UNIVERSITY OF HELSINKI

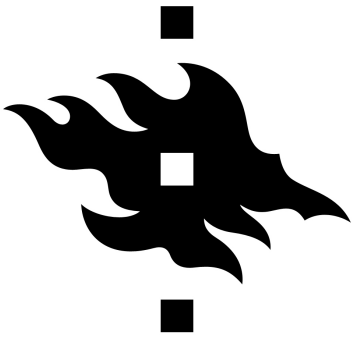
Department of Mathematics and Statistics

Master's Thesis

Preprocessing and Stochastic Local Search in Maximum Satisfiability

Marcus Leivo

June 18, 2020



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Supervisors: *Doctor* Jeremias Berg, *Associate Professor* Matti Järvisalo
Examiners: *Doctor* Jeremias Berg, *Associate Professor* Matti Järvisalo, *Professor* Juha Kontinen

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Avdelning — Department
Faculty of Science	Department of Mathematics and Statistics

Tekijä — Författare — Author

Marcus Leivo

Työn nimi — Arbetets titel — Title

Preprocessing and Stochastic Local Search in Maximum Satisfiability

Oppiaine — Läroämne — Subject

Mathematics

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's Thesis	June 18, 2020	57 pages

Tiivistelmä — Referat — Abstract

Problems which ask to compute an optimal solution to its instances are called optimization problems. The maximum satisfiability (MaxSAT) problem is a well-studied combinatorial optimization problem with many applications in domains such as cancer therapy design, electronic markets, hardware debugging and routing. Many problems, including the aforementioned ones, can be encoded in MaxSAT. Thus MaxSAT serves as a general optimization paradigm and therefore advances in MaxSAT algorithms translate to advances in solving other problems.

In this thesis, we analyze the effects of MaxSAT preprocessing, the process of reformulating the input instance prior to solving, on the perceived costs of solutions during search. We show that after preprocessing most MaxSAT solvers may misinterpret the costs of non-optimal solutions. Many MaxSAT algorithms use the found non-optimal solutions in guiding the search for solutions and so the misinterpretation of costs may misguide the search.

Towards remedying this issue, we introduce and study the concept of locally minimal solutions. We show that for some of the central preprocessing techniques for MaxSAT, the perceived cost of a locally minimal solution to a preprocessed instance equals the cost of the corresponding reconstructed solution to the original instance.

We develop a stochastic local search algorithm for MaxSAT, called LMS-SLS, that is prepended with a preprocessor and that searches over locally minimal solutions. We implement LMS-SLS and analyze the performance of its different components, particularly the effects of preprocessing and computing locally minimal solutions, and also compare LMS-SLS with the state-of-the-art SLS solver SATLike for MaxSAT.

Avainsanat — Nyckelord — Keywords

Maximum satisfiability, preprocessing, stochastic local search

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Optimization refers to the process of searching for a best solution to a given problem where "best" depends on the underlying problem. We face optimization problems in our lives on a daily basis whether it be for example finding the shortest route to work or adjusting our schedules so as to allow us to meet our deadlines. Problems may involve both *hard* constraints which need to be adhered to and *soft* constraints which may be ignored though not without some sort of penalty. In finding a route to work, an example of a hard constraint is that the route should eventually lead one to work instead of a local coffee shop. As an example of soft constraints, each road poses one, telling us not to traverse it as in doing so we accumulate kilometers traveled. Of course, some of the roads need to be traversed unless one lives at their workplace, so some of these soft constraints will and must be violated. A shortest route from home to work is one where the sum of the penalties (kilometers) related to the violated constraints (roads to travel) is as low as possible.

In practice, myriads of **NP**-hard optimization problems arise. Multiple polynomial-time algorithms for finding the shortest route have been developed and formalized during the 20th century [1–3], but for the similar problem of finding the shortest route for a road trip to visit attractions around the country (known as the *traveling salesperson problem* [4–6]) no polynomial-time algorithm is currently known. The question of whether or not there is a polynomial-time algorithm to solve the latter problem remains an open question and the person who proves that there is or is not such an algorithm will be rewarded one million U.S. dollars by the Clay Mathematics Institute [7] as the proof would answer whether $\mathbf{P} = \mathbf{NP}$ or not [5, 8].

Some of the most interesting problems in practice are ones with no known algorithms that are polynomial-time even in the worst case. Such problems include data visualization [9], hardware and software verification [10, 11] and routing air traffic [12]. Although polynomial-time algorithms to **NP**-hard optimization problems arising in practical applications remain elusive, practically efficient algorithms for them are nevertheless sought after due to their importance. Improvements in these algorithms may save significant amounts of time and resources. Through ingenuity and domain knowledge, practically efficient algorithms for finding good quality (or even optimal) solutions to **NP**-hard optimization problems have been devised [5, 13–18] defying theoretical restrictions.

Since there are numerous distinct optimization problems, designing specific algorithms for each problem is rather time consuming. Decades of research has gone into the study of certain optimization problems such as the traveling salesperson problem [4–6, 16]. As such, coming up with algorithms efficient in practice specific for new theoretically difficult problems has the potential to take years if not decades. Therefore, one might opt for a declarative approach such as *integer programming* (IP) [17, 18] and *maximum satisfiability* (MaxSAT) [8, 19]. Rather than developing a separate algorithm for a given problem, one can encode the problem in a constraint language such as propositional logic underlying the maximum satisfiability problem. For MaxSAT, several algorithms that are efficient in solving many types of practical instances exist [13–15, 20] and so suitable encodings of problems in propositional

logic allow exploiting the performance of MaxSAT solvers in other problem domains. Both IP and MaxSAT are used as declarative paradigms [9, 12, 17, 21], IP being a classical approach and MaxSAT a relatively young one. In this thesis, we focus on MaxSAT.

MaxSAT is a generalization of the famous *boolean satisfiability* (SAT) [8, 19] problem. In SAT, the constraints are *clauses* i.e. disjunctions of boolean variables and their negations called *literals*. The SAT problem asks if there is a way to assign the boolean variables so that each clause is satisfied. In MaxSAT, a clause is *hard* if it must be satisfied and *soft* otherwise. Soft clauses are allowed to have weights associated with them, denoting their relative importance. The MaxSAT problem then asks to assign the boolean variables so (i) that all hard clauses are satisfied and (ii) the sum of the weights of soft clauses left unsatisfied is minimized (or equivalently, the sum of the weights of satisfied clauses is maximized). Owing to the expressivity of propositional logic, many practical problems such as cancer therapy design [22], electronic markets [23], hardware debugging [24, 25] and scheduling [21, 26, 27] have efficient encodings in propositional logic. As efficient MaxSAT solvers exist [13–15, 20, 28], MaxSAT solvers are used in solving these problems.

The MaxSAT encoding of a problem can be modified by applying MaxSAT *preprocessing* before solving. Preprocessing refers to the process of reformulating the input instance prior to solving with the aim that the time it takes to preprocess the input and run a solver on the preprocessed instance is lower than directly running the solver on the input instance [29]. In this thesis, we examine the effects of preprocessing on the relative order of solutions under cost.

Proofs of correctness of preprocessing techniques in MaxSAT are based on showing that the apparent cost of an optimal solution τ equals the cost of the (optimal) solution, reconstructed from τ, to the original instance [30]. However, as we will show in this thesis, these proofs do not extend to arbitrary solutions. This implies that the order of solutions in terms of cost may change after solution reconstruction. Our analysis shows this problem occurs not only in theory but in practice as well.

To mitigate this issue, we introduce the concept of *locally minimal* solutions—a concept far less restrictive than optimality—and show that for locally minimal solutions to preprocessed instances, their apparent cost equals the cost of the corresponding reconstructed solutions given that only preprocessing techniques that preserve locally minimal solutions are used. Any MaxSAT solver limiting its search to locally minimal solutions is then guaranteed to not misinterpret the costs of solutions if the used preprocessing techniques all preserve locally minimal solutions.

For evaluating the effectiveness of limiting search to locally minimal solutions, we introduce a novel *stochastic local search* (SLS) [8] algorithm for MaxSAT designed to search over locally minimal solutions. Stochastic local search for SAT and MaxSAT is a search procedure which begins by generating an initial assignment and iteratively modifying it by reversing the truth value of a heuristically chosen variable [29]. Stochastic local search are *incomplete* i.e. they do not prove the optimality of solutions as *complete* solvers do [29]. The advantage of incomplete approaches is that they need not spend time in proving optimality, and that most scale better to

bigger instances, finding good quality solutions in limited time [8].

We implement the described SLS solver, titled LMS-SLS, and evaluate the effectiveness of its main components. In particular, we show that preprocessing combined with a heuristic to limit the search to locally minimal solutions yields the best results. Moreover, we show that including a preprocessing technique that does not preserve locally minimal solutions degrades the performance of the solver. Finally, we compare our implementation of LMS-SLS with a state-of-the-art SLS solver for MaxSAT called SATLike [14] and show that the two exhibit complementary performance.

The contents of this work are organized as follows. In Chapter 2, we go through the needed background on propositional logic, satisfiability and maximum satisfiability and briefly overview different approaches for MaxSAT solving. We then cover background on stochastic local search in Chapter 3. Preprocessing for MaxSAT is discussed in Chapter 4. In Chapter 5, we discuss potential issues in the way in which most MaxSAT solvers apply preprocessing, and introduce and study the concept of locally minimal solutions. In Chapter 6, we cover the SLS algorithm LMS-SLS for MaxSAT and evaluate the performance of its components and also compare LMS-SLS with SATLike, on top of which our implementation of LMS-SLS is built on. Lastly, we conclude the thesis.

The results of this thesis are based on a paper [31]—to appear in ECAI 2020—which is a joint work of the author and the supervisors of this thesis, Jeremias Berg and Matti Järvisalo.

2 Preliminaries

The main goal of this chapter is to cover the maximum satisfiability problem, an extension of the boolean satisfiability problem. The first section goes through the necessary background on propositional logic and the boolean satisfiability problem. In the second section we discuss maximum satisfiability.

2.1 Propositional Logic and Satisfiability

Propositional logic is a formal language consisting of formulas of simple true-or-false statements joined with connectives. A *propositional formula* is constructed by connecting *boolean variables* (true or false statements) with connectives such as $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, which have the semantic meaning of "not", "and", "or", "implies", "is equivalent to", respectively. The symbol $\wedge$ is called a *conjunct* and the symbol $\vee$ a *disjunct*.

We do not define the syntax of propositional logic in full as for our purposes the following subset of propositional formulas suffices. For a comprehensive treatment of propositional logic, see e.g. [32]. We use B to denote a fixed, countably infinite, set of boolean variables.

Definition 2.1 (Syntax of CNF formulas)**.** *A* literal *is a boolean variable $x \in B$ or its negation $\neg x$ (alternatively denoted by $\bar{x}$). A disjunction of a finite non-empty collection of literals l_i is a* clause*, denoted by $(l_1 \vee \ldots \vee l_n)$. A formula in* conjunctive normal form *(a CNF formula for short) is any conjunction of a finite collection of clauses C_i, denoted by $C_1 \wedge \ldots \wedge C_n$.*

Example 2.2. Let $x, y, z \in B$. Then the following are CNF formulas:

$$(x \vee y) \wedge (\neg x \vee \neg y \vee z), \quad (y \vee \bar{x}) \wedge (x) \wedge (\bar{y}).$$

When speaking of clauses and formulas we often refer to the variables contained in them. We use $\textsc{Var}(C)$ and $\textsc{Var}(\mathcal{F})$ to denote the set of variables occurring in the clause C and in the formula $\mathcal{F}$ respectively.

Definition 2.3 (Assignments)**.** *A function $\tau\colon B \to \{0, 1\}$ is an assignment.*

We identify the value 1 with *true* and the value 0 with *false* when speaking of assignments. An assignment τ can also be viewed as a set of literals it sets to true. That is, we identify an assignment τ with the set

$$\{x \mid \tau(x) = 1\} \cup \{\neg x \mid \tau(x) = 0\}.$$

Now $\tau(l) = 1$ is equivalent to $l \in \tau$. Given a literal l, we define τ_l as $\tau \setminus \{\neg l\} \cup \{l\}$. When convenient, we identify assignments that agree on a set of variables. For example, by $\tau = \{x, \bar{y}\}$ we mean any assignment τ that sets x to true and y to false.

We are now ready to define the semantics of CNF formulas.

Definition 2.4 (Semantics of CNF formulas)**.** *Let $\mathcal{F}$ be a CNF formula. An assignment τ* satisfies *a clause $C \in \mathcal{F}$ if $\tau(x) = 1$ for some $x \in C$ or $\tau(x) = 0$ for some $\neg x \in C$. A CNF formula $\mathcal{F}$ is* satisfiable *if there is an assignment τ for $\mathcal{F}$ that satisfies each of its clauses. Such an assignment τ is a* satisfying assignment *of $\mathcal{F}$. If $\mathcal{F}$ has no satisfying assignments, $\mathcal{F}$ is* unsatisfiable*.*

The order of literals in a clause and the order of clauses in a formula do not matter, and we identify clauses $(l_1 \vee \cdots \vee l_n)$ with the set $\{l_1, \ldots, l_n\}$ and CNF formulas $C_1 \wedge \cdots \wedge C_n$ with the set $\{C_1, \ldots, C_n\}$.

Given a clause C and a CNF formula $\mathcal{F}$ we use the notations $\tau(C) = 1$ and $\tau(\mathcal{F}) = 1$ to mean that τ satisfies the clause C and the formula $\mathcal{F}$ respectively. We define $\tau(C) = 0$ and $\tau(\mathcal{F}) = 0$ similarly.

Omitting other propositional formulas and focusing only on CNF formulas is only a minor restriction as there is a linear-time algorithm called the "Tseitin transformation" to transform any propositional formula into a CNF formula that is tightly connected to the original formula [33]. Given a satisfying assignment $\tau_{\mathcal{F}}$ for the Tseitin transformation $\mathcal{F}$ of a propositional formula $\mathcal{F}'$, the assignment $\tau_{\mathcal{F}}$ is also a satisfying assignment of $\mathcal{F}'$.

The problem of finding a satisfying assignment for a propositional formula is called the satisfiability problem, abbreviated SAT. The problem where the set of propositional formulas is restricted to CNF formulas is the CNF-SAT problem. Due to the Tseitin transformation, SAT has a linear-time reduction to CNF-SAT and so we will use SAT to refer to CNF-SAT. SAT is **NP**-complete [34].

Definition 2.5 (SAT)**.** *Given a CNF formula $\mathcal{F}$, does $\mathcal{F}$ have a satisfying assignment?*

Example 2.6. The CNF formula $(x \vee y) \wedge (\neg x \vee \neg y \vee z)$ is satisfiable, while the formula $(y \vee \bar{x}) \wedge (x) \wedge (\bar{y})$ is unsatisfiable.

Let us look at a concrete example on how CNF formulas can be used to model a practical problem by *encoding* it in CNF. An encoding of a problem is a way of representing the instances of the problem.

Example 2.7. Suppose that a company has hired Alex, Blair, Carey and Dale to work at the company's warehouse. The company is now considering whom of the four to assign to work the morning shifts and whom to assign the evening shifts. However, HR has learned that the four share a history: (i) Alex harbors resentment towards Blair as Blair insulted Alex in high school, (ii) Carey is mad at Alex for not settling things with Blair, (iii) Blair refuses to work without their best friend Dale. To avoid any drama and maximize happiness, HR concludes that

1. Alex should not be working the same shifts as Blair,
2. Carey should not work with Alex,
3. Blair should be assigned with Dale.

To model this issue in CNF we first introduce eight variables, two for each person:

$$m_a, m_b, m_c, m_d, e_a, e_b, e_c, e_d.$$

The intuition here is that if m_a or e_a is true then Alex is assigned the morning shifts or evening shifts respectively. In addition to the above constraints, we need to make sure each person is assigned exactly one shift. This is achieved with the pairs of clauses $(m_i \vee e_i) \wedge (\bar{m}_i \vee \bar{e}_i)$ where $i \in \{a, b, c, d\}$; if the first clause is true then the person corresponding to i is assigned at least one shift and if the second is true the person is assigned at most one shift. The first constraint above can be expressed as the pair of clauses $(m_a \vee m_b) \wedge (e_a \vee e_b)$; either Alex or Blair works in the morning shifts and likewise either one of them works the evening shifts. The second one is similarly $(m_a \vee m_c) \wedge (e_a \vee e_c)$. Finally, the third constraint is expressed in CNF by $(\bar{m}_b \vee m_d) \wedge (\bar{e}_b \vee e_d)$; the first clause states if Blair works the morning shift, then so should Dale (the intuition for the second clause is similar). The formula in full is

$$\begin{aligned}
\mathcal{F}_1 = {} & (m_a \vee e_a) \ \wedge (\bar{m}_a \vee \bar{e}_a) \wedge (m_b \vee e_b) \ \wedge (\bar{m}_b \vee \bar{e}_b) \wedge & (1)\\
& (m_c \vee e_c) \ \wedge (\bar{m}_c \vee \bar{e}_c) \wedge (m_d \vee e_d) \ \wedge (\bar{m}_d \vee \bar{e}_d) \wedge & (2)\\
& (m_a \vee m_b) \wedge (e_a \vee e_b) \ \wedge (m_a \vee m_c) \wedge (e_a \vee e_c) \ \wedge & (3)\\
& (\bar{m}_b \vee m_d) \wedge (\bar{e}_b \vee e_d). & (4)
\end{aligned}$$

The process of encoding a problem in SAT requires care though. For example, the constraint $(m_a \vee m_b) \wedge (e_a \vee e_b)$ on row 3 alone is not sufficient to encompass the constraint "Alex should not work with Blair" as the two clauses are simultaneously satisfied by assigning Alex both morning and evening shifts! Fortunately, adding the constraints that each person works exactly one shift (rows 1 and 2) remedies this issue.

Astute readers may have noticed that if each person is assigned exactly one shift, then surely m_i must be true if and only if e_i is false. This observation is indeed correct and we may in fact replace each occurrence of e_i with $\bar{m}_i$ in $\mathcal{F}_1$. Not only does this eliminate four of the variables, but also eight clauses as well. Namely, now the conjunctions $(m_i \vee e_i) \wedge (\bar{m}_i \vee \bar{e}_i)$ equal $(m_i \vee \bar{m}_i) \wedge (\bar{m}_i \vee m_i) = (m_i \vee \bar{m}_i)$. The clauses $(m_i \vee \bar{m}_i)$ are called *tautologies* as they are true no matter if we assign the boolean variables m_i to true or false and hence they may be removed. For this reason we often tacitly assume no tautologies.

Replacing the variables e_i with $\bar{m}_i$ and removing tautologies leads to the simplified formula

$$\mathcal{F}_2 = \ (m_a \vee m_b) \wedge (\bar{m}_a \vee \bar{m}_b) \wedge (m_a \vee m_c) \wedge (\bar{m}_a \vee \bar{m}_c) \wedge (\bar{m}_b \vee m_d) \wedge (m_b \vee \bar{m}_d).$$

The formula $\mathcal{F}_2$ has two satisfying assignments[1], namely

$$\tau_1 = \{m_a, \bar{m}_b, \bar{m}_c, \bar{m}_d\}, \quad \tau_2 = \{\bar{m}_a, m_b, m_c, m_d\}$$

from which we may conclude that if Alex is assigned the morning shifts the rest should be assigned to evening shifts and vice versa.

[1]Here we identify assignments agreeing on a set of variables for the first time.

Example 2.7 can be viewed as an instance of a type of shift-scheduling problem [27] which asks is there is a way of assigning shifts to n employees while satisfying constraints of the form "employee i wishes not to work with employee j" and "employee i wishes to work with employee j".

The representation in CNF seen in Example 2.7 can be generalized for an arbitrary instance of the underlying shift-scheduling problem: given n employees along with their preferences, the encoding for the instance is

$$\{(m_i \vee m_j), (\bar{m_i} \vee \bar{m_j}) \mid \text{employee } i \text{ does not want to work with employee } j\} \cup$$
$$\{(m_i \vee \bar{m_j}), (\bar{m_i} \vee m_j) \mid \text{employee } i \text{ wants to work with employee } j\}.$$

If τ is a satisfying assignment for the instance, $m_i \in \tau$ is interpreted as "employee i should be assigned the morning shifts".

In an ideal world, each instance of this shift-scheduling problem would have a corresponding satisfying assignment; each employee's wishes can be satisfied. Sadly, in reality there is often no way of pleasing everybody.

