Preprocessing in Incomplete MaxSAT Solving

Marcus Leivo and **Jeremias Berg** and **Matti Järvisalo**[1]

Abstract. Motivated by the success of preprocessing in Boolean satisfiability (SAT) solving, development and analysis of preprocessing in maximum satisfiability (MaxSAT)—the optimization extension of SAT—has received noticeable attention recently. The correctness of preprocessing techniques for MaxSAT is standardly established by arguing that optimal solutions are maintained. However, the effects of preprocessing on the relative perceived costs of *non-optimal* solutions has not been considered, despite the fact that one of the most recent directions in MaxSAT research is developing *incomplete* solvers, i.e., solvers that are designed to provide good (but not necessarily optimal) solutions fast. In this paper, we bridge this gap by showing that employing central preprocessing techniques misleads MaxSAT solvers in terms of their interpretation of the costs of non-optimal solutions seen during search. This issue impacts both complete and incomplete solvers and the effects can be shown to be present also in practice with different types of MaxSAT solvers. Furthermore, we propose ideas for circumventing these negative effects in the context of stochastic local search algorithms for MaxSAT.

1 INTRODUCTION

Extending Boolean satisfiability (SAT) to optimization, maximum satisfiability (MaxSAT) is today a viable Boolean optimization paradigm for various NP-hard optimization problems arising from AI and industrial applications. Various algorithmic advances, based on iterative applications of SAT solvers, have brought on this success.

A main focus in algorithmic MaxSAT research—especially until recently—has been on developing new search techniques to obtain further improvements in the scalability and robustness of state-of-the-art *complete* MaxSAT solvers, i.e., solvers that can provide provably optimal solutions. A recent new direction, complementing complete solvers, is the development of *incomplete* solvers which ideally can provide good (albeit not optimal, or provably optimal) solutions fast. Incomplete solving is motivated by the fact that in many real-world application settings, obtaining relatively good solutions fast is beneficial. Since 2012, the MaxSAT Evaluations [3, 5] have provided further incentives for developing incomplete MaxSAT solving techniques via a special track for incomplete solvers.

In addition to advances in search techniques for MaxSAT solving [7, 2, 20, 35, 9, 37], there is noticeable recent progress in solver-independent preprocessing techniques for MaxSAT [6, 8, 11, 13, 10, 28, 12], motivated by the success of preprocessing as a central part of the modern SAT solving workflow. However, while MaxSAT preprocessing has been shown to be beneficial, currently preprocessing appears to bring more modest performance improvements in the context of MaxSAT solving compared to SAT solving.

[1] HIIT, Department of Computer Science, University of Helsinki, Finland, email: firstname.lastname@helsinki.fi

In this work, we strive for a deeper understanding about the impact of preprocessing on the internal behavior of MaxSAT solvers, starting from the facts that (i) both complete and incomplete MaxSAT solvers during their search encounter not only optimal but also non-optimal solutions, and that (ii) the correctness analysis of various MaxSAT preprocessing disregards non-optimal solutions, centering around proving that the cost of optimal solutions is maintained in a controlled way.

In this paper we will bridge these two seemingly somewhat unrelated facts together. In particular, we will show that preprocessing can, both in theory and in practice, result in various types of state-of-the-art MaxSAT solving algorithms misinterpreting the costs of non-optimal solutions to be higher than they actually are. This is problematic in particular since solvers make use of the information provided by the costs of non-optimal solutions encountered during search in various ways, depending on the algorithmic approach: in the core-guided approach [37, 38, 40] for bounding and hardening via stratification [1], in the implicit hitting set approach [18, 19, 17, 41] for bounding and reduced cost fixing [4], in the model-improving approach [36, 29, 39] for deciding the next bound to check, and in local search approaches for heuristic guidance [16, 14, 15, 20, 22, 30, 33, 34], etc. Through a more detailed analysis, we show that for what we refer to as *locally minimal solutions* the issue can be avoided in the case of most preprocessing techniques, but this requires changes to MaxSAT solvers.

More specifically, we show that most central preprocessing techniques (as we consider here, many of which are liftings of SAT preprocessing techniques) preserve locally minimal solutions in the sense that the cost of a locally minimal solution equals that of a corresponding solution obtained through standard linear-time solution reconstruction techniques for the original MaxSAT instance at hand. Hence such techniques—including e.g. bounded variable elimination [21], self-subsuming resolution and subsumed label elimination [13, 28]—applied together with a MaxSAT solver that is guaranteed to consider only locally minimal solutions—are not problematic in terms of misinterpreting costs of non-optimal solutions. However, it turns out that not all preprocessing techniques have such a property; in particular, we identify that blocked clause elimination [26] results in misinterpretations of the costs of non-optimal solutions even in the case of locally minimal solutions.

