT Evaluations.

24.5.4. Parallel Solving

Nowadays, extra computational power is no longer coming from higher processor frequencies but rather from adding additional processors. Distributed systems are also becoming more predominant and cheaper. For instance, it is not uncommon to deploy distributed tools to web services such as Amazon Web Services. Even though parallel approaches for MaxSAT are not as predominant as in SAT solving, there are a few parallel MaxSAT solvers that can take advantage of these new architectures. These parallel solvers can be mainly categorized into two classes: (i) portfolio approaches, and (ii) search space splitting.

24.5.4.1. Portfolio Approaches

Portfolio approaches are not restricted to sequential solving and can be extended to a parallel setting where different strategies are run in parallel. These different strategies can be based on different MaxSAT algorithms (e.g., model-improving algorithms, core-guided algorithms, implicit hitting set algorithms) or by using different settings of a given algorithm (e.g., changing the encoding for cardinality or pseudo-Boolean constraints). An example of a portfolio solver is PWBO [MML11] which allocates half of the workers to use model-improving algorithms, and the other half core-guided algorithms. To further increase diversity between workers using the same algorithm, PWBO translates cardinality and pseudo-Boolean constraints with different encodings [MML11].

The portfolio approaches for parallel MaxSAT solving are closely related to portfolio approaches for parallel SAT solving. The main difference consists on how diversity is employed. Both approaches perform clause sharing where learned clauses between different workers are exchanged to further prune the search space and boost the performance of the parallel solver.

Portfolio approaches are better suited for multicore approaches since diversity does not usually scale and clause sharing is more efficient when performed in shared memory.

24.5.4.2. Search Space Splitting

Another approach for parallel MaxSAT solving is to split the original problem into subproblems such that each worker solves a smaller formula that is hopefully easier to solve. The two main approaches to divide the search space is to (i) perform interval splitting or (ii) split the search space with guiding paths.

The optimal solution value of a MaxSAT problem is bounded between 0 and the sum of the weights of the soft clauses. Therefore one way of splitting the search space is to divide this interval and have different workers searching on different bounds in order to narrow the search interval and converge to an optimal solution. This approach is employed in a version of PWBO [MML12].

The search space can also be viewed as a binary tree, where each node corresponds to a variable and each of its edges corresponds to an assignment to that variable. The search space can then be split by using different paths in the tree (called guiding paths) to split the search into sub-trees and assign each of them to a different worker. There are many ways of generating guiding paths. One of the most successful approaches has been using lookahead solvers [vdTHB12] to split the search tree. This strategy has been used in a distributed MaxSAT solver [TLM16].

One of the advantages of this approach is that it is easier to split the search space into many workers and therefore can be more suitable for a distributed setting. However, since some of the sub-trees may be much easier to solve than others, it is necessary to have a dynamic load balancing scheme that continuously splits the search space and keeps all workers busy.

24.5.5. Partitioning-based MaxSAT

Divide-and-conquer techniques are not restricted to parallel MaxSAT solving and can also be applied to sequential MaxSAT solving. For instance, partitioning-based MaxSAT has been successfully used to improve the performance of core-based algorithms [MML13, NMJ+15].

Core-based algorithms rely on unsatisfiable cores returned by the SAT solver. Usually, core-based algorithms perform better when small cores are found early. However, this is not necessarily the case, since these unsatisfiable cores are not guaranteed to be minimal, and there are no guarantees that smaller cores will be found first. The order in which cores are enumerated can have a large impact on the performance of MaxSAT algorithms.

Similarly to SAT instances, MaxSAT instances can also be represented using graphs. For instance, a MaxSAT instance can be represented by either the variable incidence graph (VIG) or the clause variable incidence graph (CVIG) [MML13]. In the VIG representation, vertices correspond to the variables of the problem, and edges correspond to clauses where both variables occur. In the CVIG representation, vertices correspond to both variables and clauses and there is an edge between the vertices that represent the variables and the clauses where they occur. One can also use the resolution rule to construct a graph where vertices are clauses and edges correspond to clauses that can be resolved with each other (RES) [NMJ+15].

Partitioning-based MaxSAT uses graph partition techniques to split the formula into disjoint subsets (e.g., using a modularity clustering approach [GN02]). By decomposing the MaxSAT instance into disjoint subformulas, the SAT solver is forced to find unsatisfiable cores on a subset of the formula. If such unsatisfiable cores exist, then they may be much smaller than if the entire MaxSAT instance was given to the SAT solver. The RES representation [NMJ+15] is implemented on top of the Open-WBO system [MML14] using the MSU3 algorithm and has been shown to be particularly effective for unweighted MaxSAT.