Example 2.8. Consider again the situation of Example 2.7 and the formula

$$\mathcal{F}_2 = (m_a \vee m_b) \wedge (\bar{m_a} \vee \bar{m_b}) \wedge (m_a \vee m_c) \wedge (\bar{m_a} \vee \bar{m_c}) \wedge (\bar{m_b} \vee m_d) \wedge (m_b \vee \bar{m_d}).$$

Suppose in addition that Carey would like to work without having to see Blaire. This is encoded via the constraint $(m_b \vee m_c) \wedge (\bar{m_b} \vee \bar{m_c})$. Adding this constraint to $\mathcal{F}_2$ leads to the instance

$$\mathcal{F}_3 = (m_a \vee m_b) \wedge (\bar{m_a} \vee \bar{m_b}) \wedge (m_a \vee m_c) \wedge (\bar{m_a} \vee \bar{m_c}) \wedge$$
$$(\bar{m_b} \vee m_d) \wedge (m_b \vee \bar{m_d}) \wedge (m_b \vee m_c) \wedge (\bar{m_b} \vee \bar{m_c}).$$

The satisfying assignments τ_1, τ_2 for $\mathcal{F}_2$ from Example 2.7 are the only possible satisfying assignments for $\mathcal{F}_3$ as $\mathcal{F}_2 \subseteq \mathcal{F}_3$. However, $\tau_1((m_b \vee m_c)) = 0$ and $\tau_2((\bar{m_b} \vee \bar{m_c})) = 0$ so $\mathcal{F}_3$ has no satisfying assignments.

Example 2.8 demonstrates that not all constraints can always be satisfied simultaneously. Nevertheless, we can ask for the next best thing. Namely, what is the maximum number of satisfied constraints that can be satisfied simultaneously or equivalently the minimum number of constraints left unsatisfied? The next section discusses maximum satisfiability, an optimization variant of the SAT problem.

2.2 Maximum Satisfiability

A satisfying assignment for a CNF formula $\mathcal{F}$ satisfies *all* the clauses $C \in \mathcal{F}$. As witnessed in e.g. Example 2.8, such an assignment does not necessarily exist. A natural extension to the question posed in the SAT problem (Problem 2.5) is to ask for the maximum number of clauses that can be simultaneously satisfied by an assignment. This extension is known as the maximum satisfiability problem (MaxSAT) [19].

A further extension of MaxSAT allows clauses to have *weights* associated with them which serve as a gauge describing the importance of C being satisfied. This extension of MaxSAT is known as the *Weighted* Maximum Satisfiability problem [19].

Another extension of MaxSAT is the *Partial* MaxSAT problem. An instance of Partial MaxSAT has two sets of clauses; a set of *hard* and a set of *soft* clauses. Any solution to an instance of the Partial MaxSAT problem must satisfy all the hard clauses, while an optimal solution also maximizes the number of satisfied soft clauses [19].

Dividing clauses into hard and soft clauses while also allowing the soft clauses to have weights leads to the *Weighted Partial* MaxSAT problem [19]. The first subsection covers the widely used way of defining Weighted Partial MaxSAT which we call *clause-centric* MaxSAT. In the second subsection, we cover an alternative *literal-centric* definition for MaxSAT [30] as it is better suited for preprocessing which we will focus on later in the thesis. From now on, we will refer to the Weighted Partial MaxSAT simply as MaxSAT.

2.2.1 Clause-Centric MaxSAT

SAT may be extended by allowing each clause in an instance of SAT to have a weight associated to it. Intuitively, the weight describes the importance of C being satisfied relative to other clauses. If a clause C has no weight associated with it, we interpret C as a hard clause which is a clause that must be satisfied. Clauses paired with a weight are called soft clauses. This is the standard way of extending SAT to an optimization problem [30].

Definition 2.9 (Clause-Centric MaxSAT)**.** *A clause-centric MaxSAT instance $\mathcal{F}$ is a tuple (F_h, F_s, w) where F_h and F_s are sets of clauses and w is a function from F_s to $\mathbb{Z}_{>0} = \{1, 2, 3, \ldots\}$.*

We extend the notation used with CNF formulas in a natural way to clause-centric MaxSAT instances e.g. $\textsc{Var}(\mathcal{F})$ is the set of variables occurring in the clauses in $\mathcal{F}$. In addition, we use $C \in \mathcal{F}$ as a shorthand for $C \in F_h \cup F_s$ where $\mathcal{F} = (F_h, F_s, w)$.

Definition 2.10. *An assignment τ is a* solution *to $\mathcal{F} = (F_h, F_s, w)$ if $\tau(F_h) = 1$. The cost of a solution τ to $\mathcal{F}$ is defined as*

$$\textsc{Cost}(\mathcal{F}, \tau) = \sum_{C \in F_s} w(C) \cdot (1 - \tau(C)).$$

For non-solutions τ, we set $\textsc{Cost}(\mathcal{F}, \tau) = \infty$. A solution τ to $\mathcal{F}$ is optimal if for all solutions τ' to $\mathcal{F}$

$$\textsc{Cost}(\mathcal{F}, \tau) \leq \textsc{Cost}(\mathcal{F}, \tau').$$

The MaxSAT problem asks to find an optimal solution for a given clause-centric MaxSAT instance. Due to the **NP**-completeness of SAT, MaxSAT is **NP**-hard.

Definition 2.11 (MaxSAT)**.** *Given a MaxSAT instance $\mathcal{F}$, find an optimal solution to $\mathcal{F}$.*

Instances of SAT can be viewed as MaxSAT instances. Finding a satisfying assignment for a CNF formula F is equivalent to finding a solution of cost zero to the MaxSAT instance $(F, \emptyset, w)$.

Example 2.12. Consider the unsatisfiable CNF formula

$$\begin{aligned}\mathcal{F}_3 = (m_a \vee m_b) \wedge (\bar{m_a} \vee \bar{m_b}) \wedge (m_a \vee m_c) \wedge (\bar{m_a} \vee \bar{m_c}) \wedge \\ (\bar{m_b} \vee m_d) \wedge (m_b \vee \bar{m_d}) \wedge (m_b \vee m_c) \wedge (\bar{m_b} \vee \bar{m_c})\end{aligned}$$

from Example 2.8. To avoid lawsuits management deems it absolutely necessary that Alex does not to work with Blaire. On the other hand, management concludes that Carey can work with Alex or Blaire although it would be preferable Carey did not have to. Likewise Blaire and Dale can work separate shifts but preferably they worked same shift. These preferences can be modeled for example with the following MaxSAT instance, where the superscripts denote the clause weights:

$$\begin{aligned}\mathcal{F}_3^w = (m_a \vee m_b) \ \wedge (\bar{m_a} \vee \bar{m_b}) \ \wedge (m_a \vee m_c)^3 \wedge (\bar{m_a} \vee \bar{m_c})^3 \wedge \\ (\bar{m_b} \vee m_d)^1 \wedge (m_b \vee \bar{m_d})^1 \wedge (m_b \vee m_c)^1 \wedge (\bar{m_b} \vee \bar{m_c})^1.\end{aligned}$$

Here management decided that it is much more critical to have Alex and Carey work separate shifts (weight 3) than having Blaire work the same shifts as Dale (weight 1) or having Blaire and Carey work separate shifts (weight 1). Notice that if any of these three constraints is not satisfied it leads to cost increasing by one or three and not two times one or two times three. This is because e.g. in the pair of clauses $(m_a \vee m_c)^3 \wedge (\bar{m_a} \vee \bar{m_c})^3$ the first clause contains m_a and the latter its negation $\bar{m_a}$. Hence at least one of them is always satisfied. The mentioned two clauses can be viewed as a *group* [35], where cost is increased by a constant if both clauses in the group are not satisfied.

The MaxSAT instance $\mathcal{F}_3^w$ now has multiple solutions as satisfying all of the clauses is no longer mandatory. Examples of solutions are

$$\tau_1 = \{m_a, \bar{m_b}, m_c, \bar{m_d}\}, \quad \tau_2 = \{m_a, \bar{m_b}, \bar{m_c}, \bar{m_d}\}, \quad \tau_3 = \{\bar{m_a}, m_b, m_c, m_d\}.$$

The cost of the solutions are 3, 1 and 1 respectively. The solutions τ_2 and τ_3 are optimal since one is a lower bound for the cost of solutions as $\mathcal{F}_3$ is unsatisfiable. Any assignment that assigns both m_a and m_b to true or to false is not a solution since then either the hard clause $(m_a \vee m_b)$ or $(\bar{m_a} \vee \bar{m_b})$ is left unsatisfied. For example $\tau_4 = \{m_a, m_b, \bar{m_c}, m_d\}$ is not a solution.

2.2.2 Literal-Centric MaxSAT

Literal-centric MaxSAT is an alternative way of defining MaxSAT, essentially equivalent to clause-centric MaxSAT. Instead of assigning weights to clauses, weights are assigned to variables. This approach to MaxSAT has benefits in preprocessing as we will see in Chapter 4.

Definition 2.13 (Literal-Centric MaxSAT)**.** *Let F be a set of clauses and let w be a function from a subset $\mathcal{S}(F) \subseteq \textsc{Var}(F)$ to $\mathbb{Z}_{>0}$. The pair $\mathcal{F}^L = (F, w)$ is a literal-centric MaxSAT instance.*

The variables in the set $\mathcal{S}(F)$ are *soft variables* and the clauses containing soft variables are *soft clauses*. Clauses with no soft variables are *hard clauses*. We use the notation $\mathcal{S}(\mathcal{F}^L)$ to denote the set of soft variables in $\mathcal{F}^L$. In addition, we adopt the convention of distinguishing literal-centric MaxSAT instances from clause-centric MaxSAT instances by using the superscript L with literal-centric instances i.e. $\mathcal{F}^L$ denotes a literal-centric instance and $\mathcal{F}$ a clause-centric instance. When clear from context, we use $\mathcal{F} = (F, w)$ and F interchangeably.

Outside this chapter, we will see literal-centric instances solely in the context of preprocessing. Each solution τ to a preprocessed instance $\mathcal{P}(\mathcal{F})$ (which is always literal-centric) is tied to a solution $\text{REC}(\tau)$ to a clause-centric instance $\mathcal{F}$. Thus we reserve the term "cost" for solutions to clause-centric instances. Preprocessing and its effects are discussed in chapters 4 and 5.

Definition 2.14. *An assignment τ is a solution to $\mathcal{F}^L = (F, w)$ if $\tau(F) = 1$. The* apparent cost *of a solution τ is defined as*

$$\textsc{Appar-Cost}(\mathcal{F}^L, \tau) = \sum_{x \in \mathcal{S}(\mathcal{F}^L)} \tau(x) w(x).$$

A solution τ to $\mathcal{F}^L$ is optimal if for any other solution τ' to $\mathcal{F}^L$

$$\textsc{Appar-Cost}(\mathcal{F}^L, \tau) \leq \textsc{Appar-Cost}(\mathcal{F}^L, \tau').$$

Example 2.15. Consider again the instance

$$\begin{aligned}\mathcal{F}_3 = (m_a \vee m_b) \wedge (\bar{m_a} \vee \bar{m_b}) \wedge (m_a \vee m_c) \wedge (\bar{m_a} \vee \bar{m_c}) \wedge \\ (\bar{m_b} \vee m_d) \wedge (m_b \vee \bar{m_d}) \wedge (m_b \vee m_c) \wedge (\bar{m_b} \vee \bar{m_c})\end{aligned}$$

from Example 2.8. To model the situation of Example 2.12 via literal-centric MaxSAT we introduce three new soft variables s_1, s_2 and s_3 with weights $3, 1$ and 1 respectively to obtain

$$\begin{aligned}\mathcal{F}_3^L = (m_a \vee m_b) \quad \wedge (\bar{m_a} \vee \bar{m_b}) \quad \wedge (m_a \vee m_c \vee s_1) \wedge (\bar{m_a} \vee \bar{m_c} \vee s_1) \wedge \\ (\bar{m_b} \vee m_d \vee s_2) \wedge (m_b \vee \bar{m_d} \vee s_2) \wedge (m_b \vee m_c \vee s_3) \wedge (\bar{m_b} \vee \bar{m_c} \vee s_3).\end{aligned}$$

Since all clauses of a literal-centric instance must be satisfied, any solution to $\mathcal{F}_3^L$ sets some soft variables to true as removing the soft variables from the clauses in $\mathcal{F}_3^L$ leads to an unsatisfiable instance.

A solution τ may have a soft variable assigned to one, unnecessarily increasing the cost. The following solutions have apparent costs three and four respectively:

$$\tau_1 = \{m_a, \bar{m_b}, m_c, \bar{m_d}, s_1, \bar{s_2}, \bar{s_3}\}, \quad \tau_2 = \{m_a, \bar{m_b}, m_c, \bar{m_d}, s_1, \bar{s_2}, s_3\}.$$

Here τ_2 would remain as a solution even if s_3 was assigned to zero instead.

In theory, the literal-centric view on MaxSAT differs little from the clause-centric view as there is a simple way to convert a literal-centric instance $\mathcal{F}^L$ to a clause-centric instance $\mathcal{F}$ and vice versa. An instance $\mathcal{F}^L = (F, w)$ is transformed into

a corresponding clause-centric instance $\mathcal{F}$ by emulating literal weights with soft clauses:

$$\mathcal{F} = (F, \{(\bar{s}) \mid s \in \mathcal{S}\left(\mathcal{F}^L\right)\}, w'),$$

where w' maps the soft clauses $(\bar{s})$ to $w(s)$. Each solution τ^L to $\mathcal{F}^L$ is also a solution to $\mathcal{F}$ where the apparent cost of τ^L equals its cost. Likewise any solution τ to $\mathcal{F}$ is a solution to $\mathcal{F}^L$ where the cost of τ equals its apparent cost.

To convert a clause-centric instance $\mathcal{F} = (F_h, F_s, w)$ into a literal-centric one we first introduce a new variable s_C for each $C \in F_s$ to obtain a set of clauses $F'_s = \{C \vee s_C \mid C \in F_s\}$. The literal-centric instance $\mathcal{F}^L$ corresponding to $\mathcal{F}$ is then $(F_h \cup F'_s, w')$ where w' maps each s_C to $w(C)$. This transformation preserves the solutions in the following sense: (i) given a solution τ to $\mathcal{F}$, for

$$\tau^L = (\tau \setminus \{s_C, \neg s_C \mid C \in \mathcal{F}\}) \cup \{s_C \mid C \in \mathcal{F},\ \tau(C) = 0\} \cup \{\neg s_C \mid C \in \mathcal{F},\ \tau(C) = 1\}$$

we have $\textsc{Cost}(\mathcal{F}, \tau) = \textsc{Appar-Cost}(\mathcal{F}^L, \tau^L)$ where τ^L is a solution to $\mathcal{F}^L$ and (ii) any solution τ^L to $\mathcal{F}^L$ is a solution to $\mathcal{F}$ though its cost does not necessarily equal the apparent cost as even if $\tau(C) = 1$ for a soft clause C it may be that $\tau^L(s_C) = 1$.

Example 2.16. For the clause-centric instance $\mathcal{F}$

$$(x \vee y \vee \bar{z}) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee y)^4 \wedge (\bar{y} \vee z)^9$$

the corresponding literal-centric instance is

$$(x \vee y \vee \bar{z}) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee y \vee s_1) \wedge (\bar{y} \vee z \vee s_2)$$

where the weights of s_1 and s_2 are four and nine respectively.

For

$$\begin{aligned}\mathcal{F}_3^L = {} & (m_a \vee m_b) \quad \wedge (\bar{m_a} \vee \bar{m_b}) \quad \wedge (m_a \vee m_c \vee s_1) \wedge (\bar{m_a} \vee \bar{m_c} \vee s_1) \wedge \\ & (\bar{m_b} \vee m_d \vee s_2) \wedge (m_b \vee \bar{m_d} \vee s_2) \wedge (m_b \vee m_c \vee s_3) \wedge (\bar{m_b} \vee \bar{m_c} \vee s_3).\end{aligned}$$

from Example 2.15 the corresponding clause-centric instance is

$$\begin{aligned}& (m_a \vee m_b) \quad \wedge (\bar{m_a} \vee \bar{m_b}) \quad \wedge (m_a \vee m_c \vee s_1) \wedge (\bar{m_a} \vee \bar{m_c} \vee s_1) \wedge \\ & (\bar{m_b} \vee m_d \vee s_2) \wedge (m_b \vee \bar{m_d} \vee s_2) \wedge (m_b \vee m_c \vee s_3) \wedge (\bar{m_b} \vee \bar{m_c} \vee s_3) \wedge \\ & (\bar{s_1})^3 \quad \wedge (\bar{s_2})^1 \quad \wedge (\bar{s_3})^1.\end{aligned}$$

The literal-centric definition of MaxSAT is closely related to how *integer programming* (IP) [17, 18] is standardly syntactically defined. An instance of integer programming consists of a linear objective function $c_1x_1 + \ldots + c_nx_n$ and a collection of linear constraints $a_1x_1 + \ldots + a_nx_n \geq a_0$ where $a_i, c_i \in \mathbb{Z}$ are constants. The integer programming problem asks to find values $x_i \in \mathbb{Z}$ so that the objective function of an instance is minimized (or maximized).

Any instance $\mathcal{F}^L$ can be then transformed into an instance of integer programming. The objective function of the corresponding IP-instance is $\sum_{x \in \mathcal{S}(\mathcal{F}^L)} x \cdot w(x)$. For each clause C we add the constraint $x_1 + \ldots + x_n + (1 - y_1) + \ldots + (1 - y_m) \geq 1$ where x_i occur positively (i.e. are not negated) and y_i negatively in C. Finally we add the constraint $x_i \in \{0, 1\}$ for each variable $x_i \in \textsc{Var}(\mathcal{F}^L)$.

2.3 MaxSAT Algorithms

Many practical problems have polynomial-size MaxSAT encodings [22, 23, 36, 37] and so an efficient implementation of a MaxSAT algorithm serves as a general and efficient optimization tool. As such, MaxSAT has gained popularity and it has been studied extensively leading to many different MaxSAT algorithms and their implementations called *solvers* [13–15, 20, 28, 38–40].

MaxSAT solvers are either *complete* or *incomplete* [8, 29]. Complete MaxSAT solvers compute a provably optimal solution for the input instances while incomplete algorithms do not prove the optimality of solutions, focusing instead in finding good quality solutions quickly. The upside to using a complete algorithm is the guaranteed optimality of the found solutions. The downside is that complete algorithms may spend a lot of time in proving optimality.

In some domains finding good quality solutions are enough but solutions should be fast to compute. Incomplete solvers focus on providing good quality solutions in short periods of time, not spending time arguing about optimality.

Approaches used by complete solvers include the *branch and bound* [41], *core-guided* [42–47], *implicit hitting set* [15, 20, 48–50] and *linear search* [42, 51, 52] out of which all but the first one remain competitive on industrial instances[1]. Complete solvers computing intermediate solutions can be modified to store the best intermediate solution encountered during search which can then be returned in case the search is terminated before managing to prove the optimality. As such, these solvers can be also used as incomplete solvers.

Branch and bound excluded, the mentioned approaches rely on successive calls to a SAT solver. Core-guided algorithms use a SAT solver to identify unsatisfiable collections of the soft clauses called *cores* and modify the original instance in order to rule out each core. In the implicit hitting set approach a SAT solver is used to accumulate a collection of cores $\mathcal{K}$ for which a minimum cost hitting set HS is computed. If the instance restricted to clauses outside of HS is satisfiable by an assignment τ, the assignment τ is returned as the optimal solution [15]. Linear search algorithms operate by iteratively calling a SAT solver on the original instance with an added a CNF-encoded constraint enforcing solutions to have a cost lower than the currently best known solution.

Most complete solvers operate on literal-centric instances in practice, by converting the given clause-centric instances into literal-centric instances in the manner described in Section 2.2.2 [15, 53]. Using soft variables, in each call to a SAT solver MaxSAT solvers can ask the SAT solver to try to find an assignment that satisfies a soft clause C by including the requirement that s_C is set to false in the call which is less time consuming than modifying the instance given to the SAT solver between calls [54].

Today, the main incomplete approaches are *stochastic local search* [14, 28, 38, 39] and the incomplete variants of the already mentioned complete approaches [13]. Stochastic local search algorithms for MaxSAT work by first generating an initial

[1]No branch and bound solver was submitted to the 2017-2019 MaxSAT evaluations (which do not include random instances). See `https://maxsat-evaluations.github.io/`.

assignment and iteratively choosing a variable according to some heuristics and reversing its assignment [8, 14, 28, 38–40]. Stochastic local search is discussed in-depth in Chapter 3 as we will describe a new local search algorithm later in this thesis. Other incomplete methods often utilize techniques used by complete MaxSAT solvers, such as the core-guided method and linear search [13]. In addition, general approximation strategies such as dividing the weights of the soft clauses by a constant exist [55].

3 Stochastic Local Search in SAT and MaxSAT

As explained in [8], stochastic local search (SLS) is a general method for searching for solutions to instances of a given problem. SLS algorithms begin by generating an initial *configuration* (or a *candidate solution*). The definition of a configuration is problem-specific: for example, in SAT a configuration refers to an assignment and in the traveling salesperson problem to a route visiting all required points and returning back to the starting point. The search then proceeds from configuration to configuration, where the choice of the next configuration is based on "local" information, such as the quality of the configurations in the current configuration's *neighborhood*. The neighborhood of a configuration consists of the configurations that are in some sense "close" to the current configuration, as defined by the algorithm. The crux of the design of SLS algorithms is to develop good and efficient heuristics for choosing the next configuration from the current one's neighborhood. Both initialization and the process of choosing the next configuration may involve randomness; hence the word "stochastic" in SLS.