Towards developing MaxSAT algorithms which are guaranteed to work on locally minimal solutions, we propose an adaptation of stochastic local search (SLS) for MaxSAT which—in contrast to the current state-of-the-art SLS algorithms for MaxSAT—has this property. Empirical results suggest that searching over locally minimal solutions in this context can yield practical performance improvements.

After preliminaries on MaxSAT and preprocessing (Sect. 2), we analyze the impact of preprocessing on the perceived costs of non-

optimal solutions as seen by MaxSAT solvers during search (Sect. 3). Finally, we propose an SLS approach that is guaranteed to search over locally minimal solutions and empirically evaluate its performance (Sect. 4).

2 PRELIMINARIES

We start with preliminaries on MaxSAT and preprocessing.

2.1 Maximum Satisfiability

A literal is a Boolean variable x or its negation $\neg x$. A clause is a disjunction of literals (often viewed as the set of its literals), with satisfaction defined in the standard way. We will interchangeably consider a truth assignment τ as either a mapping from literals to $\{0,1\}$ or the set of literals assigned to 1 by τ. An assignment satisfies a set F of clauses (a CNF formula) if it satisfies all $C \in F$. We use $\textsc{Var}(F)$ and $\textsc{Lit}(F)$ to denote the variables and literals, respectively, of the clauses in F. A clause is a tautology if it contains both literals over some variable.

For $L \subseteq \textsc{Lit}(F)$, the set $\textsc{Cl}(F, L) \subseteq F$ consists of the clauses of F which contain a literal in L. The set $\neg L$ contains the negation of each literal in L. The restriction $F\big|_\tau$ of a set of clauses wrt a truth assignment (i.e. a set of literals) τ is obtained by removing all clauses $C \in F$ satisfied by τ and all literals falsified by τ from other clauses.

A standard way of defining the maximum satisfiability problem is as follows. A MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$ consists of the sets F_h and F_s of hard and soft clauses, respectively, and a function $w\colon F_s \to \mathbb{N}$ that assigns a positive cost to each soft clause. Any truth assignment τ that satisfies F_h is a solution to $\mathcal{F}$. The cost $\textsc{Cost}(\mathcal{F}, \tau)$ of τ is the sum of the weights of the soft clauses it falsifies, i.e., $\textsc{Cost}(\mathcal{F}, \tau) = \sum_{C \in F_s} (1 - \tau(C)) \cdot w(C)$. A solution τ is optimal if $\textsc{Cost}(\mathcal{F}, \tau) \leq \textsc{Cost}(\mathcal{F}, \tau')$ holds for all solutions τ' to $\mathcal{F}$. The (weighted partial) MaxSAT problem asks to compute an optimal solution to a given instance $\mathcal{F}$.

A central concept in MaxSAT solving is that of a correction set (CS). A subset $S \subseteq F_s$ is a correction set if $F_h \cup (F_s \setminus S)$ is satisfiable. A correction set S is minimal (an MCS) if no $S' \subset S$ is a CS. We denote the set of all MCSes of a MaxSAT instance $\mathcal{F}$ by $\textsc{MCS}(\mathcal{F})$. Each solution τ of $\mathcal{F}$ corresponds to a CS $M^\tau = \{C \in F_s \mid \tau(C) = 0\}$ of $\mathcal{F}$. Conversely, for each correction set M of $\mathcal{F}$, there is a corresponding solution τ that satisfies $F_h \cup (F_s \setminus M)$. A solution τ is minimal if $M^\tau \in \textsc{MCS}(\mathcal{F})$. A CS M is optimal if it corresponds to an optimal solution. Since $\textsc{Cost}(\mathcal{F}, \tau) = \sum_{C \in M^\tau} w(C)$, every optimal CS is minimal and MaxSAT can be seen as the problem of computing an optimal MCS of $\mathcal{F}$.

Example 1 *Let $\mathcal{F} = (F_h, F_s, w)$ be a MaxSAT instance with $F_h = \{(y \vee x)\}$, $F_s = \{(x), (\neg x), (\neg y)\}$ with $w((x)) = w((\neg x)) = 1$ and $w((\neg y)) = 2$. For this instance, $\textsc{MCS}(\mathcal{F}) = \{\{(\neg x)\}, \{(x), (\neg y)\}\}$. The solution $\tau = \{\neg y, x\}$ is optimal for $\mathcal{F}$. It has $\textsc{Cost}(\mathcal{F}, \tau) = 1$ and corresponds to the optimal correction set $M^\tau = \{(\neg x)\} \in \textsc{MCS}(\mathcal{F})$. The solution $\tau_2 = \{\neg x, y\}$ is minimal, but not optimal since $M^{\tau_2} = \{(x), (\neg y)\} \in \textsc{MCS}(\mathcal{F})$ but $\textsc{Cost}(\mathcal{F}, \tau_2) = 3$. Finally, the solution $\tau_3 = \{x, y\}$ is not minimal as $M^{\tau_3} = \{(\neg x), (\neg y)\} \supset M^\tau$.*

2.2 Applying SAT Preprocessing in MaxSAT

We will focus on recently proposed MaxSAT-liftings of central SAT preprocessing techniques. In particular, we will consider the SAT preprocessing techniques of bounded variable elimination (BVE) [21], blocked clause elimination (BCE) [26], subsumption elimination (SE) and self-subsuming resolution (SSR).