24.6. Summary

Advances in practical aspects of maximum satisfiability within the last 10-15 years have established MaxSAT as a general-purpose Boolean constraint optimization paradigm. The improvements have come in increasing numbers both in terms of domain-specific MaxSAT encodings of various hard optimization problems to novel algorithmic techniques and their efficient implementations in general-purpose MaxSAT solvers. In terms of algorithmic advances, the advent of SAT-based approaches to practical MaxSAT solving in its different forms is arguably at the center of this success. To this end, we have here aimed at covering central research advances on practical aspects of MaxSAT within the last 10 years, with a strong focus on the three main SAT-based approaches to MaxSAT solving: the model-improving, core-guided, and the implicit hitting set approach. Beyond further improvements to these algorithmic approaches to MaxSAT, there are plenty of avenues for further research on MaxSAT in terms of novel applications of the

technology, as well as further advances in the recent research directions on incomplete and parallel approaches to MaxSAT solving.

Acknowledgments

The authors thank Jeremias Berg, Alexey Ignatiev, Joao Marques-Silva, and Antonio Morgado, who provided various helpful comments on a draft version of this chapter.

References

[ABJM17] C. Ansótegui, F. Bacchus, M. Järvisalo, and R. Martins, editors. *MaxSAT Evaluation 2017: Solver and Benchmark Descriptions*, volume B-2017-2 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2017.

[ABL09] C. Ansótegui, M. L. Bonet, and J. Levy. Solving (weighted) partial MaxSAT through satisfiability testing. 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 427–440. Springer, 2009.

[ABL10a] C. Ansótegui, M. L. Bonet, and J. Levy. A new algorithm for weighted partial MaxSAT. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[ABL+10b] J. Argelich, D. L. Berre, I. Lynce, J. Marques-Silva, and P. Rapicault. Solving linux upgradeability problems using boolean optimization. In I. Lynce and R. Treinen, editors, *Proceedings First International Workshop on Logics for Component Configuration, LoCoCo 2010, Edinburgh, UK, 10th July 2010.*, volume 29 of *EPTCS*, pages 11–22, 2010.

[ABL13] C. Ansótegui, M. L. Bonet, and J. Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[ACLM12] J. Argelich, A. Cabiscol, I. Lynce, and F. Manyà. Efficient encodings from CSP into SAT, and from MaxCSP into MaxSAT. *Multiple-Valued Logic and Soft Computing*, 19(1-3):3–23, 2012.

[ADG15] C. Ansótegui, F. Didier, and J. Gabàs. Exploiting the structure of unsatisfiable cores in MaxSAT. In Q. Yang and M. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 283–289. AAAI Press, 2015.

[ADR15] M. Alviano, C. Dodaro, and F. Ricca. A MaxSAT algorithm using cardinality constraints of bounded size. In Q. Yang and M. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*

2015, Buenos Aires, Argentina, July 25-31, 2015, pages 2677–2683. AAAI Press, 2015.

[AG13] C. Ansótegui and J. Gabàs. Solving (weighted) partial MaxSAT with ILP. In C. P. Gomes and M. Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings*, volume 7874 of *Lecture Notes in Computer Science*, pages 403–409. Springer, 2013.

[AG17] C. Ansótegui and J. Gabàs. WPM3: an (in)complete algorithm for weighted partial MaxSAT. *Artificial Intelligence*, 250:37–57, 2017.

[AIMT13] C. Ansótegui, I. Izquierdo, F. Manyà, and J. Torres-Jiménez. A Max-SAT-based approach to constructing optimal covering arrays. In K. Gibert, V. J. Botti, and R. R. Bolaño, editors, *Artificial Intelligence Research and Development - Proceedings of the 16th International Conference of the Catalan Association for Artificial Intelligence, Vic, Catalonia, Spain, October 23-25, 2013.*, volume 256 of *Frontiers in Artificial Intelligence and Applications*, pages 51–59. IOS Press, 2013.

[AKMS12] B. Andres, B. Kaufmann, O. Matheis, and T. Schaub. Unsatisfiability-based optimization in clasp. In A. Dovier and V. S. Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[ALM08] J. Argelich, C. M. Li, and F. Manyà. A preprocessor for Max-SAT solvers. In H. K. Büning and X. Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 15–20. Springer, 2008.

[ALM09] J. Argelich, I. Lynce, and J. Marques-Silva. On solving boolean multilevel optimization problems. In C. Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 393–398. AAAI Press, 2009.

[ALMP08] J. Argelich, C. M. Li, F. Manyà, and J. Planes. The first and second Max-SAT evaluations. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):251–278, 2008.