SLS has been developed and successfully applied in solving instances from a number of different combinatorial problems like the *graph coloring* problem [8, 56], the traveling salesperson problem [8, 16], a variety of scheduling problems [8, 57, 58], as well as SAT [8, 29, 59–73] and MaxSAT [8, 14, 28, 38–40, 67]. SLS algorithms are incomplete and thus do not prove the non-existence of solutions for decision problems nor the (non-)optimality of found solutions.

SLS algorithms for SAT and MaxSAT show complementary behavior to complete algorithms [8, 14, 60, 74]. Hybrid algorithms combining a complete algorithm and an SLS algorithm have performed well in the SAT Competitions[1] and MaxSAT evaluations[2]. Notable examples of hybrid solvers include *ReasonLS* [75] and *Sparrow2Riss* [75] which won the 2018 SAT Competition's no-limits and random track respectively. In the MaxSAT world, SATLike-c [76] won both unweighted tracks in the 2018 MaxSAT Evaluation (MSE)[3] and placed second and third in the 2019 MSE[4] in the 60s and 300s tracks respectively. The Open-WBO-Inc-BMO-SATLike [77] solver placed third on both of the weighted tracks in the 2019 MSE.

Many of the ideas employed in SLS algorithms for MaxSAT come from the SLS algorithms for SAT and thus in the first subsection we overview SLS strategies for SAT. The main difference in SLS algorithms for MaxSAT compared to SAT is due to having to take account weights and having to satisfy all the hard clauses in weighted and partial MaxSAT respectively. Moreover, algorithms for MaxSAT store intermediate solutions as unlike in SAT, the solutions are not necessarily of the same quality and so found solutions ought to be improved.

We then move onto MaxSAT and discuss a state-of-the-art SLS algorithm for MaxSAT called SATLike. In Section 6.1 we introduce a new SLS MaxSAT algorithm that searches over locally minimal solutions, building on the ideas in SATLike.

[1] See `http://satcompetition.org/`.

[2] See `https://maxsat-evaluations.github.io/`.

[3] See `https://maxsat-evaluations.github.io/2018/rankings.html`.

[4] See `https://maxsat-evaluations.github.io/2019/rankings.html`.

3.1 Stochastic Local Search for Satisfiability

Stochastic local search algorithms for SAT work by generating an initial assignment τ which is then refined by successively *flipping* a variable x i.e. setting $\tau(x)$ to $1-\tau(x)$, where x is a heuristically chosen variable returned by a $pickVar$ function [29]. The neighborhood in SLS SAT algorithms is thus a subset of the *1-exchange neighborhood* [8] which contains the assignments obtainable from the current assignment with one flip.

Neighboring assignments are often given a *score* which can be used in guiding the search. The score often depends on one or two of the values *make* and *break* where for a variable x, $make(x)$ denotes the number of unsatisfied clauses that would be satisfied after flipping x. Similarly, $break(x)$ is the number of satisfied clauses that would become unsatisfied after flipping x. The value $make(x) - break(x)$ then equals the change in the number of satisfied clauses if x is flipped. Usually, $score(x)$ is defined as the difference of $make(x)$ and $break(x)$. In this section, if not explicitly mentioned, we tacitly assume that the definition of *score*, *make* and *break* are as defined here.

Definition 3.1. *Let $\mathcal{F}$ be a CNF formula, τ an assignment, $x \in \text{VAR}(\mathcal{F})$,*

$$A = \{C \in \mathcal{F} \mid x \in \text{VAR}(C),\ \tau(C) = 0\}$$

and

$$B = \{C \in \mathcal{F} \mid x \in \text{VAR}(C), \tau(C) = 1, \tau(C \setminus \{x, \neg x\}) = 0\}.$$

Then $make(x) \coloneqq |A|$, $break(x) \coloneqq |B|$ and $score(x) \coloneqq make(x) - break(x)$.

Moving from a configuration to a neighboring configuration with the highest *score* is called a *greedy* step. Greedy steps alone seldom suffice as they often get stuck in *local minimums* which is a configuration where no improving flip can be made according to the *score* function. Since local minimums are not necessarily solutions, heuristics which may direct the search away from local minimums are also needed. Steps going against the score function in order to avoid stagnation are called *diversifying steps*. An example of such a step is flipping a variable at random.

We discuss three classical categories for SLS SAT algorithms: *GSAT*-like algorithms [70], *WalkSAT*-like algorithms [72] and *dynamic local search* algorithms [8]. Most SLS algorithms for SAT can be divided into these categories [61].

3.1.1 GSAT

One of the first SLS algorithms for SAT is the GSAT algorithm from the early 1990s [8, 70, 71]. The algorithm takes on two user defined parameters mr (maximum number of rounds) and mf (maximum number of flips) in addition to the input instance. In each round, first an initial assignment is chosen uniformly at random from the set of all possible assignments and is then refined until a satisfying assignment is found or until mf flips have been performed. In the latter case a new

round is started. The working assignment is refined by choosing one variable with the best (highest) *score* uniformly at random and flipping it.[1]

Example 3.2. Let $\mathcal{F}$ be the CNF formula

$$(x \vee y) \wedge (\bar{x} \vee z) \wedge (\bar{y} \vee z) \wedge (y \vee \bar{z}) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z).$$

Suppose first that the initial assignment chosen by GSAT is $\tau = \{\bar{x}, \bar{y}, \bar{z}\}$. The assignment satisfies all but the first clause. At this point,

$$score(x) = 0, \quad score(y) = -1, \quad score(z) = -1$$

whereupon x is flipped to obtain $\tau = \{x, \bar{y}, \bar{z}\}$. However, this doesn't affect the scores i.e.

$$score(x) = 0, \quad score(y) = -1, \quad score(z) = -1$$

still, so x is flipped again leading us back to $\tau = \{\bar{x}, \bar{y}, \bar{z}\}$. Therefore GSAT is stuck alternating between the local minimums $\{\bar{x}, \bar{y}, \bar{z}\}$ and $\{x, \bar{y}, \bar{z}\}$.

Suppose then that the generated initial assignment is $\tau = \{\bar{x}, \bar{y}, z\}$. Now

$$score(x) = 0, \quad score(y) = 2, \quad score(z) = -1$$

so y is flipped, leading to the satisfying assignment $\tau = \{\bar{x}, y, z\}$.

The most time consuming part in GSAT is identifying variables x with the best score. A naive implementation of GSAT recomputes the scores of all variables which requires $\mathcal{O}(n^2)$ steps each time where n is the number of variables. A more efficient way is to use *caching* [8, 29]. After flipping x, only the scores of its *neighboring* variables need to be updated. A variable y is a neighbor of x if $x, y \in \textsc{Var}(C)$ for some clause C. Without going into details, we note that this can be further improved by exploiting the fact that only clauses whose number of satisfying literals either (i) dropped from two or one or (ii) increased from zero or one affect the scores.

GSAT often gets stuck in local minimal due to the greedy nature of the way it chooses the next variable to flip [8]. Restarting after every mf flips helps to mitigate this issue. Other diversification methods have been implemented into the original GSAT algorithm which have led to algorithms such as *GWSAT* [8], *GSAT/Tabu* [8], *HSAT* [8, 78] and *HWSAT* [8, 79].

3.1.2 WalkSAT

Another early SLS algorithm for SAT is *WalkSAT* [72]. The neighborhood relation in WalkSAT is defined by the variables in unsatisfied clauses. Instead of choosing from the set of all variables as in variants of GSAT, WalkSAT first chooses an unsatisfied clause C uniformly at random as in [80] and then selects a variable from C. The variable is chosen randomly from the variables in the clause with zero *break* which are the variables that do not falsify clauses when flipped. If no variable x

[1] In the original paper [70], the *score* of a variable x is defined as the number of unsatisfied clauses resulting from flipping x, but this difference has no effect in choosing variables.

with $break(x) = 0$ exist in the clause, with probability p a random variable in the clause is chosen and otherwise (with probability $1-p$) the variable with the smallest break value is selected (breaking ties randomly) [8, 29].

The *focused* search in WalkSAT where an unsatisfied clause is picked first serves as a basis for multiple different algorithms. Algorithms utilizing focused search include *Novelty* [68], *gNovelty*$^+$ [69], *Sparrow* [29, 59], and *probSAT* [29].

The probSAT algorithm performed much the better than Sparrow—created by replacing the Novelty$^+$ heuristic in gNovelty$^+$ with another one [29]—on the instances of the random track of the 2012 SAT Challenge [29]. Interestingly, probSAT is also arguably the most simple out of the four algorithms, being almost identical to the original WalkSAT algorithm. The only difference between the WalkSAT and probSAT lies in the way they choose the variable from the selected unsatisfied clause. In probSAT, the variable is chosen according to a probability distribution where the probability of the variables to be chosen depends on both *make* and *break* whereas WalkSAT uses the uniform distribution in its probabilistic choices. The fact that probSAT outperforms WalkSAT significantly illustrates well the importance of a suitably chosen variable selection method; a good neighborhood relation alone does not suffice.

3.1.3 Dynamic Local Search

Dynamic local search [8, 29] is a general approach in SLS algorithms where the definition of the underlying score function is modified after encountering local minimums [8]. In context of SAT this can mean for example imposing dynamic weights on clauses, or forbidding flipping a variable x (by imposing a large enough penalty to assignments with x flipped). Both techniques help diversify the search by steering the search away from recently visited assignments.

Clause weighting has been applied in multiple different ways in SLS algorithms for SAT [8, 29, 60–63, 65, 67, 69, 71, 73, 81, 82]. The general intuition is that after encountering a local minimum, the weights of unsatisfied clauses are increased to highlight the clauses that are often left unsatisfied in local minimums. The *make* and *break* now refer to the *sum of the weights* of new clauses that would become satisfied or unsatisfied respectively after flipping a variable. As such, the *score* of a variable x is now the increase in the weights of the satisfied clauses which would result from flipping x. After the weights of the clauses left unsatisfied in the local minimum are increased sufficiently, the scores of some variables become positive and assignments around the local minimum become available to greedy heuristics. Early algorithms implementing dynamic clause weighting include *GSAT with Clause Weights* [8, 71], *GLSSAT* [67], *DLM* [81], *ESG* [82], *SAPS* [65], and *PAWS* [73].

Let us look at how adding dynamic clause weighting to GSAT helps escape the local minimum in Example 3.2.

Example 3.3. Let $\mathcal{F}$ be the CNF formula

$$(x \vee y)^1 \wedge (\bar{x} \vee z)^1 \wedge (\bar{y} \vee z)^1 \wedge (y \vee \bar{z})^1 \wedge (\bar{x} \vee y \vee \bar{z})^1 \wedge (x \vee \bar{y} \vee z)^1.$$

from Example 3.2 where the superscripts denote the dynamic weight of the clauses (initialized to one). Consider again the instance $\tau = \{\bar{x}, \bar{y}, \bar{z}\}$ which leaves the first clause unsatisfied. Before moving onto the search phase, GSAT with Clause Weights increases the weight of each unsatisfied clause by one so the weight of the first clause $(x \vee y)$ increases from one to two leading to

$$(x \vee y)^2 \wedge (\bar{x} \vee z)^1 \wedge (\bar{y} \vee z)^1 \wedge (y \vee \bar{z})^1 \wedge (\bar{x} \vee y \vee \bar{z})^1 \wedge (x \vee \bar{y} \vee z)^1.$$

Now

$$score(x) = 1, \quad score(y) = 0, \quad score(z) = 0$$

so x is flipped in the current working assignment: $\tau = \{x, \bar{y}, \bar{z}\}$. The scores are then

$$score(x) = -1, \quad score(y) = -1, \quad score(z) = -1$$

so the algorithm picks one variable out of the three uniformly at random. Suppose z was flipped (so $\tau = \{x, \bar{y}, z\}$) after which

$$score(x) = -1, \quad score(y) = 2, \quad score(z) = 1.$$

Finally, y is flipped which leads to a satisfying assignment $\tau = \{x, y, z\}$ and the search is terminated.

More recent algorithms utilize *configuration checking* (CC) first introduced for solving the minimum vertex cover problem [83] which has been then adapted to SLS SAT algorithms [60–63]. Configuration checking is a method which disallows flipping variables whose *configuration* has not changed [60]; in this context the configuration of a variable is typically defined as a vector of the truth values of its neighboring variables [38, 60–63].

Algorithms combining configuration checking and clause weighting such as the hybrid algorithm CCASat [62] (which couples Swcca [61] and CCAsubscore [62]) and CCAnr [60] have performed well. CCASat won in the random track of SAT Challenge 2012 [62] outperforming other SLS algorithms such as probSAT. In addition, CCAnr was among the best SLS algorithms on structured SAT instances in the SAT Competitions of 2013 and 2014 (although in 2014 CCAnr participated as a hybrid algorithm combined with glucose) [60]. More recently, a hybrid solver *ReasonLS* [75] combining CCAnr and a modified version of Maple_LCM_Dist [84] placed first on the no-limits track of the 2018 competition.

3.2 Stochastic Local Search for Maximum Satisfiability

The SLS methods for SAT are almost directly applicable to MaxSAT as well. In MaxSAT the objective is to find low cost solutions i.e. all solutions are not of the same quality, the algorithms should store intermediate solutions and not immediately stagnate after finding a solution. Of course, algorithms for MaxSAT should take into account the distinction between hard and soft clauses as well as the weights given to soft clauses.

Early algorithms for MaxSAT include a WalkSAT variant *WalkSAT-JKS* [8, 85] for weighted partial MaxSAT and a dynamic local search algorithm *DLM-SW* [8, 86] based on the DLM algorithm for SAT. More recent algorithms include *Dist* [39, 40], *CCEHC* [28], *SATLike* [14] which are all dynamic local search algorithms, modifying at least the dynamic weights of the hard clauses. The Dist and CCEHC algorithms both employ two score functions, one considering only hard clauses and the other considering only soft clauses, while SATLike operates using a single score function.

The main goal of this section is to introduce the SLS algorithm SATLike [14] for MaxSAT which serves as the basis for a new SLS algorithm *LMS-SLS* introduced in Section 6.1. We first discuss a decimation based algorithm used as a subroutine by both SATLike and LMS-SLS. We then cover the SATLike algorithm which brought the performance of SLS algorithms on par with complete solvers in the incomplete track of the 2018 MSE[1]. At the time of writing the SATLike algorithm—built on ideas from SLS SAT algorithms—is a state-of-the-art SLS algorithm for MaxSAT. The key ideas in SATLike are a new *score* function and a new weighting mechanism similar to PAWS [73]. In contrast to CCEHC and Dist, the *score* function in SATLike takes into account both hard and soft clauses simultaneously. Moreover, the weighting mechanism works on both hard and soft clauses where as in CCEHC and Dist a dynamic weighting scheme is applied only on hard clauses.

SATLike participated in the 2018 and 2019 MSEs[1]. In the 2018 MSE, SATLike participated as a standalone algorithm and as a hybrid algorithm SATLike-c. The SATLike algorithm itself did not participate in the 2019 evaluation but SATLike-c and another hybrid algorithm Open-WBO-Inc-BMO-SATLike did. SATLike-c won the 60s and 300s unweighted incomplete tracks 2018 whereas SATLike was placed third and fourth respectively. The performance of SATLike and SATLike-c diminish on the weighted incomplete track however, both placed in the lower half in the rankings on the 60s and 300s subtracks. In the 2019 evaluation, SATLike-c placed 3rd and 2nd on the 60s and 300s tracks of the unweighted incomplete track respectively. On the weighted incomplete track SATLike-c placed 7th when the time limit was set to 60s and 5th with a 300s time limit while Open-WBO-Inc-BMO-SATLike placed 3rd with both time limits.

3.2.1 Decimation Based Initialization

Each SLS algorithm begins with an initial configuration. As many of the SLS algorithms utilize restarts—that is, they generate a new initial configuration and continue from scratch—it is natural to consider utilizing the information from the previous iteration of the local search procedure in generating new initial configurations.

We discuss a *unit propagation* based decimation procedure close to the strategy introduced in [74] which is used both in SATLike and in LMS-SLS. Unit propagation is a procedure to simplify the given instance by choosing a unit clause (l) and then removing all literals $\neg l$ and all clauses C with $l \in C$.

[1]See `https://maxsat-evaluations.github.io/2018` and `https://maxsat-evaluations.github.io/2019`.

The decimation procedure is used in conjunction with an SLS MaxSAT algorithm. After a search round of the underlying SLS MaxSAT algorithm, the decimation procedure is provided with the currently best found assignment τ^*. The procedure starts generating an arbitrary assignment τ and defining the status of each variable in $\mathcal{F}$ as "unassigned". In case there are no unit clauses, the decimation procedure then uses τ^* to add a unit clause (l) where l is picked from the literals for which $\tau^*(l) = 1$. Then, the decimation algorithm chooses one unit clause (l) out of them for unit propagation and sets $\tau = \tau_l$ and marks the variable corresponding to l as "assigned". After all variables in $\mathcal{F}$ have been assigned, τ is given to the SLS MaxSAT algorithm as an initial assignment.

As the SLS MaxSAT algorithm finds better solutions τ^*, the decimation procedure will (hopefully) yield better initial assignments which in turn helps the SLS MaxSAT algorithm in finding new improving solutions. In the beginning when search has not yet been performed or the SLS MaxSAT algorithm failed to find a solution, τ^* is selected uniformly at random among all assignments.

More specifically, the decimation procedure constructs an assignment by assigning one variable at each round. The variable and its value is chosen according to the following scheme:

1. if there are no unit clauses, select a variable x heuristically and assign it to $\tau^*(x)$,

2. if there are contradictory unit clauses (x) and $(\neg x)$, assign x to $\tau^*(x)$,

3. if there is a unit clause (x) or $(\neg x)$ but no contradictory unit clauses, assign x to true or false respectively.

After assigning a variable x, x is marked as "assigned", the clause (l) is added where $l = x$ if x was assigned to true and $l = \neg x$ otherwise. Next, unit propagation on (l) is performed.

The heuristic variable selection procedure chooses a variable using the *best from multiple selection* (BMS) strategy [87], which chooses k variables (where k is a user-defined parameter) with replacement from the set of candidate variables and then returns the variable that was assigned latest during the last decimation round. Using the BMS strategy and selecting the variable assigned latest helps to diversify the generation of an initial assignment. In the case the input instance is partial, the clause selection in 2 and 3 is restricted to unit hard clauses if there are any.[1]

3.2.2 SATLike

The score of a variable x in SATLike [14] is the increase in the sum of the dynamic weights of the satisfied clauses that would result from flipping x, as in many of the SLS algorithms for SAT mentioned earlier.

[1]We found that the BMS strategy was left out in the implementation of SATLike and is chosen uniformly at random instead.

Definition 3.4. *Let $\mathcal{F} = (F_h, F_s, w)$ be a clause-centric MaxSAT instance and τ an assignment, $x \in \textsc{Var}(\mathcal{F})$ and let $w_{dyn}(C)$ denote the current (dynamic) weight of a clause $C \in \mathcal{F}$. Denote*

$$A \coloneqq \{C \in \mathcal{F} \mid x \in \textsc{Var}(C),\ \tau(C) = 0\}$$

and

$$B \coloneqq \{C \in \mathcal{F} \mid x \in \textsc{Var}(C),\ \tau(C) = 1,\ \tau(C \setminus \{x, \neg x\}) = 0\}.$$

Let

$$make(x) \coloneqq \sum_{C \in A} w_{dyn}(C), \quad break(x) \coloneqq \sum_{C \in B} w_{dyn}(C)$$

and

$$score(x) \coloneqq make(x) - break(x).$$

The *score* function in SATLike does not take into account the distinction between hard and soft clauses. In SATLike the emphasis on hard clauses comes instead from a new dynamic weighting scheme which increases the weights of unsatisfied hard clauses more than the weights of unsatisfied soft clauses. SATLike has a user-defined parameter $h\_inc \geq 1$ which is used as the increment for the weights of hard clauses. The weights of soft clauses are increased by one and so, informally speaking, choosing $h\_inc > 1$ will help in highlighting the hard clauses.

In SATLike, a limit ζ for the weights of soft clauses is enforced, where ζ is a user defined parameter. Without the limit, the weights of soft clauses can grow arbitrarily large and thus dominate the scores of (some) variables. Consider for example the case where all soft clauses have a weight of one thousand (1000) except for one clause C which has a weight of one. If the weight of C can grow indefinitely, it may overshadow the hard clauses and the other soft clauses that are initially considered more important.

SATLike is a two-phased algorithm, consisting of a greedy component and a random walk component. If there are variables with a positive score, the greedy component is used. Otherwise the algorithm increases the weights of the unsatisfied clauses and performs a random walk step.