[ALMP11] J. Argelich, C. M. Li, F. Manyà, and J. Planes. Analyzing the instances of the MaxSAT evaluation. In K. A. Sakallah and L. Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 360–

361. Springer, 2011.

[AMS14] C. Ansótegui, Y. Malitsky, and M. Sellmann. MaxSAT by improved instance-specific algorithm configuration. In C. E. Brodley and P. Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2594–2600. AAAI Press, 2014.

[AN14] R. J. A. Achá and R. Nieuwenhuis. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research*, 218(1):71–91, 2014.

[ANOR09] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks and their applications. 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 167–180. Springer, 2009.

[ARMS02] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Generic ILP versus specialized 0-1 ILP: an update. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 450–457, 2002.

[BACF17] A. Belabed, E. Aïmeur, M. A. Chikh, and H. Fethallah. A privacy-preserving approach for composite web service selection. *Transactions on Data Privacy*, 10(2):83–115, 2017.

[BB03] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In F. Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003.

[BBR09] O. Bailleux, Y. Boufkhad, and O. Roussel. New encodings of pseudo-boolean constraints into CNF. 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 181–194. Springer, 2009.

[BDS11] P. Bieber, R. Delmas, and C. Seguin. DALculus – theory and tool for development assurance level allocation. In F. Flammini, S. Bologna, and V. Vittorini, editors, *Computer Safety, Reliability, and Security - 30th International Conference, SAFECOMP 2011, Naples, Italy, September 19-22, 2011. Proceedings*, volume 6894 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2011.

[BGSV15] M. Bofill, M. Garcia, J. Suy, and M. Villaret. MaxSAT-based scheduling of B2B meetings. In L. Michel, editor, *Integration of AI and OR Techniques in Constraint Programming - 12th*

International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings, volume 9075 of *Lecture Notes in Computer Science*, pages 65–73. Springer, 2015.

[BHJS17] F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko. Reduced cost fixing in MaxSAT. In J. C. Beck, editor, *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2017.

[BJ13] J. Berg and M. Järvisalo. Optimal correlation clustering via MaxSAT. In W. Ding, T. Washio, H. Xiong, G. Karypis, B. M. Thuraisingham, D. J. Cook, and X. Wu, editors, *13th IEEE International Conference on Data Mining Workshops, ICDM Workshops, TX, USA, December 7-10, 2013*, pages 750–757. IEEE Computer Society, 2013.

[BJ14] J. Berg and M. Järvisalo. SAT-based approaches to treewidth computation: An evaluation. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, pages 328–335. IEEE Computer Society, 2014.

[BJ16] J. Berg and M. Järvisalo. Impact of SAT-based preprocessing on core-guided MaxSAT solving. In M. Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 66–85. Springer, 2016.

[BJ17] J. Berg and M. Järvisalo. Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. *Artificial Intelligence*, 244:110–142, 2017.

[BJ19] J. Berg and M. Järvisalo. Unifying reasoning and core-guided search for maximum satisfiability. In F. Calimeri, N. Leone, and M. Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 287–303. Springer, 2019.

[BJB+14] K. Bunte, M. Järvisalo, J. Berg, P. Myllymäki, J. Peltonen, and S. Kaski. Optimal neighborhood preserving visualization by maximum satisfiability. In C. E. Brodley and P. Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1694–1700. AAAI Press, 2014.

[BJM13] A. Belov, M. Järvisalo, and J. Marques-Silva. Formula preprocessing in MUS extraction. In N. Piterman and S. A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24,*

2013. Proceedings, volume 7795 of *Lecture Notes in Computer Science*, pages 108–123. Springer, 2013.

[BJM14] J. Berg, M. Järvisalo, and B. Malone. Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In J. Corander and S. Kaski, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 86–95. JMLR.org, 2014.

[BJM18] F. Bacchus, M. Järvisalo, and R. Martins, editors. *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, volume B-2018-2 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2018.

[BJM19] F. Bacchus, M. Järvisalo, and R. Martins. MaxSAT Evaluation 2018: New developments and detailed results. *Journal on Satisfiability, Boolean Modeling and Computation*, 11:99–131, 2019.

[BLM07] M. L. Bonet, J. Levy, and F. Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8-9):606–618, 2007.

[BM12] A. Belov and J. Marques-Silva. Generalizing redundancy in propositional logic: Foundations and hitting sets duality. *CoRR*, abs/1207.1257, 2012.

[BMM13] A. Belov, A. Morgado, and J. Marques-Silva. SAT-based preprocessing for MaxSAT. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2013.

[BP10] D. L. Berre and A. Parrain. The Sat4j library, release 2.2. *Journal of Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–6, 2010.

[BSJ15a] J. Berg, P. Saikko, and M. Järvisalo. Improving the effectiveness of SAT-based preprocessing for MaxSAT. In Q. Yang and M. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 239–245. AAAI Press, 2015.

[BSJ15b] J. Berg, P. Saikko, and M. Järvisalo. Re-using auxiliary variables for MaxSAT preprocessing. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, pages 813–820. IEEE Computer Society, 2015.

[BSJ16] J. Berg, P. Saikko, and M. Järvisalo. Subsumed label elimination for maximum satisfiability. In G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, editors, *ECAI 2016 - 22nd European Con-*

ference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 630–638. IOS Press, 2016.

[CHB17] E. Cohen, G. Huang, and J. C. Beck. (I can get) satisfaction: Preference-based scheduling for concert-goers at multi-venue music festivals. In S. Gaspers and T. Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2017.

[CJP83] H. Crowder, E. L. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[CJS15] S. Cai, Z. Jie, and K. Su. An effective variable selection heuristic in SLS for weighted Max-2-SAT. *Journal of Heuristics*, 21(3):433–456, 2015.

[CLLS16] S. Cai, C. Luo, J. Lin, and K. Su. New local search methods for partial MaxSAT. *Artificial Intelligence*, 240:1–18, 2016.

[CLTS14] S. Cai, C. Luo, J. Thornton, and K. Su. Tailoring local search for partial MaxSAT. In C. E. Brodley and P. Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2623–2629. AAAI Press, 2014.

[CSMV10] Y. Chen, S. Safarpour, J. Marques-Silva, and A. G. Veneris. Automated design debugging with maximum satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(11):1804–1817, 2010.

[CSVMS09] Y. Chen, S. Safarpour, A. G. Veneris, and J. Marques-Silva. Spatial and temporal design debug using partial MaxSAT. In F. Lombardi, S. Bhanja, Y. Massoud, and R. I. Bahar, editors, *Proceedings of the 19th ACM Great Lakes Symposium on VLSI 2009, Boston Area, MA, USA, May 10-12 2009*, pages 345–350. ACM, 2009.

[Dav13] J. Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction.* PhD thesis, University of Toronto, 2013.

[DB11] J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In J. H. 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.

[DB13a] J. Davies and F. Bacchus. Exploiting the power of MIP solvers in MaxSAT. In M. Järvisalo and A. V. 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.

[DB13b] 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.

[DCB10] J. Davies, J. Cho, and F. Bacchus. Using learnt clauses in MaxSAT. In D. Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, volume 6308 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2010.

[DCS18] E. Demirovic, G. Chu, and P. J. Stuckey. Solution-based phase saving for CP: A value-selection heuristic to simulate local search behavior in complete solvers. In J. N. Hooker, editor, *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*, pages 99–108. Springer, 2018.

[Dec03] R. Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

[DFJ54] G. B. Danzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2:393–410, 1954.

[DGT18] R. Dimitrova, M. Ghasemi, and U. Topcu. Maximum realizability for linear temporal logic specifications. In S. K. Lahiri and C. Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 458–475. Springer, 2018.

[DM17] E. Demirovic and N. Musliu. MaxSAT-based large neighborhood search for high school timetabling. *Computers & Operations Research*, 78:172–180, 2017.

[DMW17] E. Demirović, N. Musliu, and F. Winter. Modeling and solving staff scheduling with partial weighted MaxSAT. *Annals of Operations Research*, 2017.

[dOS15] R. T. de Oliveira and F. Silva. On a relative MaxSAT encoding for the steiner tree problem in graphs. In O. Pichardo-Lagunas, O. Herrera-Alcántara, and G. Arroyo-Figueroa, editors, *Advances in Artificial Intelligence and Its Applications - 14th Mexican International Conference on Artificial Intelligence, MICAI 2015, Cuernavaca, Morelos, Mexico, October 25-31, 2015. Proceedings, Part II*, volume 9414 of *Lecture Notes in Computer Science*, pages 422–434. Springer, 2015.

[DS18] E. Demirovic and P. J. Stuckey. LinSBPS. In *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, volume B-2018-2 of *Department of Computer Science Series of Publications B*, page 8. University of Helsinki, 2018.

[ER18] R. Ehlers and F. P. Romero. Approximately propagation complete and conflict propagating constraint encodings. In O. Beyersdorff and C. M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2018.

[ES03] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.

[ES06] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[FBM+17] Y. Feng, O. Bastani, R. Martins, I. Dillig, and S. Anand. Automated synthesis of semantic malware signatures using maximum satisfiability. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[FG10] A. M. Frisch and P. A. Giannoros. SAT encodings of the at-most-k constraint. In *Procedings of 9th International Workshop on Constraint Modelling and Reformulation*, 2010.

[FLQ+14] Z. Fang, C. Li, K. Qiao, X. Feng, and K. Xu. Solving maximum weight clique using maximum satisfiability reasoning. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 303–308. IOS Press, 2014.