Algorithm 1 contains the pseudocode of SATLike. The initialization is done on lines 1–2 before moving into the search loop on line 3. If the input instance is partial but not weighted, the initial assignment is generated by the decimation procedure described earlier. According to the authors of [14] the decimation procedure is not useful for weighted and partial MaxSAT instances and the initial assignment is generated randomly.

Each iteration of the search loop begins with an update procedure where the cost of the current incumbent solution τ is evaluated (lines 4–5). If the cost of τ is less than the cost of the currently best known solution τ^*, then τ^* is set to τ on line 5.

In the greedy component on lines 6–7 a variable from the set D of variables with a positive score is selected. The variable is chosen according to the BMS strategy which works in SATLike by first sampling t variables from D with replacement and then selecting the variable with the greatest score from the sample where ties are

Algorithm 1: SATLike

Input : A clause-centric MaxSAT instance $\mathcal{F}$
Output: A solution τ along with its cost if a solution is found, otherwise "no solution found".

```
1  τ := an initial assignment
2  τ* := τ
3  while no termination criteria is met do
4  |  if τ is a solution and Cost(F, τ) < Cost(F, τ*) then
5  |  |  τ* := τ
6  |  if D := {v ∈ Var(F) | score(v) > 0} ≠ ∅ then
7  |  |  v := a variable chosen by the BMS strategy
8  |  else
9  |  |  update clause weights
10 |  |  if ∃ an unsatisfied hard clause C then
11 |  |  |  C := a random unsatisfied hard clause
12 |  |  else
13 |  |  |  C := a random unsatisfied soft clause
14 |  |  v := a variable in C with the highest score, breaking ties by age
15 |  τ := τ with v flipped
16 if τ* is a solution then
17 |  return REC(τ*), Cost(F, REC(τ*))
18 else return "no solution found"
```

broken by *age*, the time when the variable was last flipped, preferring the oldest variable.

The random walk component on lines 8–14 first updates the weights of unsatisfied clauses, increasing the weights of the unsatisfied hard clauses by h_inc and the weights of each unsatisfied soft clause C with $w(C) < \zeta$ by one, with probability $1 - sp$. Otherwise (with probability sp) the weights of the clauses are *smoothed*: for each satisfied hard or soft clause C with $w(C) > 1$, $w(C)$ is decreased by h_inc or by one respectively[1]. After updating the weights, SATLike selects an unsatisfied clause C and flips the variable with the highest score in C. The clause C is chosen uniformly at random from the set of unsatisfied clauses if there are any, and otherwise an unsatisfied soft clause at random.

After the search is terminated, the algorithm returns the best found solution or reports that no solutions were found (lines 15–18).

[1]The authors of [14] do not explicitly mention whether or not weights can become negative. The implementation of SATLike available at `http://lcs.ios.ac.cn/~caisw/MaxSAT.html` suggests that the weights may indeed become negative.

4 MaxSAT Preprocessing

Preprocessing in SAT and MaxSAT refers to the process of reformulating a given instance prior to solving, the aim being that the reformulated instance is easier for solvers to operate on. In particular, the aim in applying preprocessing is that it takes less time to preprocess and run the solver on the preprocessed instance than running the solver directly on the instance.

A numerous amount of different preprocessing techniques for SAT have been developed [88–109]. Recently, the study of preprocessing techniques in SAT has matured to the point where a general framework covering numerous preprocessing techniques for SAT has been developed [110]. The framework allows learning and removing clauses satisfying certain general conditions providing a unified approach to analyzing the correctness of preprocessing techniques for SAT; proving the correctness of most preprocessing techniques $\mathcal{P}$ for SAT most often boils down to showing $\mathcal{P}$ can be expressed as a combination of clause learning and removal steps. The framework also provides a linear time (in terms of the sum of the sizes of the removed clauses) reconstruction algorithm to produce a satisfying assignment for the original formula based on a satisfying assignment for the preprocessed formula.

Devising preprocessing techniques for MaxSAT poses additional difficulties compared to SAT, as the weights of the clauses have to be taken into account. Especially, optimal solutions to the preprocessed instance should correspond to optimal solutions to the original instance.

Due to these additional requirements, most SAT-based preprocessing techniques cannot be directly applied to clause-centric MaxSAT instances [30]. However, if clause-centric instances are first converted into literal-centric instances, a wide range of preprocessing techniques for SAT become available in the following sense: (i) A solution to the original clause-centric instance can be reconstructed in polynomial time from a solution to the preprocessed instance and (ii) for each optimal solution with an apparent cost c to the preprocessed instance, the corresponding solution to the original instance obtained via solution reconstruction is also optimal and has cost c [30, 53]. In particular, the SAT-based preprocessing techniques *blocked clause elimination*[1] (BCE) [89, 94], *bounded variable elimination* (BVE) [91–93], *subsumption elimination* (SE) [90] and *self-subsuming resolution* (SSR) [91] discussed in detail in Section 4.1 can be lifted to literal-centric MaxSAT instances with minor restrictions. On top of the SAT-based preprocessing techniques, several MaxSAT-specific techniques have been introduced [88, 109, 111]. Out of the MaxSAT-specific techniques we will discuss *Group-subsumed label elimination* [88].

The work presented in [53] provides a unified approach to analyzing the correctness of MaxSAT preprocessing techniques, including most SAT-based techniques. Combined with the earlier results in [110], the same linear time reconstruction procedure for most of the SAT-based techniques (including BCE, BVE, SE and SSR) can be used as is in MaxSAT preprocessing i.e. the aforementioned conditions (i) and (ii) are satisfied.

[1] BCE is also directly applicable to clause-centric MaxSAT [30].

Preprocessors—programs applying preprocessing techniques—such as Coprocessor [112, 113], MaxPre [88] and SatELite [91] implement many of the preprocessing techniques for SAT and MaxSAT. Any SAT or MaxSAT solver can be prepended with a preprocessor by feeding the given input instance first to a preprocessor which then returns a corresponding preprocessed instance for the solver to operate on. If the solver returns a solution, the preprocessor uses it to reconstruct a solution for the original instance. Both Coprocessor and MaxPre can also be used to preprocess MaxSAT instances [15, 88].

Preprocessing is a successful method in increasing the performance of SAT solvers. For example, the winner of the random track of the 2018 SAT competition[1], SparrowToRiss [75], uses the CP3 extension of Coprocessor 2.0 while the solver Plingeling [75]—which placed first on the parallel SAT track and second on the parallel UNSAT track—uses several preprocessing such as blocked clause elimination and bounded variable elimination.

Compared to SAT, the effects of MaxSAT preprocessing have been more modest [30, 111, 114, 115]. In the 2019 MaxSAT Evaluation[2] most of the solvers did not include preprocessing techniques. The Open-WBO-ms-pre variant of the Open-WBO solver [77, 116], included the MaxPre preprocessor, but its performance on the unweighted and weighted incomplete tracks was poor on both when compared with other other solvers. However, the performance of Open-WBO-ms-pre was slightly better than the performance of Open-WBO-ms [77] which does not integrate a preprocessor. Later in Chapter 6 we show that preprocessing increases the performance of two stochastic local search solvers for MaxSAT.

In this chapter we overview some of the main preprocessing techniques for MaxSAT. We then show how a solution to the original instance can be recovered from a preprocessed instance.

4.1 Preprocessing Techniques

As mentioned earlier, numerous preprocessing techniques for SAT—and therefore also for MaxSAT—exist. In this work, we overview five main ones for MaxSAT, which are all implemented and used by default in the MaxPre preprocessor.

Two of the preprocessing techniques we will cover utilize the notion of a *resolvent*. Given clauses C and D with $l \in C$ and $\bar{l} \in D$ the resolvent of C and D with respect to a literal l is the clause $(C \cup D) \setminus \{l, \bar{l}\}$, denoted by $C \bowtie_l D$. Resolution is a rule of inference in propositional logic, where the resolvent $C \bowtie_l D$ is entailed by $C \wedge D$.

Resolution alone can be used to determine the satisfiability of a CNF formula by replacing two clauses C and D with their resolvent $C \bowtie_l D$ when $l \in C$, $\bar{l} \in D$, eventually leading to the empty formula or an empty clause. In the former case, the formula is satisfiable and unsatisfiable in the latter. However, applying this procedure, the size of the formula may grow exponentially large as is the case with the CNF encoding of the pigeonhole principle [117].

[1] See `http://satcompetition.org/`.

[2] See `https://maxsat-evaluations.github.io/2019/`.

Let $\mathcal{F}^L$ be a literal-centric instance. The techniques we cover are the following, out of which the first four are MaxSAT liftings of preprocessing techniques for SAT [30].

***Blocked clause elimination* (BCE)** removes a clause C *blocked by a literal* l if $l \in C \setminus \mathcal{S}(\mathcal{F}^L)$ such that the resolvent $C \bowtie_l D$ is a tautology for all $D \in \mathcal{F}^L$ with $\bar{l} \in D$. A clause is called *blocked* if it is blocked by a literal.

***Bounded variable elimination* (BVE)** is a resolution based technique to eliminate a variable $x \notin \mathcal{S}(\mathcal{F}^L)$. The instance obtained by applying BVE to eliminate the variable x is

$$\begin{aligned}&\left(\mathcal{F}^L \setminus \{C \in \mathcal{F}^L \mid x \in C \text{ or } \bar{x} \in C\}\right) \cup \\ &\{C \mid \exists A, B \in \mathcal{F}^L \text{ s.t. } C = A \bowtie_x B \text{ is not a tautology}\}.\end{aligned}$$

***Subsumption elimination* (SE)** removes a clause $C \in \mathcal{F}^L$ if $D \subseteq C$ for some $D \in \mathcal{F}^L$.

***Self-subsuming resolution* (SSR)** replaces a clause C with the clause $C \bowtie_l D$ if $C \bowtie_l D \subseteq C$ for some literal $l \notin \mathcal{S}(\mathcal{F}^L)$ with $l \in C$, $\bar{l} \in D$.

***Group-subsumed label elimination* (GSLE)** removes a soft variable s if there are soft variables $s_1, \ldots, s_n \neq s$ such that $s \in C$ implies $s_i \in C$ for some $i = 1, \ldots, n$ and $\sum_{i=1}^{n} w(s_i) \leq w(s)$.

The *bounded* part in BVE comes from an added restriction in practical applications. That is, the size of any given instance is not allowed to grow after applying BVE i.e. $|\text{BVE}(\mathcal{F}^L)| \leq |\mathcal{F}^L|$. We omit this restriction as it has no effect on the following theory.

For a literal-centric instance $\mathcal{F}^L$, we use $\mathcal{P}(\mathcal{F}^L)$ to denote any instance obtained from $\mathcal{F}^L$ with application(s) of a preprocessing technique $\mathcal{P}$. For a clause-centric instance $\mathcal{F}$, we use $\mathcal{P}(\mathcal{F})$ to denote $\mathcal{P}(\mathcal{F}^L)$ where $\mathcal{F}^L$ is the literal-centric instance corresponding to $\mathcal{F}$.

Example 4.1. Consider the literal-centric instance $\mathcal{F}^L = \{(x \vee \bar{y}), (\bar{x} \vee y), (\bar{x} \vee \bar{y})\}$ with no soft variables. Resolving $(x \vee \bar{y})$ with $(\bar{x} \vee \bar{y})$ on x gives $(\bar{y})$ which subsumes $(x \vee \bar{y})$ so applying SSR gives $\text{SSR}(\mathcal{F}^L) = \{(\bar{y}), (\bar{x} \vee y), (\bar{x} \vee \bar{y})\}$. We can further apply preprocessing to $\text{SSR}(\mathcal{F}^L)$. Notice that the clause $(\bar{y})$ is contained in the clause $(\bar{x} \vee \bar{y})$ so subsumption elimination yields $\text{SE}(\text{SSR}(\mathcal{F}^L)) = \{(\bar{y}), (\bar{x} \vee y)\}$.

Next, since $\text{SE}(\text{SSR}(\mathcal{F}^L))$ has no occurrences of x we may apply BCE to obtain $\text{BCE}(\text{SE}(\text{SSR}(\mathcal{F}^L))) = \{(\bar{y})\}$. Alternatively, using BVE to eliminate y gives $\text{BVE}(\text{SE}(\text{SSR}(\mathcal{F}^L))) = \{(\bar{x})\}$. Finally, applying BCE or BVE to either of these two instances results in the empty formula.

The MaxSAT-specific technique GSLE is redundant if no clause contains more than one soft variable. Thus GSLE is only of use after applying techniques such as BVE.

Example 4.2. Let

$$\mathcal{F}^L = (x) \wedge (x \vee \bar{y} \vee s_1) \wedge (\bar{x} \vee s_2) \wedge (\bar{x} \vee \bar{y} \vee s_3) \wedge (y \vee s_4)$$

where $w(s_i) = i$. Observe that GSLE is not applicable at this point. By eliminating the variable x we get the instance

$$\text{BVE}\left(\mathcal{F}^L\right) = (s_2) \wedge (\bar{y} \vee s_3) \wedge (\bar{y} \vee s_1 \vee s_2) \wedge (\bar{y} \vee s_1 \vee s_3) \wedge (y \vee s_4).$$

Applying BVE a second time to remove occurrences of y gives

$$\text{BVE}\left(\text{BVE}\left(\mathcal{F}^L\right)\right) = (s_2) \wedge (s_3 \vee s_4) \wedge (s_1 \vee s_2 \vee s_4) \wedge (s_1 \vee s_3 \vee s_4).$$

Since $w(s_2) + w(s_3) > w(s_4)$, s_4 is not subsumable by s_2 and s_3. However, for s_1 and s_3 we have $w(s_1) + w(s_3) \leq w(s_4)$ and each clause containing s_4 also contains s_1 or s_3. Thus s_4 is subsumed by s_1 and s_3 together giving

$$\text{GSLE}\left(\text{BVE}\left(\text{BVE}\left(\mathcal{F}^L\right)\right)\right) = (s_2) \wedge (s_3) \wedge (s_1 \vee s_2) \wedge (s_1 \vee s_3).$$

The soft variables s_2 and s_3 both appear in a unit clause so neither can be eliminated with GSLE. In addition, $w(s_1) < w(s_2), w(s_3)$. Thus GSLE cannot be applied again.

4.2 Solution Reconstruction

In applications of MaxSAT, the cost of an optimal solution alone is often not enough i.e. the solution itself needs to be returned. Therefore, if preprocessing is applied there should be a fast method of reconstructing a solution to the original instance from any solution to the preprocessed instance. There is a general reconstruction method for most of the SAT-based preprocessing techniques [53, 110], and a polynomial-time reconstruction algorithm for most MaxSAT-specific techniques [53]. We will not cover the general technique and instead show explicitly how solution reconstruction can be applied to the techniques defined earlier.

Let $\mathcal{F}$ be a clause-centric MaxSAT instance, $\mathcal{F}^L$ the corresponding literal-centric instance and let τ be a solution to $\mathcal{P}(\mathcal{F})$ where $\mathcal{P} \in \{\text{BCE}, \text{BVE}, \text{SE}, \text{SSR}, \text{GSLE}\}$. Let τ' denote the reconstructed assignment.

BCE: Let $C \vee l \in \mathcal{F}^L$ be the eliminated clause, blocked on $l \notin \mathcal{S}(\mathcal{F}^L)$. If $\tau(C) = 1$, then $\tau' = \tau$ and otherwise $\tau' = \tau_l$.

BVE: Let l be the eliminated literal. If there is a clause $A \vee l \in \mathcal{F}^L$ with $\tau(A) = 0$, then $\tau' = \tau_l$ and otherwise $\tau' = \tau_{\neg l}$.

SE: The reconstructed solution is $\tau' = \tau$.

SSR: The reconstructed solution is $\tau' = \tau$.

GSLE: Let $s \in \mathcal{S}(\mathcal{F})$ be the subsumed soft variable. Then $\tau' = \tau_{\neg s}$.

If multiple preprocessing steps are applied, as they usually are, the reconstruction steps are performed in reverse order of application. We use the notation $\text{REC}(\tau)$ to denote the assignment obtained after all reconstruction steps.

The *correctness* of the first four techniques follow from [53, 110] where by correctness we mean that (i) the instances $\mathcal{F}$ and $\mathcal{P}(\mathcal{F})$ viewed as CNF formulas are satisfiability-equivalent and (ii) for any optimal solution τ to $\mathcal{P}(\mathcal{F})$ with apparent cost c, the assignment $\mathrm{REC}(\tau)$ is an optimal solution for $\mathcal{F}$ with cost c.

To see that the reconstruction procedure for GSLE works correctly, note that by its definition if s is subsumed by $s_1, \ldots, s_n$ then any solution τ with $\tau(s) = 1$ can be converted to another solution τ' that assigns $s_1, \ldots, s_n$ to true instead of s where the cost of τ' is lower or equal to the cost of τ. That is, the assignment τ' to $\mathrm{GSLE}(\mathcal{F})$ with

$$\tau'(s_i) = 1, \quad \tau'(s) = 0, \quad \forall x \neq s_1, \ldots, s_n \;\; \tau'(x) = \tau(x)$$

is a solution to $\mathrm{GSLE}(\mathcal{F})$ and $\mathrm{REC}(\tau') = \tau'$ is a solution to $\mathcal{F}$ with cost at most equal to the cost of τ.

Theorem 4.3. *Each* $\mathcal{P} \in \{\mathrm{BCE}, \mathrm{BVE}, \mathrm{SE}, \mathrm{SSR}, \mathrm{GSLE}\}$ *works correctly.*

Example 4.4. Consider again the instance $\mathcal{F}^L = \{(x \vee \bar{y}), (\bar{x} \vee y), (\bar{x} \vee \bar{y})\}$, the preprocessed instance $\mathrm{BCE}\left(\mathrm{SE}\left(\mathrm{SSR}\left(\mathcal{F}^L\right)\right)\right) = \{(\bar{y})\}$ from Example 4.1 and the solution $\tau = \{\bar{y}\}$. Since the last applied reconstruction technique was BCE we first apply its reconstruction procedure to obtain a solution τ' to the instance $\mathrm{SE}\left(\mathrm{SSR}\left(\mathcal{F}^L\right)\right) = \{(\bar{y}), (\bar{x} \vee y)\}$; since we have $\tau(C) = 0$, where $C = (y)$ and $C \vee \bar{x} \in \mathrm{SE}\left(\mathrm{SSR}\left(\mathcal{F}^L\right)\right)$, by the reconstruction step for BCE we have $\tau' = \tau_{\neg x}$. The reconstruction steps for SSR and SE leave τ' unaltered so $\mathrm{REC}(\tau) = \tau' = \{\bar{x}, \bar{y}\}$.

Example 4.5. Let $\mathcal{F}^L$ be as in Example 4.2 so that

$$\mathrm{GSLE}\left(\mathrm{BVE}\left(\mathrm{BVE}\left(\mathcal{F}^L\right)\right)\right) = (s_2) \wedge (s_3) \wedge (s_1 \vee s_2) \wedge (s_1 \vee s_3).$$

Consider the solution $\tau = \{\bar{s_1}, s_2, s_3, s_4\}$. Since s_4 was subsumed, the reconstruction procedure for GSLE yields the solution $\tau' = \tau_{\neg s_4}$ to $\mathrm{BVE}\left(\mathrm{BVE}\left(\mathcal{F}^L\right)\right)$. The assignment τ' does not satisfy $C = (s_4)$, where $C \vee y = (y \vee s_4) \in \mathrm{BVE}\left(\mathcal{F}^L\right)$ so the reconstruction procedure sets $\tau' = \tau'_y$. Finally, the assignment τ' does not satisfy the empty clause $C = ()$, where $C \vee x = (x) \in \mathcal{F}^L$, so $\mathrm{REC}(\tau) = \tau'_x = \{x, y, \bar{s_1}, s_2, s_3, \bar{s_4}\}$.

Let us then examine what would happen if we were to apply the defined preprocessing techniques to a clause-centric instance. The trouble comes from the weights associated with soft clauses. Neither BVE nor SSR can be applied correctly no matter how the weights of the resolvents are chosen. The BCE technique is the only one of the techniques we consider that works correctly for clause-centric instances [30].

Example 4.6. Let $\mathcal{F}_1 = (x)^2 \wedge (\bar{x})^1$, $\mathcal{F}_2 = (x)^1 \wedge (\bar{x})^2$. For both instances, an application of BVE leads to the instances $\mathcal{E}_1 = ()^{c_1}$ and $\mathcal{E}_2 = ()^{c_2}$ obtained from $\mathcal{F}_1$ and $\mathcal{F}_2$ respectively, where c_i is the weight of the resolvent $() = (x) \bowtie_x (\bar{x})$. Now, no matter the weights c_i, any assignment τ is an optimal solution to both $\mathcal{E}_1$ and $\mathcal{E}_2$. However, $\mathrm{REC}(\tau) \in \{\tau_x, \tau_{\neg x}\}$ cannot be optimal for both $\mathcal{F}_1$ and $\mathcal{F}_2$ simultaneously.

A similar argument can be made for SSR as well. Applying SSR on $\mathcal{F}_1$ and $\mathcal{F}_2$ to subsume the clause (x) leads to the instances $\mathcal{E}_1 = ()^{c_1} \wedge (\bar{x})^1$ and $\mathcal{E}_2 = ()^{c_2} \wedge (\bar{x})^2$.

Now $\tau = \{\bar{x}\}$ is optimal to both $\mathcal{E}_1$ and $\mathcal{E}_2$ but $\mathrm{REC}(\tau) = \tau$ is not an optimal solution to $\mathcal{F}_1$.

Finally, applying SE to subsume $(x \vee y)$ from $\mathcal{F} = (\bar{y}) \wedge (\bar{x})^2 \wedge (x)^1 \wedge (x \vee y)^2$ yields $\mathcal{E} = (\bar{y}) \wedge (\bar{x})^2 \wedge (x)^1$ for which $\tau = \{\bar{x}, \bar{y}\}$ is optimal with cost 2 but $\mathrm{REC}(\tau) = \tau$ is not optimal for $\mathcal{F}$.

5 Locally Minimal Solutions

Most MaxSAT solvers, both complete and incomplete, make use of non-optimal solutions found during search. For example, stochastic local search algorithms for MaxSAT (which are incomplete) use the costs of non-optimal solutions as heuristic guidance to visit solutions with lower cost [14, 28, 38–40] while in linear search algorithms for MaxSAT the cost of encountered solutions is used as an additional constraint to enforce that successive solutions have a lower cost than the solutions found earlier during search [51, 118, 119].

MaxSAT solvers are usually oblivious to whether or not the input instance is preprocessed [13, 14, 28, 38–40, 49] as MaxSAT solver are rarely designed to rely on preprocessing. These solvers view the soft variables as unit soft clauses. Therefore, most MaxSAT solvers equate the apparent cost of a solution τ to a preprocessed instance with the cost of the solution $\text{REC}(\tau)$ to the original instance. For this reason it is essential for solvers utilizing preprocessing that for two solutions τ_1, τ_2 to a preprocessed instance $\mathcal{P}(\mathcal{F})$ with $\textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau_1) \leq \textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau_2)$ we have $\textsc{Cost}(\mathcal{F}, \text{REC}(\tau_1)) \leq \textsc{Cost}(\mathcal{P}(\mathcal{F}), \text{REC}(\tau_2))$. Otherwise the search may be misguided as solvers could mistakenly consider τ_1 to be the better solution due to its better apparent cost. Preferably, the stronger claim that the apparent cost of a solution τ to $\mathcal{P}(\mathcal{F})$ equals the cost of $\text{REC}(\tau)$ to $\mathcal{F}$ holds.

Thus far, establishing the correctness of different MaxSAT preprocessing techniques has so far been based on proving the preservation of optimal solutions. Given a preprocessing technique $\mathcal{P} \in \{\text{BCE}, \text{BVE}, \text{SE}, \text{SSR}, \text{GSLE}\}$ and an optimal solution τ to $\mathcal{P}(\mathcal{F})$, the reconstructed solution $\text{REC}(\tau)$ is optimal for $\mathcal{F}$ and the cost of $\text{REC}(\tau)$ equals apparent cost of τ [30, 53, 109]. However, these arguments rely on assuming the optimality of τ. Therefore the proofs do not necessarily generalize to arbitrary solutions.

We will show in Section 5.1 that these proofs indeed do not generalize to arbitrary solutions and solvers may *misinterpret costs*, i.e. solvers mistakenly consider the apparent cost of a solution to be the cost of the solution $\text{REC}(\tau)$ even though the two differ. Moreover, we show that there are solutions τ_1, τ_2 where τ_1 has a lower apparent cost than τ_2 but nevertheless $\text{REC}(\tau_1)$ has a higher cost than $\text{REC}(\tau_2)$. To the best of our knowledge, this issue had gone unnoticed since the adaptation of many effective SAT-based preprocessing techniques to MaxSAT in [30]. We hypothesize the issue of misinterpreting costs explains at least partially the more modest impact of preprocessing in MaxSAT solving as compared to preprocessing in context of SAT solving.

Towards rectifying the issue of misinterpreting costs, in Section 5.2 we introduce *locally minimal solutions* and prove that for most of the preprocessing techniques discussed the costs of locally minimal solutions are interpreted correctly. Each solution τ for a preprocessed instance $\mathcal{P}(\mathcal{F})$ can be turned into a locally minimal solution efficiently so most MaxSAT solvers can be modified with little impact on performance to store only locally minimal solutions for which the apparent cost is equal to the cost of the reconstructed solution. In Section 6.1 we propose an SLS solver for MaxSAT that only searches over locally minimal solutions.

5.1 Issues in Applications of MaxSAT Preprocessing

We begin this section by defining the cost of a solution τ for a preprocessed (literal-centric) instance $\mathcal{P}(\mathcal{F})$.

Definition 5.1. *Let $\mathcal{F}$ be a clause-centric MaxSAT instance, $\mathcal{P}(\mathcal{F})$ a preprocessed instance and τ a solution to $\mathcal{P}(\mathcal{F})$. The* cost *of the solution τ is* $\textsc{Cost}(\text{REC}(\tau), \mathcal{F})$.

In applications of preprocessing, we would expect the following claim to hold.

Claim 5.2. *Let $\mathcal{F}$ be a clause-centric instance and let $\mathcal{P}$ be a preprocessing technique. Then* $\textsc{Cost}(\mathcal{F}, \text{REC}(\tau)) = \textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau)$ *for any solution τ to $\mathcal{P}(\mathcal{F})$.*

The following example demonstrates that this claim does not hold in general.

Example 5.3. Let $\mathcal{F} = (\bar{x} \vee y)^1 \wedge (x \vee \bar{y}) \wedge (\bar{x} \vee \bar{y})$ be a clause-centric MaxSAT instance. The corresponding literal-centric instance is $\mathcal{F}^L = (\bar{x} \vee y \vee s) \wedge (x \vee \bar{y}) \wedge (\bar{x} \vee \bar{y})$ where $w(s) = 1$. The clause $(x \vee \bar{y})$ is blocked on $\bar{y}$ so applying BCE once we obtain the instance $\text{BCE}(\mathcal{F}) = (\bar{x} \vee y \vee s) \wedge (\bar{x} \vee \bar{y})$ which has a solution $\tau = \{s, \bar{x}, \bar{y}\}$. Since $\tau((\bar{x})) = 1$, where $(\bar{x} \vee \bar{y}) \in \mathcal{F}^L$, we have $\text{REC}(\tau) = \tau$ but

$$\textsc{Cost}(\mathcal{F}, \text{REC}(\tau)) = 0 \neq 1 = \textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau).$$

This implies that the apparent cost of a solution cannot in general be equated with its cost. Moreover, if a solver uses apparent cost in comparing the quality of solutions, it is not guaranteed that a solution τ_1 with a lower apparent cost than another solution τ_2 would be of better quality post reconstruction. However, even though apparent cost can differ from cost, it could still be the case that the order of solutions in terms of apparent cost is the same as if ordered in terms of cost instead.

Claim 5.4. *Let $\mathcal{P}$ be a preprocessing technique. Given two solutions τ_1 and τ_2 to a preprocessed instance $\mathcal{P}(\mathcal{F})$ with* $\textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}),\tau_1) \leq \textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}),\tau_2)$ *we have* $\textsc{Cost}(\mathcal{F}, \text{REC}(\tau_1)) \leq \textsc{Cost}(\mathcal{F}, \text{REC}(\tau_2))$.

In case Claim 5.4 holds, a solver that returns the solution with the best apparent cost still returns the best solution in terms of cost even though the cost is not necessarily equal to the apparent cost. The cost can be evaluated manually in linear time after search which adds little overhead as this has to be done only once. As it turns out, Claim 5.4 is also false.

Example 5.5. Let $\mathcal{F} = (x \vee y)^1 \wedge (\bar{x})^2$. Now $\mathcal{F}^L = (x \vee y \vee s_1) \wedge (\bar{x} \vee s_2)$ where $w(s_1) = 1$ and $w(s_2) = 2$. Applying BVE to remove the variable x gives $\text{BVE}(\mathcal{F}) = (y \vee s_1 \vee s_2)$. Let $\tau = \{\bar{y}, s_1, \bar{s_2}\}$ and $\tau' = \{y, \bar{s_1}, s_2\}$ be solutions to $\text{BVE}(\mathcal{F})$, for which

$$\textsc{Appar-Cost}(\text{BVE}(\mathcal{F}), \tau) = 1, \quad \textsc{Appar-Cost}(\text{BVE}(\mathcal{F}), \tau') = 2.$$

Since $C = (y \vee s_1)$, where $C \vee x \in \mathcal{F}^L$ is the only clause in $\mathcal{F}^L$ containing x, is satisfied by both τ and τ' we have $\text{REC}(\tau) = \tau_{\neg x}$ and $\text{REC}(\tau') = \tau'_{\neg x}$. However,

$$\textsc{Cost}(\mathcal{F}, \text{REC}(\tau)) = 1, \quad \textsc{Cost}(\mathcal{F}, \text{REC}(\tau')) = 0.$$

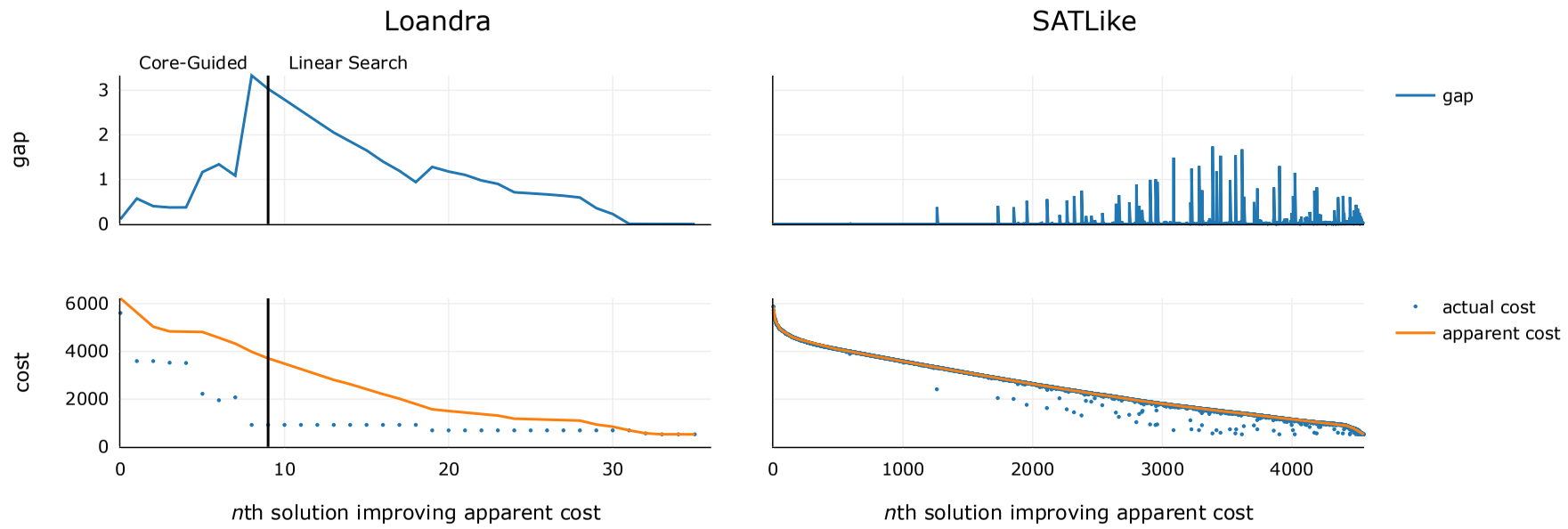


Figure 1: Difference between costs and apparent costs of solutions: Loandra's core-guided and linear-search phases (left) and SATLike (right) on the instance ram_k3_n12.ra1.wcnf.

This has adverse implications for MaxSAT algorithms utilizing preprocessing. The search process may be seriously misguided if apparent costs are used in steering the search. Figure 1 illustrates this effect occurring in the MaxSAT solvers Loandra[1] [13] and SATLike[2] [14]. Loandra combines core-guided and linear search techniques while SATLike is a stochastic local search solver.

Both Loandra and SATLike were augmented with the MaxPre [88] preprocessor so that the input instance $\mathcal{F}$ is first given to MaxPre which then returns a preprocessed instance $\mathcal{P}(\mathcal{F})$ to the solver. The solvers were then ran on the instance ram_k3_n12.ra1.wcnf from the 2019 MaxSAT Evaluation. For each solution τ improving the apparent cost, the value $\textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau)$ was recorded along with the cost $\textsc{Cost}(\mathcal{F}, \text{REC}(\tau))$. Moreover, the *gap* between the apparent costs and costs was computed. The gap of a solution τ is defined as

$$\frac{\textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau)}{\textsc{Cost}(\mathcal{F}, \text{REC}(\tau))} - 1.$$

Both solvers made drastic over-approximations during search. From Figure 1 we see that the gap was over three at worst both in Loandra's core-guided phase and in its linear search phase. That is, the apparent cost is at worst over three times larger than the cost. For SATLike the gap is nearly two at worst.

In the linear search phase, Loandra spent time improving the apparent cost with no improvement in the cost. For $\tau_{10}, \ldots, \tau_{20}$, where τ_n denotes the nth found solution, the apparent cost decreases but the cost remains constant at 919. Similarly for solutions $\tau_{21}, \ldots \tau_{33}$ apparent cost is improved but the cost stays at 689.

Worse still, some solutions recorded by both solvers improved the apparent cost but increased the cost. For example, the eight solution τ_8 found by Loandra in its core guided phase had

$$\textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau_8) = 4572, \quad \textsc{Cost}(\mathcal{F}, \text{REC}(\tau_8)) = 1953.$$

The ninth solution τ_9 improved the apparent cost but decreased the cost as

$$\textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau_9) = 4324, \quad \textsc{Cost}(\mathcal{F}, \text{REC}(\tau_9)) = 2073.$$

Likewise for SATLike

$$\textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau_{3089}) = 1739, \quad \textsc{Cost}(\mathcal{F}, \text{REC}(\tau_{3089})) = 698$$

but

$$\textsc{Appar-Cost}(\mathcal{P}(\mathcal{F}), \tau_{3090}) = 1738, \quad \textsc{Cost}(\mathcal{F}, \text{REC}(\tau_{3090})) = 1738.$$

For Loandra, the apparent costs do differ from the costs in the core-guided phase as well as the linear-search phase suggesting that this problem extends to other solvers (complete or incomplete) applying core-guided and linear-search methods. Likewise, when prepended with a preprocessor, misinterpretation of costs extends to implicit hitting approaches and other stochastic local search algorithms which make use the costs of found solutions [14, 15, 28, 38, 48, 49].

[1] Available at `https://github.com/jezberg/loandra` under the MIT license.

[2] Available at `http://lcs.ios.ac.cn/~caisw/MaxSAT.html` under the MIT license.

5.2 Locally Minimal Solutions

There are several ways to remedy the issue of misinterpretation of costs when preprocessing is applied. One way around the issue is to ignore apparent costs and instead apply solution reconstruction to evaluate the cost of the found solutions. However, even though the reconstruction procedure takes linear time it severely impacts the performance of solvers that encounter lots of solutions in short periods of time. This is particularly the case with SLS solvers as they often encounter thousands of solutions (or more) within seconds.

A more sophisticated way around the issue would be to only compute *minimal solutions* (defined in [30]) as for them apparent cost equals cost [30]. Unfortunately, to decide whether a solution is minimal is an NP-hard task [120] and so devising a practically efficient algorithm guaranteed to search over minimal solutions is not realistic, at least not if $\mathbf{P} \neq \mathbf{NP}$.

Our approach around misinterpreting costs is to restrict the set of considered solution candidates more suitably. To motivate our approach, let us return to Example 5.3. In that example, we had $\text{BCE}(\mathcal{F}) = (\bar{x} \vee y \vee s) \wedge (\bar{x} \vee \bar{y})$ which has the solution $\tau = \{s, \bar{x}, \bar{y}\}$ with apparent cost of one. Setting the soft variable s to true is not necessary though as $\tau_{\neg s}$ is also a solution to $\text{BCE}(\mathcal{F})$ with apparent cost of zero. The difference of τ and $\tau_{\neg s}$ is that $\tau_{\neg s}$ has no soft variables set to true unnecessarily. We call such solutions *locally minimal*.

Definition 5.6. *Let $\mathcal{F}^L$ be a literal-centric MaxSAT instance. A solution τ to $\mathcal{F}^L$ is a* locally minimal solution *if there is no $s \in \mathcal{S}(\mathcal{F}^L)$ with $\tau(s) = 1$ such that $\tau_{\neg s}$ is also a solution.*

Since locally minimal solutions τ to $\mathcal{F}^L$ have no soft variables set to true unnecessarily, i.e. if $\tau(s) = 1$ for $s \in \mathcal{S}(\mathcal{F}^L)$ then there is a clause $C \vee s$ with $\tau(C) = 0$, for $\mathcal{F}$ and its literal-centric counterpart $\mathcal{F}^L$ we have $\textsc{Cost}(\mathcal{F}, \tau) = \textsc{Appar-Cost}(\mathcal{F}^L, \tau)$ when τ is a locally minimal solution. Therefore, if a preprocessing technique *preserves* locally minimal solutions in the following sense, apparent costs equal costs.

Definition 5.7. *Let $\mathcal{F}^L$ be a literal-centric instance. A preprocessing technique $\mathcal{P}$ for MaxSAT preserves locally minimal solutions if for all locally minimal solutions τ to $\mathcal{P}(\mathcal{F})$ the solution $\text{REC}(\tau)$ is locally minimal to $\mathcal{F}^L$.*

Example 5.8. Let $\mathcal{F} = (x \vee y)^1 \wedge (\bar{x})^2$, $\mathcal{F}^L = (x \vee y \vee s_1) \wedge (\bar{x} \vee s_2)$ where $w(s_i) = i$, so that after eliminating x we have $\text{BVE}(\mathcal{F}) = (y \vee s_1 \vee s_2)$ as in Example 5.5. There are three locally minimal solutions to $\text{BVE}(\mathcal{F})$, namely

$$\tau_1 = \{y, \bar{s_1}, \bar{s_2}\}, \quad \tau_2 = \{\bar{y}, s_1, \bar{s_2}\}, \quad \tau_3 = \{\bar{y}, \bar{s_1}, s_2\},$$

with apparent costs 0, 1, 2 respectively. After reconstruction, we have

$$\text{REC}(\tau_1) = \{\bar{x}, y, \bar{s_1}, \bar{s_2}\}, \quad \text{REC}(\tau_2) = \{\bar{x}, \bar{y}, s_1, \bar{s_2}\}, \quad \text{REC}(\tau_3) = \{x, \bar{y}, \bar{s_1}, s_2\},$$

with costs 0, 1 and 2 respectively.

The question of which preprocessing techniques preserve locally minimal solutions then arises. We will show next that all the discussed preprocessing techniques except for BCE preserve locally minimal solutions.

Theorem 5.9. BVE, SE, SSR *and* GSLE *preserve locally minimal solutions.*

Proof. First, we note that for any locally minimal solution τ to a literal-centric $\mathcal{F}^L$ and $s \in \mathcal{S}(\mathcal{F}^L)$ with $\tau(s) = 1$ there is a clause C so that $s \in C \in \mathcal{F}^L$ and $\tau(C \setminus \{s\}) = 0$. Otherwise $\tau_{\neg s}$ is a solution to $\mathcal{F}^L$ contradicting the local minimality of τ.

Let then $\mathcal{F}^L$ be a literal-centric instance and let $\mathcal{P} \in \{\text{BVE}, \text{SE}, \text{SSR}, \text{GSLE}\}$. Furthermore, let τ be a locally minimal solution to $\mathcal{P}(\mathcal{F}^L)$. If $\mathcal{S}(\mathcal{P}(\mathcal{F}^L)) = \emptyset$ the claim follows as the reconstruction procedure does not set soft variables to true. Thus we may assume that $\mathcal{P}(\mathcal{F}^L)$ contains soft variables.

Let $s \in \mathcal{S}(\mathcal{P}(\mathcal{F}^L))$ be a soft variable with $\tau(s) = 1$ for which there is a clause C such that $C \vee s \in \mathcal{P}(\mathcal{F}^L)$ and $\tau(C) = 0$. We prove the claim by showing $\text{REC}(\tau)_{\neg s}$ is not a solution to $\mathcal{F}^L$. If $C \vee s \in \mathcal{F}^L$ the claim already follows as $\text{REC}(\tau)(C) = 0$. Hence we may also assume $C \vee s \notin \mathcal{F}^L$.