[FM06] 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.

[GL12] J. Guerra and I. Lynce. Reasoning over biological networks using maximum satisfiability. In M. Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 941–956. Springer, 2012.

[GML11] A. Graça, J. Marques-Silva, and I. Lynce. Haplotype inference using propositional satisfiability. In R. Bruni, editor, *Mathematical Approaches to Polymer Sequence Analysis and Related Problems*, pages 127–147. Springer, 2011.

[GMLO11] A. Graça, J. Marques-Silva, I. Lynce, and A. L. Oliveira. Haplotype inference with pseudo-boolean optimization. *Annals of Operations Research*, 184(1):137–162, 2011.

[GN02] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[GW93] M. X. Goemans and D. P. Williamson. A new $\frac{3}{4}$-approximation algorithm for MAXSAT. In G. Rinaldi and L. A. Wolsey, editors, *Proceedings of the 3rd Integer Programming and Combinatorial Optimization Conference, Erice, Italy, April 29 - May 1, 1993*, pages 313–321. CIACO, 1993.

[HKD+16] W. Huang, D. A. Kitchaev, S. T. Dacek, Z. Rong, A. Urban, S. Cao, C. Luo, and G. Ceder. Finding and proving the exact ground state of a generalized ising model by convex optimization and MAX-SAT. *Physical Review B*, 94:134424, 2016.

[HLdGS08] F. Heras, J. Larrosa, S. de Givry, and T. Schiex. 2006 and 2007 Max-SAT Evaluations: Contributed instances. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):239–250, 2008.

[HLH19] M. Hague, A. W. Lin, and C.-D. Hong. CSS minification via constraint solving. In *Transactions on Programming Languages and Systems*. ACM, 2019.

[HLO08] F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSAT: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[HMM15] F. Heras, A. Morgado, and J. Marques-Silva. MaxSAT-based encodings for group MaxSAT. *AI Communications*, 28(2):195–214, 2015.

[HMS12] S. Hölldobler, N. Manthey, and P. Steinke. A compact encoding of pseudo-boolean constraints into SAT. In B. Glimm and A. Krüger, editors, *KI 2012: Advances in Artificial Intelligence - 35th Annual German Conference on AI, Saarbrücken, Germany, September 24-27, 2012. Proceedings*, volume 7526 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2012.

[HPJ+17] A. Hyttinen, S. M. Plis, M. Järvisalo, F. Eberhardt, and D. Danks. A constraint optimization approach to causal discovery from subsampled time series data. *International Journal of Approximate Reasoning*, 90:208–225, 2017.

[HSJ17] A. Hyttinen, P. Saikko, and M. Järvisalo. A core-guided approach to learning optimal causal graphs. In C. Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Aus-*

tralia, August 19-25, 2017, pages 645–651. ijcai.org, 2017.

[HST17] M. Heizmann, C. Schilling, and D. Tischner. Minimization of visibly pushdown automata using partial Max-SAT. In A. Legay and T. Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10206 of *Lecture Notes in Computer Science*, pages 461–478. Springer, 2017.

[IJM14] A. Ignatiev, M. Janota, and J. Marques-Silva. Towards efficient optimization in package management systems. In P. Jalote, L. C. Briand, and A. van der Hoek, editors, *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 745–755. ACM, 2014.

[IMM+14] A. Ignatiev, A. Morgado, V. M. Manquinho, I. Lynce, and J. Marques-Silva. Progression in maximum satisfiability. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 453–458. IOS Press, 2014.

[IMM17] A. Ignatiev, A. Morgado, and J. Marques-Silva. On Tackling the Limits of Resolution in SAT Solving. In S. Gaspers and T. Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 164–183. Springer, 2017.

[IPLM15] A. Ignatiev, A. Previti, M. H. Liffiton, and J. Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In G. Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, pages 173–182, 2015.

[IPNM18] A. Ignatiev, F. Pereira, N. Narodytska, and J. Marques-Silva. A SAT-based approach to learn explainable decision sets. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 627–645. Springer, 2018.

[JHB12] M. Järvisalo, M. Heule, and A. Biere. Inprocessing rules. In B. Gramlich, D. Miller, and U. Sattler, editors, *Auto-*

mated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012.

[JKMR18] S. Joshi, P. Kumar, R. Martins, and S. Rao. Approximation strategies for incomplete MaxSAT. In J. N. Hooker, editor, *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*, pages 219–228. Springer, 2018.

[JM11] M. Jose and R. Majumdar. Cause clue clauses: error localization using maximum satisfiability. In M. W. Hall and D. A. Padua, editors, *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 437–446. ACM, 2011.