The rest of the proof boils down to a case by case analysis:

BVE: Let x be the non-soft variable removed by BVE. As $C \vee s \notin \mathcal{F}^L$ there are clauses D and E for which $C \vee s = D \bowtie_x E$. Since $\tau(C) = 0$, $\text{REC}(\tau)_{\neg s}$ satisfies D at most on x and E at most on $\neg x$. As we assume no tautologies, $\text{REC}(\tau)_{\neg s}$ cannot satisfy D and E simultaneously.

SE: The assumption $C \vee s \notin \mathcal{F}^L$ is not applicable.

SSR: Since $C \vee s \notin \mathcal{F}^L$, there are clauses $D, E \in \mathcal{F}^L$ and a non-soft variable x such that $C \vee s = D \bowtie_x E$ and $C \vee s$ subsumes D. Since $\tau(C) = 0$, s is the only reason $D \bowtie_x E$ is satisfied by τ. Therefore with $\text{REC}(\tau)_{\neg s}$, the clause D is satisfied at most on x and E at most on $\neg x$ meaning $\text{REC}(\tau)_{\neg s}$ does not satisfy both D and E.

GSLE: Let r be the removed soft variable ($r \neq s$). Since $C \vee s$ is satisfied only on s by τ and as $\text{REC}(\tau)(r) = 0$, $\text{REC}(\tau)_{\neg s}$ would leave $C \vee s \vee r \in \mathcal{F}^L$ unsatisfied.

□

Notice that in Theorem 5.9 $\mathcal{F}^L$ is a general literal-centric instance. Hence a straightforward induction yields the following corollary.

Corollary 5.10. *Let $n \in \mathbb{Z}_{>0}$, $\mathcal{P}_i \in \{\text{BVE}, \text{SE}, \text{SSR}, \text{GSLE}\}$ where $i = 1, \ldots, n$ and $\mathcal{F}^L$ be a literal-centric instance. Then for any locally minimal solution τ to $\mathcal{P}_n(\mathcal{P}_{n-1}(\ldots \mathcal{P}_1(\mathcal{F}^L)\ldots))$, the reconstructed solution $\text{REC}(\tau)$ to $\mathcal{F}^L$ is also locally minimal.*

The following example demonstrates how BCE fails to preserve locally minimal solutions.

Example 5.11. Let $\mathcal{F}$ be the clause-centric instance $\mathcal{F} = (\neg x \vee y) \wedge (x \vee \neg y)^1 \wedge (\neg x)^1$ with the corresponding literal-centric instance $\mathcal{F}^L = (\neg x \vee y) \wedge (x \vee \neg y \vee s_1) \wedge (\neg x \vee s_2)$, $w(s_1) = w(s_2) = 1$. The hard clause $(\neg x \vee y)$ is blocked on $\neg x$ so an application of BCE yields $\text{BCE}(\mathcal{F}) = (x \vee \neg y \vee s_1) \wedge (\neg x \vee s_2)$ which has the locally minimal solution $\tau = \{x, \neg y, \neg s_1, s_2\}$ with apparent cost of 1. However, since $\tau(C) = 0$, where $C = (y)$ and $C \vee \neg x \in \mathcal{F}^L$, we have $\text{REC}(\tau) = \tau_{\neg x}$ which is not locally minimal as $\text{REC}(\tau)_{\neg s_2}$ is also a solution to $\mathcal{F}^L$. The cost of τ is $\textsc{Cost}(\mathcal{F}, \text{REC}(\tau)) = 0 \neq 1$.

Unlike the other covered preprocessing techniques, BCE can also be applied directly to clause-centric instances. Perhaps the literal-centric view is the reason behind BCE misinterpreting costs? The following example demonstrates this is not the case.

Example 5.12. Applying BCE directly to the $\mathcal{F}$ in Example 5.11 leads to misinterpretation of costs as well. An application of BCE to $\mathcal{F}$ gives $(x \vee \neg y)^1 \wedge (\neg x)^1$ which has the solution $\tau = \{x, \neg y\}$ of cost 1. After reconstruction, the solution becomes $\tau = \{\neg x, \neg y\}$ with a cost of 0.

Let us then focus on the relationship of locally minimal solutions to solutions that are not. First, we address the question whether locally minimal solutions are the only solutions for which apparent cost equals cost when only preprocessing techniques preserving locally minimal solutions are used. The following example illustrates that this is not the case.

Example 5.13. Let $\mathcal{F} = (x)^1 \wedge (\bar{x} \vee y)$ for which $\mathcal{F}^L = (x \vee s) \wedge (\bar{x} \vee y)$ where $w(s) = 1$. Applying BVE to remove x gives $\text{BVE}(\mathcal{F}) = (y \vee s)$ which has a non-locally minimal solution $\tau = \{y, s\}$. Now $\text{REC}(\tau) = \tau_{\neg x}$ which is locally minimal for $\mathcal{F}^L$. Thus $\textsc{Appar-Cost}(\text{BVE}(\mathcal{F}), \tau) = \textsc{Cost}(\mathcal{F}, \text{REC}(\tau))$ even though τ was not locally minimal for $\text{BVE}(\mathcal{F})$.

By the above example, the concept of locally minimal solutions does not capture all solutions for which its apparent cost equals its cost. As such, there could be a more general property than local minimality which guarantees that costs are not misinterpreted. Finding such a property constitutes as interesting future work.

Since each non-locally minimal solution τ has some soft variables set to true unnecessarily, one would reckon that any locally minimal solution obtained from τ by changing the assignment of some soft variables to false would have a better cost than τ. The following example demonstrates that this intuition is false.

Example 5.14. Consider again the instance $\mathcal{F} = (x \vee y)^1 \wedge (\bar{x})^2$ from Example 5.5 with the corresponding literal-centric instance $\mathcal{F}^L = (x \vee y \vee s_1) \wedge (\bar{x} \vee s_2)$. Let then $\tau = \{\bar{y}, s_1, s_2\}$ and $\tau_{\neg s_1}$ be solutions to $\text{BVE}(\mathcal{F}) = (y \vee s_1 \vee s_2)$. Now $\tau_{\neg s_1}$ does not satisfy the clause $C = (y \vee s_1)$, where $C \vee x \in \mathcal{F}^L$, but τ does so $\text{REC}(\tau_{\neg s_1}) = \tau_{\neg s_1, x}$ and $\text{REC}(\tau) = \tau_{\neg x}$. For these solutions

$$\textsc{Appar-Cost}(\text{BVE}(\mathcal{F}), \tau_{\neg s_1}) = 2, \quad \textsc{Appar-Cost}(\text{BVE}(\mathcal{F}), \tau) = 3$$

and

$$\textsc{Cost}(\mathcal{F}, \text{REC}(\tau_{\neg s_1})) = 2, \quad \textsc{Cost}(\mathcal{F}, \text{REC}(\tau)) = 1.$$

So $\tau_{\neg s_1}$ has as a higher cost than τ even though $\tau_{\neg s_1}$ is locally minimal but τ is not.

Converting a solution into a locally minimal one by changing the assignment of some soft variables to false does not therefore necessarily improve the cost. This conversion however has the benefit of guaranteeing that the apparent cost of a solution equals its cost. At the time of writing, not much can be said on how the cost of a solution of a non-locally minimal solution relates with its apparent cost, other than that the cost of a solution is always at most equal to its apparent cost.

Out of the preprocessing techniques preserving locally minimal solutions, only with BVE the cost of the solutions can worsen when assigning unnecessary soft variables to false. This is due to the fact that the reconstruction procedures for SE, SSR and GSLE do not alter how non-soft variables are assigned.

6 Stochastic Local Search over Locally Minimal Solutions

In this section we first introduce a new SLS algorithm called LMS-SLS which is guaranteed to search over locally minimal solutions. We then evaluate the effectiveness of the different components of LMS-SLS. In addition, we show that the performance of LMS-SLS is on par and complementary to the state-of-the-art SLS-solver SATLike.

6.1 LMS-SLS

A naive way to circumvent misinterpretation of costs is to apply solution reconstruction every time a new solution is found. While this indeed avoids misinterpretations, the reconstruction steps consume a lot of time in case the number of solutions encountered is high. Thus this approach is not suited for SLS algorithms as they often visit tens of thousands of solutions in short periods of time. Our algorithm—titled LMS-SLS—searches over locally minimal solutions, thus ensuring no misinterpretation of costs if all preprocessing techniques used preserve locally minimal solutions.

The main idea behind LMS-SLS is to exploit the group structure of a preprocessed (literal-centric) instance. Given a preprocessed instance $\mathcal{P}(\mathcal{F})$, the clauses of $\mathcal{P}(\mathcal{F})$ are first partitioned into sets H and S containing the hard clauses and the soft clauses of $\mathcal{P}(\mathcal{F})$ respectively. The clauses of S are further partitioned into groups

$$S_g^x = \{C \mid C \in S, \text{and } x \text{ is the cheapest soft variable in } C\}.$$

In case $C \in S$ does not have a unique cheapest soft variable, the group of C is chosen uniformly at random from the set $\{S_g^x \mid x \text{ is a cheapest soft variable in } C\}$. Hard clauses C are identified with the group $\{C\}$ and we denote the collection of all hard groups and soft groups by H_g and S_g respectively.

The intuition underlying the groups is as follows: suppose τ is a solution to $\mathcal{P}(\mathcal{F})$ with $\tau_{\neg x}(S_g^x) = 0$. Since τ is a solution, we must have $\tau(x) = 1$ as at least one clause in S_g^x is unsatisfied. To not incur cost from x we need to satisfy all the clauses in S_g^x. LMS-SLS utilizes this observation by focusing on satisfying groups of clauses rather than single clauses resembling Group MaxSAT [35].

In contrast to SATLike, LMS-SLS only searches over the non-soft variables. The soft variables serve only in evaluating cost so after flipping a variable LMS-SLS updates the working assignment τ so that the invariant "$\tau(x) = 0$ iff $\tau(S_g^x) = 1$" holds for all soft variables x. Assuming each soft clause contains only one soft variable, enforcing the invariant already implies that the solutions LMS-SLS finds are locally minimal. For the instances of the weighted incomplete track from the 2019 MaxSAT evaluations this assumption holds for approximately 68% of the instances. In case some clauses contain more than one soft variable, LMS-SLS generates a locally minimal solution τ' by first setting $\tau' \coloneqq \tau$ and then iteratively $\tau'(x) = 0$ if $\tau'_{\neg x}$ is a solution, considering the soft variables x in decreasing order of $w(x)$.

Algorithm 2 contains the pseudocode of LMS-SLS. The algorithm begins by preprocessing the given clause-centric MaxSAT instance to obtain the literal centric

Algorithm 2: LMS-SLS

Input : A clause-centric MaxSAT instance $\mathcal{F}$
Output: A solution τ along with its cost if a solution is found, otherwise "no solution found".

```
1  Preprocess F to obtain P(F)
2  Form the groups H_g and S_g
3  τ := a random initial assignment for P(F)
4  for x_c ∈ S(F) do
5  |  if S_g^{x_c} is satisfied then τ(x_c) = 0
6  |  else τ(x_c) := 1
7  τ := decimation on τ and P(F) without soft variables
8  τ* := τ
9  while no termination criteria is met do
10 |  if τ is a solution then
11 |  |  τ' := greedily computed locally minimal solution from τ
12 |  |  if APPAR-COST(P(F), τ') < APPAR-COST(P(F), τ*) then τ* := τ'
13 |  if D := {v ∉ S(F) | score(v) > 0} ≠ ∅ then
14 |  |  v := a variable in D chosen by the BMS strategy
15 |  |  τ := τ with v flipped
16 |  else
17 |  |  update group weights
18 |  |  if ∃G ∈ H_g that is unsatisfied then
19 |  |  |  G := an unsatisfied group from H_g
20 |  |  else G := a group from S_g that is satisfied at most by a soft variable
21 |  |  τ := SATISFY(τ, G)
22 |  update τ(x_c) for all x_c ∈ S(F)
23 if τ* is a solution then
24 |  return REC(τ*), COST(F, REC(τ*))
25 else return "no solution found"
```

instance $\mathcal{P}(\mathcal{F})$ (line 1). After this, the algorithm groups the clauses on line 2 and generates a working assignment on lines 3–8. Lines 4–6 guarantee local minimality in the case each soft clause in $\mathcal{P}(\mathcal{F})$ contains one soft variable. The working assignment is refined on line 7 by the unit propagation based decimation procedure discussed in Section 3.2.1.

The search itself begins on line 9. In the beginning of each iteration, LMS-SLS first checks whether the current assignment is a solution and if so, computes a locally minimal solution τ' on line 11 in the greedy manner described above. Then, if the apparent cost of τ' is lower than apparent cost of the currently the best found solution τ^*, τ^* is set to τ'. Recall that the apparent cost of a locally minimal solution equals its cost.

Next on lines 13–15 a greedy step similar to SATLike is performed if the set D of variables with a positive *score* is non-empty. If so, one such variable is chosen

according to the best from multiple selection strategy [14] where the number of samples is a user defined parameter t. We adopt the dynamic weighting scheme from SATLike but applied to groups instead of clauses. In detail, we initialize the weight of each hard group to one and the weight of each $S_g^x \in S_g$ to $w(x)$. During search the weight of each unsatisfied hard group is increased by a constant h_inc. Following Definition 3.4, the score of a non-soft variable v is defined as the increase in the total weight of satisfied groups if v is flipped.

In case the set D is empty—i.e. a greedy step is not possible—LMS-SLS first updates the weights of the hard groups. Then a hard or soft group that is satisfied at most by a soft variable is chosen to be satisfied next, preferring hard groups. The assignment τ is then modified by the subroutine SATISFY so that G is satisfied without soft variables. If $G \in H_g$, SATISFY flips a variable in G with the highest score in it, breaking ties randomly. Otherwise the subroutine employs a simple local search algorithm which flips a random variable in G with probability $1 - gp$ or instead (with probability gp) a variable with the highest *make* (breaking ties randomly). Here the *make* of a (non-soft) variable v is defined simply as the number of unsatisfied clauses that would become satisfied after flipping v. The subroutine then iterates this process until G is satisfied.

Lastly, on line 22 LMS-SLS modifies τ so that for each soft variable x we have $\tau(x) = 1$ iff $\tau(S_g^x) = 1$. After the termination of the search loop, if a solution was found, then a solution for the original instance $\mathcal{F}$ is reconstructed from τ^*.

Before moving onto the empirical evaluations, we note that the process of turning a solution into a locally minimal by LMS-SLS on line 11 might inadvertently worsen the cost of the found solution. This follows from observing that after the procedure sets soft variables to false, fewer of the clauses in $\mathcal{F}^L$ may be satisfied.

Example 6.1. Let $\mathcal{F} = (x \vee a)^1 \wedge (x \vee b)^4 \wedge (\bar{x} \vee c)^3 \wedge (\bar{x})^3$ so that

$$\mathcal{F}^L = (x \vee a \vee s_1) \wedge (x \vee b \vee s_2) \wedge (\bar{x} \vee c \vee s_3) \wedge (\bar{x} \vee s_4)$$

where $w(s_1) = 1$, $w(s_2) = 4$, $w(s_3) = 3$, $w(s_4) = 3$. Applying BVE to remove the variable x gives

$$\text{BVE}(\mathcal{F}) = (a \vee c \vee s_1 \vee s_3) \wedge (a \vee s_1 \vee s_4) \wedge (b \vee c \vee s_2 \vee s_3) \wedge (b \vee s_2 \vee s_4),$$

which has the non locally minimal solution $\tau = \{\bar{a}, \bar{b}, \bar{c}, s_1, s_2, s_3, s_4\}$. Each $\tau_{\neg s_i}$ is also a solution and so as s_2 has the highest weight, LMS-SLS sets $\tau' = \tau_{\neg s_2}$. After that, $\tau'_{\neg s_i}$ is a solution only for $i = 1$ so LMS-SLS sets $\tau' = \tau'_{\neg s_1}$. But $\text{REC}(\tau) = \tau_{\neg x}$ has cost $1 + 4 = 5$ while the solution $\text{REC}(\tau') = \tau'_x$ has cost $3 + 3 = 6$.

Even though turning a solution into a locally minimal one may increase its cost, the primary objective of LMS-SLS is still achieved. Each new recorded locally minimal solution does improve the cost if used preprocessing techniques preserve locally minimal solutions.

6.2 Empirical Evaluation

We introduced the LMS-SLS-algorithm guaranteeing that recorded solutions are locally minimal. Given that the preprocessing techniques preserve locally minimal

solutions, searching only over locally minimal solutions hopefully leads to performance improvements. This is indeed the case in our implementation of LMS-SLS as we will show in Section 6.2.1 where we investigate the effects of removing preprocessing and grouping in our implementation of LMS-SLS.

The impact of preprocessing has been more modest in context of MaxSAT than in SAT [114]. To see if this is the case with SATLike as well, we also investigate the effects of preprocessing in SATLike. In addition, we compare SATLike with LMS-SLS and observe complementary behavior i.e. SATLike performs better on some benchmark families than LMS-SLS and vice versa.

All the experiments were run on 2.4-GHz Intel Xeon E5-2680-v4, 256-GB computers. A time limit of 300 seconds (preprocessing time included) was enforced along with a memory limit of 32GB. The benchmark instances used are the ones from the weighted incomplete track of the 2019 MaxSAT evaluations[1].

For the experiments, we built our implementation of LMS-SLS in C++ on top of the MIT-licensed SATLike-framework. Following [14], we set $h\_inc = 300$ if the average weight of the soft clauses is over 10000 and otherwise $h\_inc = 3$. Based on preliminary experiments, we set $t = 15$ and $gp = 0.8$. We opted for MaxPre [88] as the underlying preprocessor. Since preprocessing can take quite some time, we limited the preprocessing time to ten seconds which is enough time for MaxPre to finish on most of the instances. We noticed that MaxPre finished preprocessing within ten seconds on 250 out of the 297 instances.

6.2.1 Effectiveness of Preprocessing and Grouping

We investigate the impact of preprocessing and grouping in LMS-SLS. For this end, we removed grouping from LMS-SLS to see if preprocessing alone would yield better results. Furthermore, we also removed preprocessing (along with grouping) from the picture to see if these two main ideas combined were any good in reality. Observe that removing preprocessing and grouping leads to an almost identical algorithm to SATLike; the only distinction is in the random walk component where SATLike chooses a variable from the selected clause with the highest score where where as in LMS-SLS with preprocessing and grouping removed the variable might be chosen uniformly at random (with probability $1 - gp$). Finally, we included BCE in the preprocessing procedure. The variants of the LMS-SLS algorithm are thus

default: LMS-SLS in full, including all preprocessing techniques covered that preserve locally-minimal solutions,

+BCE: LMS-SLS in full, but including BCE—which does not preserve locally minimal solutions—in preprocessing,

-G: LMS-SLS excluding clause grouping i.e. each clause is treated as a group of size one,

-G-pre: LMS-SLS without preprocessing and clause grouping.

[1] Available at `https://maxsat-evaluations.github.io/2019/benchmarks.html`.