[JMM15] S. Joshi, R. Martins, and V. M. Manquinho. Generalized totalizer encoding for pseudo-boolean constraints. In G. Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2015.

[Kar10] R. M. Karp. Implicit hitting set problems and multi-genome alignment. In A. Amir and L. Parida, editors, *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, volume 6129 of *Lecture Notes in Computer Science*, page 151. Springer, 2010.

[KBSJ17] T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo. MaxPre: An extended MaxSAT preprocessor. In S. Gaspers and T. Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017.

[Kle11] J. D. Kleer. Hitting set algorithms for model-based diagnosis. In *Proceedings of the 22nd International Workshop on Principles of Diagnosis (DX 2011)*, pages 100–105, 2011.

[KMST10] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. ISAC - instance-specific algorithm configuration. In H. Coelho, R. Studer, and M. J. Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 751–756. IOS Press, 2010.

[KZFH12] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal of Satisfiabil-*

ity, Boolean Modeling and Computation, 8(1/2):95–100, 2012.

[LC18] Z. Lei and S. Cai. Solving (weighted) partial MaxSAT by dynamic local search for SAT. In J. Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 1346–1352. ijcai.org, 2018.

[LCSH17] C. Luo, S. Cai, K. Su, and W. Huang. CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243:26–44, 2017.

[LCW+15] C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su. CCLS: an efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7):1830–1843, 2015.

[LH05] J. Larrosa and F. Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In L. P. Kaelbling and A. Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 193–198. Professional Book Center, 2005.

[LJX15] C. Li, H. Jiang, and R. Xu. Incremental MaxSAT reasoning to reduce branches in a branch-and-bound algorithm for MaxClique. In C. Dhaenens, L. Jourdan, and M. Marmion, editors, *Learning and Intelligent Optimization - 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*, volume 8994 of *Lecture Notes in Computer Science*, pages 268–274. Springer, 2015.

[LK12] P.-C. K. Lin and S. P. Khatri. Application of Max-SAT-based ATPG to optimal cancer therapy design. *BMC Genomics*, 13, 2012.

[LM11] I. Lynce and J. Marques-Silva. Restoring CSP satisfiability with MaxSAT. *Fundamenta Informatica*, 107(2-3):249–266, 2011.

[LMMP10] C. M. Li, F. Manyà, N. O. Mohamedou, and J. Planes. Resolution-based lower bounds in MaxSAT. *Constraints*, 15(4):456–484, 2010.

[LQ10] C. M. Li and Z. Quan. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[LZK16] X. Liao, H. Zhang, and M. Koshimura. Reconstructing AES key schedule images with SAT and MaxSAT. *IEICE Transactions*, 99-D(1):141–150, 2016.

[LZMS12] C. M. Li, Z. Zhu, F. Manyà, and L. Simon. Optimizing with minimum satisfiability. *Artificial Intelligence*, 190:32–44, 2012.

[MAGL11] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce. Boolean lexicographic optimization: algorithms & applications. *An-*

nals of Mathematics and Artificial Intelligence, 62(3-4):317–343, 2011.

[Man12] N. Manthey. Coprocessor 2.0 - A flexible CNF simplifier - (tool presentation). In A. Cimatti and R. Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 436–441. Springer, 2012.

[Mar17] R. Martins. Solving RNA alignment with MaxSAT. In *MaxSAT Evaluation 2017: Solver and Benchmark Descriptions*, volume B-2017-2 of *Department of Computer Science Series of Publications B*, pages 29–30. University of Helsinki, 2017.

[MBM16] C. J. Muise, J. C. Beck, and S. A. McIlraith. Optimal partial-order plan relaxation via MaxSAT. *Journal of Artificial Intelligence Research*, 57:113–149, 2016.

[MDM14] 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.

[MHJ+13] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. On computing minimal correction subsets. In F. Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 615–622. IJCAI/AAAI, 2013.

[MHM12] A. Morgado, F. Heras, and J. Marques-Silva. Improvements to core-guided binary search for MaxSAT. In A. Cimatti and R. Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 284–297. Springer, 2012.

[MIM17] J. Marques-Silva, A. Ignatiev, and A. Morgado. Horn maximum satisfiability: Reductions, algorithms and applications. In E. C. Oliveira, J. Gama, Z. A. Vale, and H. L. Cardoso, editors, *Progress in Artificial Intelligence - 18th EPIA Conference on Artificial Intelligence, EPIA 2017, Porto, Portugal, September 5-8, 2017, Proceedings*, volume 10423 of *Lecture Notes in Computer Science*, pages 681–694. Springer, 2017.

[MJIM15] J. Marques-Silva, M. Janota, A. Ignatiev, and A. Morgado. Efficient model based diagnosis with maximum satisfiability. In Q. Yang and M. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1966–1972. AAAI Press, 2015.

[MJML14a] R. Martins, S. Joshi, V. Manquinho, and I. Lynce. On using in-

cremental encodings in unsatisfiability-based MaxSAT solving. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:59–81, 2014.

[MJML14b] R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce. Incremental cardinality constraints for MaxSAT. 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 531–548. Springer, 2014.

[MKTR16] T. Marcel Kevin and S. Tilak Raj. Finding pre-production vehicle configurations using a MaxSAT framework. In *Proceedings of the 18th International Configuration Workshop*, pages 117–122. École des Mines d'Albi-Carmaux, 2016.

[MLM13] A. Morgado, M. H. Liffiton, and J. Marques-Silva. MaxSAT-based MCS enumeration. In A. Biere, A. Nahir, and T. E. J. Vos, editors, *Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2013.

[MM08] J. Marques-Silva and V. M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In H. K. Büning and X. Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 225–230. Springer, 2008.

[MM10] A. Morgado and J. Marques-Silva. Combinatorial optimization solutions for the maximum quartet consistency problem. *Fundamenta Informatica*, 102(3-4):363–389, 2010.

[MM11] A. Morgado and J. Marques-Silva. On validating boolean optimizers. In *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011*, pages 924–926, 2011.

[MM18] D. Maliotov and K. S. Meel. MLIC: A MaxSAT-based framework for learning interpretable classification rules. In J. N. Hooker, editor, *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2018.

[MML11] 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.

[MML12] R. Martins, V. M. Manquinho, and I. Lynce. Parallel search

for maximum satisfiability. *AI Communications*, 25(2):75–95, 2012.

[MML13] R. Martins, V. M. Manquinho, and I. Lynce. Community-based partitioning for MaxSAT solving. In M. Järvisalo and A. V. 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 182–191. Springer, 2013.

[MML14] R. Martins, V. M. Manquinho, and I. Lynce. Open-WBO: A modular MaxSAT solver. In C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014.

[MMP09] 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.

[MNRS17] F. Manyà, S. Negrete, C. Roig, and J. R. Soler. A MaxSAT-based approach to the team composition problem in a classroom. In G. Sukthankar and J. A. Rodríguez-Aguilar, editors, *Autonomous Agents and Multiagent Systems - AAMAS 2017 Workshops, Visionary Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers*, volume 10643 of *Lecture Notes in Computer Science*, pages 164–173. Springer, 2017.

[MP07] J. Marques-Silva and J. Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, abs/0712.1097, 2007.

[MPLM08] P. J. Matos, J. Planes, F. Letombe, and J. Marques-Silva. A MAX-SAT algorithm portfolio. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. M. Avouris, editors, *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 911–912. IOS Press, 2008.

[MPM15] C. Mencía, A. Previti, and J. Marques-Silva. Literal-based MCS extraction. In Q. Yang and M. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1973–1979. AAAI Press, 2015.

[MPS14] N. Manthey, T. Philipp, and P. Steinke. A more compact translation of pseudo-boolean constraints into CNF such that generalized arc consistency is maintained. In C. Lutz and M. Thielscher, editors, *KI 2014: Advances in Artificial In-*

telligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings, volume 8736 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2014.

[MS17] R. Martins and J. Sherry. Lisbon wedding: Seating arrangements using MaxSAT. In *MaxSAT Evaluation 2017: Solver and Benchmark Descriptions*, volume B-2017-2 of *Department of Computer Science Series of Publications B*, pages 25–26. University of Helsinki, 2017.

[MZNN15a] R. Mangal, X. Zhang, A. V. Nori, and M. Naik. A user-guided approach to program analysis. In E. D. Nitto, M. Harman, and P. Heymans, editors, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 462–473. ACM, 2015.

[MZNN15b] R. Mangal, X. Zhang, A. V. Nori, and M. Naik. Volt: A lazy grounding framework for solving very large MaxSAT instances. In M. Heule and S. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 299–306. Springer, 2015.

[Nad18] A. Nadel. Solving MaxSAT with bit-vector optimization. In O. Beyersdorff and C. M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2018.

[NB14] N. Narodytska and F. Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In C. E. Brodley and P. Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2717–2723. AAAI Press, 2014.

[NLH+04] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. Understanding random SAT: beyond the clauses-to-variables ratio. In M. Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 438–452. Springer, 2004.

[NMJ+15] M. Neves, R. Martins, M. Janota, I. Lynce, and V. M. Manquinho. Exploiting resolution-based representations for MaxSAT solving. In M. Heule and S. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer*

Science, pages 272–286. Springer, 2015.

[NW99] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1999.