Table 1: Pairwise comparison of different variants of LMS-SLS

		-G-pre	**-G**	**-G-pre**	**default**	**+BCE**	**default**
Domain	#	#wins (score)	#wins (score)	#wins (score)	#wins (score)	#wins (score)	#wins (score)
abstraction-refinement	10	**8 (0.981)**	2 (0.884)	4 (0.808)	**6 (0.847)**	**8 (1.000)**	3 (0.924)
af-synthesis	16	9 (0.967)	**9 (0.987)**	0 (0.779)	**16 (1.000)**	15 (0.997)	**16 (1.000)**
BTBNSL	14	**9 (0.993)**	5 (0.982)	**8 (0.978)**	6 (0.928)	7 (0.990)	**7 (0.991)**
causal-discovery	16	**10 (0.738)**	6 (0.705)	1 (0.394)	**15 (0.980)**	10 (0.852)	**11 (0.899)**
correlation-clustering	23	11 (0.691)	**12 (0.640)**	11 (0.909)	**12 (0.837)**	10 (0.895)	**13 (0.974)**
drmx-cryptogen	1	0 (0.223)	**1 (1.000)**	0 (0.223)	**1 (1.000)**	**1 (1.000)**	**1 (1.000)**
hs-timetabling	10	1 (0.520)	**10 (1.000)**	1 (0.411)	**10 (1.000)**	5 (0.948)	**6 (0.940)**
InterpretableClassifiers	15	**10 (0.946)**	5 (0.769)	0 (0.559)	**15 (1.000)**	**11 (0.994)**	9 (0.984)
lisbon-wedding	14	**12 (0.991)**	8 (0.901)	**11 (0.960)**	8 (0.887)	9 (0.897)	**11 (0.986)**
max-realizability	13	**13 (1.000)**	8 (0.707)	10 (0.998)	**13 (1.000)**	**13 (1.000)**	**13 (1.000)**
maxcut	16	**16 (1.000)**	**16 (1.000)**	0 (0.955)	**16 (1.000)**	**16 (1.000)**	**16 (1.000)**
metro	2	**2 (1.000)**	0 (0.691)	0 (0.860)	**2 (1.000)**	**2 (1.000)**	1 (0.988)
min-width	17	**16 (0.985)**	1 (0.961)	0 (0.907)	**17 (1.000)**	1 (0.978)	**16 (1.000)**
MinWeightDominatingSet	7	6 (1.000)	**7 (1.000)**	**4 (1.000)**	3 (1.000)	5 (1.000)	**6 (1.000)**
mpe	15	**13 (0.998)**	2 (0.911)	0 (0.699)	**15 (1.000)**	**15 (1.000)**	**15 (1.000)**
pseudoBoolean	7	**7 (1.000)**	6 (0.857)	**7 (1.000)**	6 (0.857)	**7 (1.000)**	**7 (1.000)**
railway-transport	4	2 (0.764)	**3 (0.999)**	2 (0.758)	**3 (0.750)**	2 (0.973)	**3 (0.750)**
ramsey	12	**12 (1.000)**	**12 (1.000)**	0 (0.438)	**12 (1.000)**	**12 (1.000)**	**12 (1.000)**
RBAC maintenance	15	**15 (1.000)**	1 (0.753)	5 (0.818)	**10 (0.917)**	**8 (0.962)**	7 (0.987)
relational-inference	2	1 (0.545)	**1 (0.825)**	**1 (0.517)**	1 (0.500)	**1 (0.691)**	1 (0.500)
set-covering	12	**12 (1.000)**	**12 (1.000)**	8 (0.994)	**10 (0.994)**	**11 (0.999)**	**11 (0.999)**
shiftdesign	11	**11 (1.000)**	**11 (1.000)**	**11 (1.000)**	**11 (1.000)**	**11 (1.000)**	**11 (1.000)**
spot5	5	**5 (1.000)**	0 (0.973)	0 (0.921)	**5 (1.000)**	2 (0.999)	**3 (1.000)**
staff-scheduling	11	**7 (0.962)**	4 (0.961)	4 (0.937)	**7 (0.978)**	**6 (0.985)**	6 (0.940)
tcp	13	**10 (0.997)**	7 (0.996)	1 (0.967)	**13 (1.000)**	**9 (0.996)**	8 (0.994)
timetabling	16	**10 (0.925)**	10 (0.873)	4 (0.697)	**16 (1.000)**	**12 (0.988)**	8 (0.964)
total	297	**228 (0.925)**	159 (0.887)	93 (0.810)	**249 (0.957)**	209 (0.969)	**221 (0.974)**

One way to compare implementations of MaxSAT algorithms is the number of wins (best solution found) over individual instances (see e.g. [14, 28, 39, 40, 74]). In the 2019 MaxSAT evaluations, a score metric was used (larger value is better). The score of a MaxSAT solver on an instance I is

$$\frac{\text{The cost of the best solution found by the considered solvers}}{\text{The cost of the best found solution by } I}.$$

and the score of a solver over a set of instances is its average score over the instances.

The score gives a more accurate view on the relative performance of the solvers compared. Consider the case where two solvers A and B beat each other in exactly half of the given instances. Suppose further that on the half of the instances where A wins, it wins just barely and on the instance where B wins, B wins by a wide margin. Now if we only compared the number of wins, the solver A and B seem equally good though this certainly is not the case. The solvers are practically equally good on the instances where A beats B, but otherwise B performs much better than A.

Now let us consider what would happen if we were to compare solvers A and B using the score metric. On each of the instances where A beats B, the score is exactly one for A (as it found the best solution) and for B the score is something close to one. For the other instances, the score of A is close to zero while for B the score is one. Thus the average score for A is near 0.5 and for B it is near 1.0, indicating that B is clearly the better solver which is indeed the case.

In our comparisons, we included both the number of wins and the scores. The pairwise results are listed in Table 1. We compared **-G-pre** with **-G**, **-G-pre** with **default** and **default** with +**BCE**. From Table 1 we see that preprocessing decreases the performance of LMS-SLS if grouping is switched off. The score of **-G** is almost 4%-units lower than the score of **-G-pre** although including preprocessing while not grouping seems to significantly improve the performance in the instances of hs-timetabling and drmx-cryptogen. In other domains, the performance of **-G** is either on-par with **-G-pre** or worse. In domains such as abstraction-refinement and InterpretableClassifiers **-G-pre** significantly out performs **-G**.

Switching grouping on when including preprocessing results in drastic performance improvements compared to **-G-pre**. This is likely due to the fact that after preprocessing, the algorithm satisfies only one soft clause chosen uniformly at random at a time instead of all the clauses sharing the same label. Therefore it is unlikely that all soft clauses sharing a soft variable s are satisfied and so the cost is increased by the weight of s. Without preprocessing this kind of behavior does not occur as there is no group structure. The variant **-G-pre** is does not beat **default** in any of the instances in domains such as InterpretableClassifiers, min-width, metro, spot5. Observe that in e.g. InterpretableClassifiers and min-width **-G-pre** outperformed **-G** by a wide margin, but clearly loses to **default**. This serves as evidence for the effectiveness of combining grouping and preprocessing.

Finally, we note that including BCE—a preprocessing technique that does not preserve locally minimal solutions as witnessed in examples 5.11 and 5.12—in preprocessing downgrades the overall performance of LMS-SLS. We hypothesize that the adverse effect of BCE on the performance of LMS-SLS is at least partly explained by the solver misinterpreting costs.

Table 2: Pairwise comparison of SATLike, SATLike+pre and default

		SATLike	**SATLike+pre**	**SATLike**	**default**	**SATLike+pre**	**default**
Domain	#	#wins (score)	#wins (score)	#wins (score)	#wins (score)	#wins (score)	#wins (score)
abstraction-refinement	10	**9 (1.000)**	1 (0.976)	**10 (1.000)**	0 (0.643)	**7 (0.999)**	3 (0.660)
af-synthesis	16	**16 (1.000)**	0 (0.889)	3 (0.923)	**13 (0.983)**	1 (0.832)	**15 (0.997)**
BTBNSL	14	6 (0.931)	**8 (0.992)**	**9 (0.995)**	5 (0.946)	**10 (0.993)**	4 (0.886)
causal-discovery	16	2 (0.324)	**15 (0.966)**	1 (0.332)	**16 (1.000)**	**11 (0.918)**	11 (0.889)
correlation-clustering	23	3 (0.866)	**20 (0.953)**	**15 (0.969)**	8 (0.759)	**14 (0.972)**	9 (0.699)
drmx-cryptogen	1	**1 (1.000)**	0 (0.977)	**1 (1.000)**	0 (0.902)	**1 (1.000)**	0 (0.923)
hs-timetabling	10	2 (0.412)	**10 (1.000)**	1 (0.363)	**10 (1.000)**	**6 (0.869)**	5 (0.894)
lisbon-wedding	14	6 (0.959)	**12 (0.987)**	**13 (0.996)**	5 (0.735)	**14 (1.000)**	4 (0.725)
max-realizability	13	11 (0.997)	**13 (1.000)**	11 (0.997)	**13 (1.000)**	**13 (1.000)**	**13 (1.000)**
maxcut	16	13 (0.999)	**15 (0.999)**	14 (1.000)	**15 (1.000)**	**15 (0.999)**	14 (1.000)
InterpretableClassifiers	15	**9 (0.981)**	8 (0.963)	4 (0.938)	**13 (0.982)**	4 (0.933)	**13 (0.994)**
metro	2	0 (0.865)	**2 (1.000)**	0 (0.915)	**2 (1.000)**	**2 (1.000)**	0 (0.945)
min-width	17	**16 (0.999)**	1 (0.981)	3 (0.980)	**14 (0.998)**	1 (0.964)	**16 (1.000)**
MinWeightDominatingSet	7	**6 (0.974)**	2 (0.930)	0 (0.436)	**7 (1.000)**	0 (0.403)	**7 (1.000)**
mpe	15	**13 (1.000)**	11 (0.999)	**15 (1.000)**	4 (0.993)	**15 (1.000)**	4 (0.994)
RBACMaintenance	15	**11 (0.998)**	5 (0.960)	**12 (0.994)**	3 (0.803)	**10 (0.979)**	5 (0.819)
pseudoBoolean	7	**7 (1.000)**	6 (0.857)	**7 (1.000)**	6 (0.857)	**7 (1.000)**	**7 (1.000)**
railway-transport	4	2 (0.938)	**3 (0.879)**	**4 (1.000)**	1 (0.612)	**3 (0.996)**	2 (0.685)
ramsey	12	**12 (1.000)**	4 (0.928)	**10 (0.988)**	6 (0.963)	4 (0.953)	**11 (0.999)**
relational-inference	2	0 (0.282)	**2 (1.000)**	**2 (1.000)**	0 (0.269)	**2 (1.000)**	0 (0.089)
set-covering	12	7 (0.994)	**12 (1.000)**	**12 (1.000)**	0 (0.883)	**12 (1.000)**	0 (0.878)
shiftdesign	11	**11 (1.000)**	**11 (1.000)**	**11 (1.000)**	**11 (1.000)**	**11 (1.000)**	**11 (1.000)**
spot5	5	1 (0.906)	**4 (0.977)**	1 (0.919)	**4 (0.994)**	**4 (0.983)**	1 (0.986)
staff-scheduling	11	**6 (0.968)**	5 (0.927)	**6 (0.979)**	5 (0.932)	**6 (0.976)**	5 (0.974)
tcp	13	**13 (1.000)**	1 (0.967)	**10 (0.998)**	9 (0.995)	0 (0.970)	**13 (1.000)**
timetabling	16	7 (0.778)	**13 (0.918)**	5 (0.780)	**15 (0.993)**	10 (0.866)	**10 (0.927)**
total	297	**190 (0.905)**	184 (0.963)	180 (0.902)	**185 (0.919)**	**183 (0.947)**	183 (0.906)

6.2.2 LMS-SLS versus SATLike

We augmented SATLike with MaxPre to evaluate the effects of preprocessing on the solver. The parameters of SATLike are set according to [14]. To gain insight on the effects of preprocessing on SATLike we ran experiments on two variants of SATLike, namely one without and one with preprocessing. We use the name SATLike to refer to the variant without preprocessing (the original SATLike) and SATLike+pre to SATLike with preprocessing included. We included the same preprocessing techniques as the **default** variant of LMS-SLS. We also compare SATLike and SATLike+pre with **default**. Table 6.2.1 displays the pairwise comparisons.

Interestingly we see that according to the number of wins, SATLike performs better without preprocessing in terms of the number of wins although the difference is narrow; SATLike outperforms SATLike+pre on six more instances. However, judging by the scores SATLike+pre is the clear winner. This is explained by noticing that in most cases when SATLike wins, it does so only barely. For example, in the min-width domain SATLike wins on 16 out of the 17 instances but the score of SATLike+pre in the domain is very close to one (0.981). This means SATLike+pre finds good quality solutions for the min-width instances but loses slightly to SATLike. For this reason, we argue that preprocessing has an overall positive impact on SATLike, even though locally minimal solutions are not at all guaranteed.

Though the overall performance of SATLike increases with preprocessing, the two variants exhibit complementary performance. In the af-synthesis domain SATLike beats SATLike+pre by a wide margin both in terms of wins and score. On the other hand, SATLike only has a score of 0.412 in hs-timetabling compared to SATLike+pre's score of 1.000 i.e. in at least half of the domain's instances the best solutions found by SATLike have a cost over twice of those found by SATLike+pre.

Complementary behavior is also found in both variants of SATLike and the **default** variant of LMS-SLS. The **default** variant of LMS-SLS performs slightly better than SATLike both in terms of wins and score but both solvers outperform the other by a wide margin in multiple benchmark domains. SATLike loses by a lot to **default** in domains such as causal-discovery and hs-timetabling but wins handsomely in e.g. lisbon-wedding and set-covering.

Comparing SATLike+pre and **default** we see that overall, SATLike+pre performs better but the difference is not as large as it is with SATLike and SATLike+pre. While equaling in the number of wins, SATLike+pre has a 4.1%-units larger score than **default**. Complementary behavior is observed in domains such as af-synthesis, correlation-clustering and set-covering.

One of the main disadvantages of LMS-SLS is the overhead related to using underlying the group structure in guiding the search. In MaxSAT SLS-solvers such as SATLike, the score of a variable v depends only on the clauses v occurs in. With LMS-SLS, the score of v depends not only the clauses containing it but on all clauses that occur in the same group as v. More explicitly, if C and D are two clauses in the same group with $v \in C$, $v \notin D$, then the score of v in LMS-SLS also depends on D. Therefore flipping a variable affects the score of many more variables in LMS-SLS than in SATLike.

7 Conclusions

In this thesis, we focused on the MaxSAT optimization paradigm. We examined the effects of MaxSAT preprocessing on the relative quality of solutions and developed a new SLS solver to mitigate the issues related to preprocessing. We showed that preprocessing may lead to the misinterpretation of costs of non-optimal solutions during search both in theory and in practice which to our best knowledge has thus far remained unnoticed. We introduced a class of solutions called locally minimal solutions for which the apparent cost equals cost if the used preprocessing techniques preserve locally minimal solutions. Most of the covered preprocessing techniques have this property. Therefore, MaxSAT solvers prepended with a preprocessor limited to these preprocessing techniques do not misinterpret costs if they search only over locally minimal solutions.

We developed such a solver, LMS-SLS, that preprocesses the input instances using only techniques preserving locally minimal solutions, and groups clauses to steer the search towards locally minimal solutions. We then analyzed the effectiveness of preprocessing and grouping, showing that removing preprocessing and grouping from the algorithm leads to a significant decrease in performance. On top of that, we showed that including BCE—a preprocessing technique that does not preserve locally minimal solutions—decreases the performance of LMS-SLS. This may very well be due to the fact that including BCE can lead to misinterpretation of costs. Finally we showed that LMS-SLS is competitive with SATLike, the state-of-the-art SLS solver SATLike for MaxSAT.

Adapting SAT-based MaxSAT solvers to search over locally minimal solutions is a promising research direction with potential in increasing their performance in practice. Replacing the score-function in LMS-SLS with another one to allow faster update of scores could lead to a more efficient solver. Ways other than grouping and the greedy method in LMS-SLS to guarantee locally minimal solutions should also be considered. For example, locally minimal solutions can be enforced by adding constraints after converting a clause-centric instance to a literal-centric instance; for each soft clause $(l_1 \vee \ldots \vee l_n \vee s)$ with the soft variable s add the hard constraints $(\bar{l_i} \vee \bar{s})$ for each $i = 1, \ldots, n$ to enforce that s is falsified if any l_i is satisfied. Concepts other than locally minimal solutions that also guarantee no misinterpretation of costs could also be explored.

Acknowledgments

This work presented in the thesis was partially funded by Academy of Finland under grand 322869. Computational resources were provided by Finnish Grid and Cloud Infrastructure (FCGI) [121]. I am grateful for the invaluable advice received and knowledge gained during my time as a research assistant in the Constraint Reasoning and Optimization group led by associate professor Matti Järvisalo. I thank Jeremias Berg and Matti Järvisalo, the supervisors and two of the examiners of this thesis, for their comments and guidance as well as for their collaboration on [31]. Finally, I thank professor Juha Kontinen for acting as one of the examiners of this thesis.

References

1 E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

2 B. Roy, "Transitivité et connexité," *C. R. Acad. Sci. Paris*, vol. 249, pp. 216–218, 1959.

3 A. Shimbel, "Structure in communication nets," in *Proceedings of the symposium on information networks*, pp. 119–203, Polytechnic Institute of Brooklyn, 1954.

4 M. M. Flood, "The traveling-salesman problem," *Operations research*, vol. 4, no. 1, pp. 61–75, 1956.

5 D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, "The traveling salesman problem: a computational study," 2006.

6 M. Hahsler and K. Hornik, "TSP-infrastructure for the traveling salesperson problem," *Journal of Statistical Software*, vol. 23, no. 2, pp. 1–21, 2007.

7 A. M. Jaffe, "The millennium grand challenge in mathematics," *Notices of the AMS*, vol. 53, no. 6, 2006.

8 H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

9 K. Bunte, M. Järvisalo, J. Berg, P. Myllymäki, J. Peltonen, and S. Kaski, "Optimal neighborhood preserving visualization by maximum satisfiability," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (C. E. Brodley and P. Stone, eds.), pp. 1694–1700, AAAI Press, 2014.

10 V. D'Silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, 2008.

11 T. Kropf, *Introduction to Formal Hardware Verification*. Springer, 1999.

12 J. F. Bard, G. Yu, and M. F. Arguello, "Optimizing aircraft routings in response to groundings and delays," *Iie Transactions*, vol. 33, no. 10, pp. 931–947, 2001.

13 J. Berg, E. Demirovic, and P. J. Stuckey, "Core-boosted linear search for incomplete MaxSAT," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (L. Rousseau and K. Stergiou, eds.), vol. 11494 of *Lecture Notes in Computer Science*, pp. 39–56, Springer, 2019.

14 Z. Lei and S. Cai, "Solving (weighted) partial MaxSAT by dynamic local search for SAT," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (J. Lang, ed.), pp. 1346–1352, ijcai.org, 2018.

15 P. Saikko, J. Berg, and M. Järvisalo, "LMHS: A SAT-IP hybrid MaxSAT solver," in *Theory and Applications of Satisfiability Testing* (N. Creignou and D. L. Berre, eds.), vol. 9710 of *Lecture Notes in Computer Science*, pp. 539–546, Springer, 2016.

16 M. Cornu, T. Cazenave, and D. Vanderpooten, "Perturbed decomposition algorithm applied to the multi-objective traveling salesman problem," *Comput. Oper. Res.*, vol. 79, pp. 314–330, 2017.

17 L. A. Wolsey and G. L. Nemhauser, *Integer and combinatorial optimization*, vol. 55. John Wiley & Sons, 1999.

18 A. Schrijver, *Theory of linear and integer programming.* Wiley-Interscience series in discrete mathematics and optimization, Wiley, 1999.

19 C. M. Li and F. Manyà, "MaxSAT, hard and soft constraints," in *Handbook of Satisfiability* (A. Biere, M. Heule, H. van Maaren, and T. Walsh, eds.), vol. 185 of *Frontiers in Artificial Intelligence and Applications*, pp. 613–631, IOS Press, 2009.

20 J. Davies and F. Bacchus, "Solving MAXSAT by solving a sequence of simpler SAT instances," in *Principles and Practice of Constraint Programming* (J. H. Lee, ed.), vol. 6876 of *Lecture Notes in Computer Science*, pp. 225–239, Springer, 2011.

21 X. Liao, H. Zhang, M. Koshimura, R. Huang, and W. Yu, "Maximum satisfiability formulation for optimal scheduling in overloaded real-time systems," in *PRICAI 2019: Trends in Artificial Intelligence* (A. C. Nayak and A. Sharma, eds.), vol. 11670 of *Lecture Notes in Computer Science*, pp. 618–631, Springer, 2019.

22 P. K. Lin and S. P. Khatri, "Efficient cancer therapy using boolean networks and Max-SAT-based ATPG," in *2011 IEEE International Workshop on Genomic Signal Processing and Statistics*, pp. 87–90, IEEE, 2011.

23 T. Sandholm, "An algorithm for optimal winner determination in combinatorial auctions," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (T. Dean, ed.), pp. 542–547, Morgan Kaufmann, 1999.

24 M. Jose and R. Majumdar, "Cause clue clauses: error localization using maximum satisfiability," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation* (M. W. Hall and D. A. Padua, eds.), pp. 437–446, ACM, 2011.

25 C. S. Zhu, G. Weissenbacher, and S. Malik, "Post-silicon fault localisation using maximum satisfiability and backbones," in *International Conference on Formal Methods in Computer-Aided Design* (P. Bjesse and A. Slobodová, eds.), pp. 63–66, FMCAD Inc., 2011.

26 X. Zhou, H. Xiong, and F. He, "Hybrid partition-and network-level scheduling design for distributed integrated modular avionics systems," *Chinese Journal of Aeronautics*, vol. 33, no. 1, pp. 308–323, 2020.

27 E. Demirovic, N. Musliu, and F. Winter, "Modeling and solving staff scheduling with partial weighted maxSAT," *Annals OR*, vol. 275, no. 1, pp. 79–99, 2019.

28 C. Luo, S. Cai, K. Su, and W. Huang, "CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability," *Artif. Intell.*, vol. 243, pp. 26–44, 2017.

29 A. Balint, *Engineering stochastic local search for the satisfiability problem*. PhD thesis, University of Ulm, 2014.

30 A. Belov, A. Morgado, and J. Marques-Silva, "SAT-based preprocessing for MaxSAT," in *Logic for Programming, Artificial Intelligence, and Reasoning* (K. L. McMillan, A. Middeldorp, and A. Voronkov, eds.), vol. 8312 of *Lecture Notes in Computer Science*, pp. 96–111, Springer, 2013.

31 M. Leivo, J. Berg, and M. Järvisalo, "Preprocessing in incomplete MaxSAT solving," in *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, Frontiers in Artificial Intelligence and Applications, (Netherlands), IOS PRESS, in press.

32 E. Mendelson, *Introduction to mathematical logic*. CRC press, 2009.

33 G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of reasoning*, pp. 466–483, Springer, 1983.

34 S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing* (M. A. Harrison, R. B. Banerji, and J. D. Ullman, eds.), pp. 151–158, ACM, 1971.

35 F. Heras, A. Morgado, and J. Marques-Silva, "An empirical study of encodings for group MaxSAT," in *Advances in Artificial Intelligence* (L. Kosseim and D. Inkpen, eds.), vol. 7310 of *Lecture Notes in Computer Science*, pp. 85–96, Springer, 2012.

36 E. Demirovic and N. Musliu, "MaxSAT-based large neighborhood search for high school timetabling," *Comput. Oper. Res.*, vol. 78, pp. 172–180, 2017.

37 J. Marques-Silva, M. Janota, A. Ignatiev, and A. Morgado, "Efficient model based diagnosis with maximum satisfiability," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (Q. Yang and M. J. Wooldridge, eds.), pp. 1966–1972, AAAI Press, 2015.

38 C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su, "CCLS: An efficient local search algorithm for weighted maximum satisfiability," *IEEE Trans. Computers*, vol. 64, no. 7, pp. 1830–1843, 2015.

39 S. Cai, C. Luo, J. Thornton, and K. Su, "Tailoring local search for partial MaxSAT," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (C. E. Brodley and P. Stone, eds.), pp. 2623–2629, AAAI Press, 2014.

40 S. Cai, C. Luo, J. Lin, and K. Su, "New local search methods for partial MaxSAT," *Artif. Intell.*, vol. 240, pp. 1–18, 2016.

41 C. M. Li, F. Manyà, and J. Planes, "Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers," in *Principles and Practice of Constraint Programming* (P. van Beek, ed.), vol. 3709 of *Lecture Notes in Computer Science*, pp. 403–414, Springer, 2005.

42 A. Morgado, F. Heras, M. H. Liffiton, J. Planes, and J. Marques-Silva, "Iterative and core-guided MaxSAT solving: A survey and assessment," *Constraints*, vol. 18, no. 4, pp. 478–534, 2013.

43 F. Heras, A. Morgado, and J. Marques-Silva, "Core-guided binary search algorithms for maximum satisfiability," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence* (W. Burgard and D. Roth, eds.), AAAI Press, 2011.

44 C. Ansótegui, F. Didier, and J. Gabàs, "Exploiting the structure of unsatisfiable cores in MaxSAT," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (Q. Yang and M. J. Wooldridge, eds.), pp. 283–289, AAAI Press, 2015.

45 A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided MaxSAT with soft cardinality constraints," in *Principles and Practice of Constraint Programming* (B. O'Sullivan, ed.), vol. 8656 of *Lecture Notes in Computer Science*, pp. 564–573, Springer, 2014.

46 M. Alviano, C. Dodaro, and F. Ricca, "A MaxSAT algorithm using cardinality constraints of bounded size," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (Q. Yang and M. J. Wooldridge, eds.), pp. 2677–2683, AAAI Press, 2015.

47 N. Narodytska and F. Bacchus, "Maximum satisfiability using core-guided MaxSAT resolution," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (C. E. Brodley and P. Stone, eds.), pp. 2717–2723, AAAI Press, 2014.

48 F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko, "Reduced cost fixing for maximum satisfiability," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (J. Lang, ed.), pp. 5209–5213, ijcai.org, 2018.

49 J. Davies and F. Bacchus, "Exploiting the power of MIP solvers in MAXSAT," in *Theory and Applications of Satisfiability Testing* (M. Järvisalo and A. V. Gelder, eds.), vol. 7962 of *Lecture Notes in Computer Science*, pp. 166–181, Springer, 2013.

50 J. Davies and F. Bacchus, "Postponing optimization to speed up MAXSAT solving," in *Principles and Practice of Constraint Programming* (C. Schulte, ed.), vol. 8124 of *Lecture Notes in Computer Science*, pp. 247–262, Springer, 2013.

51 M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A partial Max-SAT solver," *J. Satisf. Boolean Model. Comput.*, vol. 8, no. 1/2, pp. 95–100, 2012.

52 D. L. Berre and A. Parrain, "The Sat4j library, release 2.2," *J. Satisf. Boolean Model. Comput.*, vol. 7, no. 2-3, pp. 59–6, 2010.

53 J. Berg and M. Järvisalo, "Unifying reasoning and core-guided search for maximum satisfiability," in *European Conference on Logics in Artificial Intelligence* (F. Calimeri, N. Leone, and M. Manna, eds.), vol. 11468 of *Lecture Notes in Computer Science*, pp. 287–303, Springer, 2019.

54 N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 4, pp. 543–560, 2003.

55 S. Joshi, P. Kumar, R. Martins, and S. Rao, "Approximation strategies for incomplete MaxSAT," in *International Conference on Principles and Practice of Constraint Programming*, pp. 219–228, Springer, 2018.

56 M. Chiarandini, I. Dumitrescu, and T. Stützle, "Stochastic local search algorithms for the graph colouring problem," in *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 2: Contemporary and Emerging Applications* (T. F. Gonzalez, ed.), Chapman and Hall/CRC, 2018.

57 J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle, "Effective hybrid stochastic local search algorithms for biobjective permutation flowshop scheduling," in *International Workshop on Hybrid Metaheuristics* (M. J. Blesa, C. Blum, L. D. Gaspero, A. Roli, M. Sampels, and A. Schaerf, eds.), vol. 5818 of *Lecture Notes in Computer Science*, pp. 100–114, Springer, 2009.

58 H. G. Santos, T. A. M. Toffolo, C. L. T. F. Silva, and G. V. Berghe, "Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem," *ITOR*, vol. 26, no. 2, pp. 707–724, 2019.

59 A. Balint and A. Fröhlich, "Improving stochastic local search for SAT with a new probability distribution," in *Theory and Applications of Satisfiability Testing* (O. Strichman and S. Szeider, eds.), vol. 6175 of *Lecture Notes in Computer Science*, pp. 10–15, Springer, 2010.

60 S. Cai, C. Luo, and K. Su, "CCAnr: A configuration checking based local search solver for non-random satisfiability," in *Theory and Applications of Satisfiability Testing* (M. Heule and S. A. Weaver, eds.), vol. 9340 of *Lecture Notes in Computer Science*, pp. 1–8, Springer, 2015.

61 S. Cai and K. Su, "Configuration checking with aspiration in local search for SAT," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (J. Hoffmann and B. Selman, eds.), AAAI Press, 2012.

62 S. Cai and K. Su, "Local search for boolean satisfiability with configuration checking and subscore," *Artif. Intell.*, vol. 204, pp. 75–98, 2013.

63 S. Cai and K. Su, "Local search with configuration checking for SAT," in *IEEE 23rd International Conference on Tools with Artificial Intelligence*, pp. 59–66, IEEE Computer Society, 2011.

64 O. Gableske and M. Heule, "EagleUP: Solving random 3-SAT using SLS with unit propagation," in *Theory and Applications of Satisfiability Testing* (K. A. Sakallah and L. Simon, eds.), vol. 6695 of *Lecture Notes in Computer Science*, pp. 367–368, Springer, 2011.

65 F. Hutter, D. A. D. Tompkins, and H. H. Hoos, "Scaling and probabilistic smoothing: Efficient dynamic local search for SAT," in *Principles and Practice of Constraint Programming* (P. V. Hentenryck, ed.), vol. 2470 of *Lecture Notes in Computer Science*, pp. 233–248, Springer, 2002.

66 C. M. Li and Y. Li, "Satisfying versus falsifying in local search for satisfiability," in *Theory and Applications of Satisfiability Testing* (A. Cimatti and R. Sebastiani, eds.), vol. 7317 of *Lecture Notes in Computer Science*, pp. 477–478, Springer, 2012.

67 P. Mills and E. P. K. Tsang, "Guided local search for solving SAT and weighted MAX-SAT problems," *J. Autom. Reasoning*, vol. 24, no. 1/2, pp. 205–223, 2000.

68 D. A. McAllester, B. Selman, and H. A. Kautz, "Evidence for invariants in local search," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference* (B. Kuipers and B. L. Webber, eds.), pp. 321–326, AAAI Press / The MIT Press, 1997.

69 D. N. Pham, J. Thornton, C. Gretton, and A. Sattar, "Advances in local search for satisfiability," in *AI 2007: Advances in Artificial Intelligence* (M. A. Orgun and J. Thornton, eds.), vol. 4830 of *Lecture Notes in Computer Science*, pp. 213–222, Springer, 2007.

70 B. Selman, H. J. Levesque, and D. G. Mitchell, "A new method for solving hard satisfiability problems," in *Proceedings of the 10th National Conference on Artificial Intelligence* (W. R. Swartout, ed.), pp. 440–446, AAAI Press / The MIT Press, 1992.

71 B. Selman and H. A. Kautz, "Domain-independent extensions to GSAT: Solving large structured satisfiability problems," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (R. Bajcsy, ed.), pp. 290–295, Morgan Kaufmann, 1993.

72 B. Selman, H. A. Kautz, and B. Cohen, "Noise strategies for improving local search," in *Proceedings of the 12th National Conference on Artificial Intelligence* (B. Hayes-Roth and R. E. Korf, eds.), pp. 337–343, AAAI Press / The MIT Press, 1994.

73 J. Thornton, D. N. Pham, S. Bain, and V. Ferreira Jr., "Additive versus multiplicative clause weighting for SAT," in *Proceedings of the Nineteenth National Conference on Artificial Intelligence* (D. L. McGuinness and G. Ferguson, eds.), pp. 191–196, AAAI Press / The MIT Press, 2004.

74 S. Cai, C. Luo, and H. Zhang, "From decimation to local search and back: A new approach to MaxSAT," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence* (C. Sierra, ed.), pp. 571–577, ijcai.org, 2017.

75 M. Heule, M. Järvisalo, and M. Suda, "Proceedings of SAT Competition 2018 : Solver and benchmark descriptions," Department of Computer Science Series of Publications B, Department of Computer Science, University of Helsinki, 2018.

76 F. Bacchus, M. Järvisalo, and R. Martins, "MaxSAT Evaluation 2018 : Solver and benchmark descriptions," Department of Computer Science Series of Publications B, Department of Computer Science, University of Helsinki, 2018.

77 F. Bacchus, M. Järvisalo, and R. Martins, "MaxSAT Evaluation 2019 : Solver and benchmark descriptions," Department of Computer Science Series of Publications B, Department of Computer Science, University of Helsinki, 2018.

78 I. P. Gent and T. Walsh, "Towards an understanding of hill-climbing procedures for SAT," in *Proceedings of the 11th National Conference on Artificial Intelligence* (R. Fikes and W. G. Lehnert, eds.), pp. 28–33, AAAI Press / The MIT Press, 1993.

79 I. P. Gent and T. Walsh, "Unsatisfied variables in local search," *Hybrid problems, hybrid solutions*, vol. 27, p. 73, 1995.

80 C. H. Papadimitriou, "On selecting a satisfying truth assignment (extended abstract)," in *32nd Annual Symposium on Foundations of Computer Science*, pp. 163–169, IEEE Computer Society, 1991.

81 Y. Shang and B. W. Wah, "A discrete Lagrangian-based global-search method for solving satisfiability problems," *J. Global Optimization*, vol. 12, no. 1, pp. 61–99, 1998.

82 D. Schuurmans, F. Southey, and R. C. Holte, "The exponentiated subgradient algorithm for heuristic boolean programming," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence* (B. Nebel, ed.), pp. 334–341, Morgan Kaufmann, 2001.

83 S. Cai, K. Su, and A. Sattar, "Local search with edge weighting and configuration checking heuristics for minimum vertex cover," *Artif. Intell.*, vol. 175, no. 9-10, pp. 1672–1696, 2011.

84 M. Luo, C. Li, F. Xiao, F. Manyà, and Z. Lü, "An effective learnt clause minimization approach for CDCL SAT solvers," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence* (C. Sierra, ed.), pp. 703–711, ijcai.org, 2017.

85 Y. Jiang, H. Kautz, and B. Selman, "Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT," in *1st International Joint Workshop on Artificial Intelligence and Operations Research*, vol. 20, 1995.

86 B. W. Wah and Y. Shang, "Discrete lagrangian-based search for solving MAX-SAT problems," in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pp. 378–383, Morgan Kaufmann, 1997.

87 S. Cai, "Balance between complexity and quality: Local search for minimum vertex cover in massive graphs," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (Q. Yang and M. J. Wooldridge, eds.), pp. 747–753, AAAI Press, 2015.

88 T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo, "MaxPre: An extended MaxSAT preprocessor," in *Theory and Applications of Satisfiability Testing* (S. Gaspers and T. Walsh, eds.), vol. 10491 of *Lecture Notes in Computer Science*, pp. 449–456, Springer, 2017.

89 M. Järvisalo, A. Biere, and M. Heule, "Blocked clause elimination," in *Tools and Algorithms for the Construction and Analysis of Systems* (J. Esparza and R. Majumdar, eds.), vol. 6015 of *Lecture Notes in Computer Science*, pp. 129–144, Springer, 2010.

90 J. A. Robinson, "A machine-oriented logic based on the resolution principle," *J. ACM*, vol. 12, no. 1, pp. 23–41, 1965.

91 N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Theory and Applications of Satisfiability Testing* (F. Bacchus and T. Walsh, eds.), vol. 3569 of *Lecture Notes in Computer Science*, pp. 61–75, Springer, 2005.

92 M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, 1960.

93 S. Subbarayan and D. K. Pradhan, "Niver: Non-increasing variable elimination resolution for preprocessing SAT instances," in *Theory and Applications of Satisfiability Testing*, 2004.

94 M. Järvisalo and A. Biere, "Reconstructing solutions after blocked clause elimination," in *Theory and Applications of Satisfiability Testing* (O. Strichman and S. Szeider, eds.), vol. 6175 of *Lecture Notes in Computer Science*, pp. 340–345, Springer, 2010.

95 N. Manthey, M. Heule, and A. Biere, "Automated reencoding of boolean formulas," in *Hardware and Software: Verification and Testing* (A. Biere, A. Nahir, and T. E. J. Vos, eds.), vol. 7857 of *Lecture Notes in Computer Science*, pp. 102–117, Springer, 2012.

96 C. M. Li and A. Anbulagan, "Heuristics based on unit propagation for satisfiability problems," in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pp. 366–371, Morgan Kaufmann, 1997.

97 J. W. Freeman, *Improvements to propositional satisfiability search algorithms*. PhD thesis, Citeseer, 1995.

98 D. L. Berre, "Exploiting the real power of unit propagation lookahead," *Electron. Notes Discret. Math.*, vol. 9, pp. 59–80, 2001.

99 I. Lynce and J. P. Marques-Silva, "Probing-based preprocessing techniques for propositional satisfiability," in *15th IEEE International Conference on Tools with Artificial Intelligence*, p. 105, IEEE Computer Society, 2003.

100 M. Heule, M. Järvisalo, and A. Biere, "Covered clause elimination," in *Short papers for 17th International Conference on Logic for Programming, Artificial intelligence, and Reasoning* (A. Voronkov, G. Sutcliffe, M. Baaz, and C. G. Fermüller, eds.), vol. 13 of *EPiC Series in Computing*, pp. 41–46, EasyChair, 2010.

101 M. Heule, M. Järvisalo, and A. Biere, "Clause elimination procedures for CNF formulas," in *Logic for Programming, Artificial Intelligence, and Reasoning* (C. G. Fermüller and A. Voronkov, eds.), vol. 6397 of *Lecture Notes in Computer Science*, pp. 357–371, Springer, 2010.

102 A. V. Gelder, "Toward leaner binary-clause reasoning in a satisfiability solver," *Ann. Math. Artif. Intell.*, vol. 43, no. 1, pp. 239–253, 2005.

103 M. Heule, M. Järvisalo, and A. Biere, "Efficient CNF simplification based on binary implication graphs," in *Theory and Applications of Satisfiability Testing* (K. A. Sakallah and L. Simon, eds.), vol. 6695 of *Lecture Notes in Computer Science*, pp. 201–215, Springer, 2011.

104 A. Billionnet and A. Sutter, "An efficient algorithm for the 3-satisfiability problem," *Oper. Res. Lett.*, vol. 12, no. 1, pp. 29–36, 1992.

105 C. M. Li and Anbulagan, "Look-ahead versus look-back for satisfiability problems," in *Principles and Practice of Constraint Programming* (G. Smolka, ed.), vol. 1330 of *Lecture Notes in Computer Science*, pp. 341–355, Springer, 1997.

106 F. Bacchus, "Enhancing Davis Putnam with extended binary clause reasoning," in *Proceedings of the Eighteenth National Conference on Artificial Intelligence* (R. Dechter, M. J. Kearns, and R. S. Sutton, eds.), pp. 613–619, AAAI Press / The MIT Press, 2002.

107 F. Bacchus and J. Winter, "Effective preprocessing with hyper-resolution and equality reduction," in *Theory and Applications of Satisfiability Testing* (E. Giunchiglia and A. Tacchella, eds.), vol. 2919 of *Lecture Notes in Computer Science*, pp. 341–355, Springer, 2003.

108 W. Wei and B. Selman, "Accelerating random walks," in *Principles and Practice of Constraint Programming - CP 2002* (P. V. Hentenryck, ed.), vol. 2470 of *Lecture Notes in Computer Science*, pp. 216–232, Springer, 2002.

109 J. Berg, P. Saikko, and M. Järvisalo, "Subsumed label elimination for maximum satisfiability," in *Proceedings of the Twenty-second European Conference on Artificial Intelligence* (G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, eds.), vol. 285 of *Frontiers in Artificial Intelligence and Applications*, pp. 630–638, IOS Press, 2016.

110 M. Järvisalo, M. Heule, and A. Biere, "Inprocessing rules," in *International Joint Conference on Automated Reasoning* (B. Gramlich, D. Miller, and U. Sattler, eds.), vol. 7364 of *Lecture Notes in Computer Science*, pp. 355–370, Springer, 2012.

111 J. Berg, P. Saikko, and M. Järvisalo, "Re-using auxiliary variables for MaxSAT preprocessing," in *27th IEEE International Conference on Tools with Artificial Intelligence*, pp. 813–820, IEEE Computer Society, 2015.

112 N. Manthey, "Coprocessor - a standalone SAT preprocessor," in *Applications of Declarative Programming and Knowledge Management* (H. Tompits, S. Abreu, J. Oetsch, J. Pührer, D. Seipel, M. Umeda, and A. Wolf, eds.), vol. 7773 of *Lecture Notes in Computer Science*, pp. 297–304, Springer, 2011.

113 N. Manthey, "Coprocessor 2.0 - A flexible CNF simplifier," in *Theory and Applications of Satisfiability Testing* (A. Cimatti and R. Sebastiani, eds.), vol. 7317 of *Lecture Notes in Computer Science*, pp. 436–441, Springer, 2012.

114 J. Berg and M. Järvisalo, "Impact of SAT-based preprocessing on core-guided MaxSAT solving," in *Principles and Practice of Constraint Programming* (M. Rueher, ed.), vol. 9892 of *Lecture Notes in Computer Science*, pp. 66–85, Springer, 2016.

115 J. Berg, P. Saikko, and M. Järvisalo, "Improving the effectiveness of SAT-based preprocessing for MaxSAT," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (Q. Yang and M. J. Wooldridge, eds.), pp. 239–245, AAAI Press, 2015.

116 R. Martins, V. M. Manquinho, and I. Lynce, "Open-WBO: A modular MaxSAT solver,," in *Theory and Applications of Satisfiability Testing* (C. Sinz and U. Egly, eds.), vol. 8561 of *Lecture Notes in Computer Science*, pp. 438–445, Springer, 2014.

117 S. R. Buss and G. Turán, "Resolution proofs of generalized pigeonhole principles," *Theor. Comput. Sci.*, vol. 62, no. 3, pp. 311–317, 1988.

118 E. Demirovic and P. J. Stuckey, "Techniques inspired by local search for incomplete MaxSAT and the linear algorithm: Varying resolution and solution-guided search," in *Principles and Practice of Constraint Programming* (T. Schiex and S. de Givry, eds.), vol. 11802 of *Lecture Notes in Computer Science*, pp. 177–194, Springer, 2019.

119 R. Martins, V. M. Manquinho, and I. Lynce, "Improving linear search algorithms with model-based approaches for MaxSAT solving," *J. Exp. Theor. Artif. Intell.*, vol. 27, no. 5, pp. 673–701, 2015.

120 M. Janota and J. Marques-Silva, "On the query complexity of selecting minimal sets for monotone predicates," *Artif. Intell.*, vol. 233, pp. 73–83, 2016.

121 "Finnish Grid and Cloud Infrastructure," 2019. urn:nbn:fi:research-infras-2016072533.