[NWJ16a] A. Niskanen, J. P. Wallner, and M. Järvisalo. Optimal status enforcement in abstract argumentation. In S. Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1216–1222. IJCAI/AAAI Press, 2016.

[NWJ16b] A. Niskanen, J. P. Wallner, and M. Järvisalo. Synthesizing argumentation frameworks from examples. In G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 551–559. IOS Press, 2016.

[OLH+13] T. Ogawa, Y. Liu, R. Hasegawa, M. Koshimura, and H. Fujita. Modulo based CNF encoding of cardinality constraints and its application to MaxSAT solvers. In *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013*, pages 9–17. IEEE Computer Society, 2013.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, 1994.

[Par02] J. D. Park. Using weighted MAX-SAT engines to solve MPE. In R. Dechter and R. S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.*, pages 682–687. AAAI Press / The MIT Press, 2002.

[PFL14] P. Parviainen, H. S. Farahani, and J. Lagergren. Learning bounded tree-width Bayesian networks using integer linear programming. In J. Corander and S. Kaski, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 751–759. JMLR.org, 2014.

[PRB18] T. Paxian, S. Reimer, and B. Becker. Dynamic polynomial watchdog encoding for solving weighted MaxSAT. In O. Beyersdorff and C. M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 2018.

[Rei87] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[SA18] A. Shabani and B. Alizadeh. PMTP: A MAX-SAT based approach to detect hardware trojan using propagation of maximum transition probability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[Sai15] P. Saikko. Re-implementing and extending a hybrid SAT-IP approach to maximum satisfiability. Master's thesis, University of Helsinki, 2015.

[SBJ16] P. Saikko, J. Berg, and M. Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In N. Creignou and D. L. 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.

[Sin05] C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In P. van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005.

[SMJ15] P. Saikko, B. Malone, and M. Järvisalo. MaxSAT-based cutting planes for learning graphical models. In L. Michel, editor, *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, volume 9075 of *Lecture Notes in Computer Science*, pages 347–356. Springer, 2015.

[SMV+07] S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007, Austin, Texas, USA, November 11-14, 2007, Proceedings*, pages 13–19. IEEE Computer Society, 2007.

[SZGN17] X. Si, X. Zhang, R. Grigore, and M. Naik. Maximum satisfiability in software analysis: Applications and techniques. In R. Majumdar and V. Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10427 of *Lecture Notes in Computer Science*, pages 68–94. Springer, 2017.

[TDFDGDSJ+17] A. B. Trindade, R. De Faria Degelo, E. Galvão Dos Santos Junior, H. I. Ismail, H. Cruz Da Silva, and L. C. Cordeiro. Multicore model checking and maximum satisfiability applied to hardware-software partitioning. *International Journal of Engineering Science*, 9(6):570–582, 2017.

[TLM16] 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.

[vdTHB12] P. van der Tak, M. Heule, and A. Biere. Concurrent cube-and-conquer - (poster presentation). In A. Cimatti and R. Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 475–476. Springer, 2012.

[War98] J. P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.

[WJH18] A. A. Wakrime, S. Jabbour, and N. Hameurlain. A MaxSAT based approach for QoS cloud services. *International Journal of Parallel, Emergent and Distributed Systems*, 2018.

[WNJ17] J. P. Wallner, A. Niskanen, and M. Järvisalo. Complexity results and algorithms for extension enforcement in abstract argumentation. *Journal of Artificial Intelligence Research*, 60:1–40, 2017.

[Wol98] L. A. Wolsey. *Integer Programming*. John Wiley, 1998.

[WQL09] G. T. Wickramaarachchi, W. H. Qardaji, and N. Li. An efficient framework for user authorization queries in RBAC systems. In B. Carminati and J. Joshi, editors, *14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, Stresa, Italy, June 3-5, 2009, Proceedings*, pages 23–32. ACM, 2009.

[XHHL08] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

[XRS03] H. Xu, R. A. Rutenbar, and K. Sakallah. sub-SAT: a formulation for relaxed Boolean satisfiability with applications in routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):814–820, 2003.

[ZB12] L. Zhang and F. Bacchus. MAXSAT heuristics for cost optimal planning. In J. Hoffmann and B. Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012.

[ZLMA12] Z. Zhu, C. M. Li, F. Manyà, and J. Argelich. A new encoding from MinSAT into MaxSAT. In M. Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 455–463. Springer, 2012.

[ZMNN16] X. Zhang, R. Mangal, A. V. Nori, and M. Naik. Query-guided maximum satisfiability. In R. Bodík and R. Majumdar, edi-

tors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 109–122. ACM, 2016.

[ZWM11] C. S. Zhu, G. Weissenbacher, and S. Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In P. Bjesse and A. Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 63–66. FMCAD Inc., 2011.