In practice, the application of SAT preprocessing in the context of MaxSAT is implemented as follows. Let $\mathcal{F} = (F_h, F_s, w)$ be the MaxSAT instance to be preprocessed.

1. Each soft clause $C \in F_s$ is replaced with the clause $C \vee x_C$ where $x_C \notin \textsc{Var}(\mathcal{F})$, i.e., each soft clause is extended with a unique fresh "soft" variable[2] to obtain the set of clauses $F'_s = \{C \vee x_C \mid C \in F_s\}$. Let $\mathcal{S}(\mathcal{F})$ denote the set of soft variables added.
2. Apply preprocessing on the clauses in $F_h \cup F'_s$ with the restriction that the soft variables are not resolved upon, resulting in a set of clauses F^P.
3. The preprocessed MaxSAT instance $\mathcal{P}(\mathcal{F})$ is then $(F^P, \{(\neg x_C) \mid x_C \in \mathcal{S}(\mathcal{F})\}, w^P)$, i.e., has F^P as its hard clauses, and a unit soft clause $(\neg x_C)$ with weight $w^P((\neg x_C)) = w(C)$ for each soft variable x_C added to $\mathcal{F}$ in the first step.

Note that the weight of each original soft clause $C \in F_s$ is maintained in the corresponding soft clause $(\neg x_C)$ in $\mathcal{P}(\mathcal{F})$. However, while there is a one-to-one correspondence between C and its soft variable x_C before preprocessing, after preprocessing this is lost: a hard clause in $\mathcal{P}(\mathcal{F})$ may contain multiple soft variables, and a soft variable may occur in multiple hard clauses in $\mathcal{P}(\mathcal{F})$.

Example 2 *Consider the instance $\mathcal{F} = (F_h, F_s, w)$ from Example 1. Extending each soft clause $C \in F_s$ with a soft variable gives $F'_s = \{(x \vee l_1), (\neg x \vee l_2), (\neg y \vee l_3)\}$. Applying bounded variable elimination preprocessing to eliminate x from $F_h \cup F'_s = \{(y \vee x), (x \vee l_1), (\neg x \vee l_2), (\neg y \vee l_3)\}$ results in the clause set $F^p = \{(y \vee l_2), (l_1 \vee l_2), (\neg y \vee l_3)\}$. The preprocessed instance $\mathcal{P}(\mathcal{F})$ is then $\mathcal{P}(\mathcal{F}) = (F^p, \{(\neg l_i) \mid i = 1..3\}, w^P)$ with $w^P((\neg l_1)) = w^P((\neg l_2)) = 1$ and $w^P((\neg l_3)) = 2$.*

2.3 Literal-Centric View on MaxSAT

As we have just seen, the way SAT preprocessing can be applied on MaxSAT yields MaxSAT instances with a specific structure: all soft clauses are unit clauses (representing the weights of the original soft clauses), and the hard clauses arbitrarily contain these "soft" variables, to which weights are associated, together with other "non-soft" variables without weights (or, equivalently, with weight 0). By this observation, in analogy with e.g. how integer programs are standardly defined, a preprocessed MaxSAT instance $\mathcal{P}(\mathcal{F})$ can be viewed as the problem of minimizing $\sum_{C \in F_s} w(C) \cdot \tau(x_C)$ subject to the clauses in F^P viewed as constraints.

This motivates the following alternative definition for MaxSAT which we will refer to as "literal-centric MaxSAT" (in contrast to the "standard MaxSAT" definition of Section 2.1). A literal-centric MaxSAT instance $\mathcal{F}$ consists of a set of clauses $\textsc{Clauses}(\mathcal{F})$ and a function $w^{\mathcal{F}}\colon \textsc{Var}(\textsc{Clauses}(\mathcal{F})) \to \mathbb{N} \cup \{0\}$ assigning a non-negative weight to each variable in $\textsc{Clauses}(\mathcal{F})$. We will use $\mathcal{F}$ and $\textsc{Clauses}(\mathcal{F})$ interchangeably and drop the superscript from w when clear from context. A variable $x \in \textsc{Var}(\mathcal{F})$ is *soft* if $w(x) > 0$. The set $\mathcal{S}(\mathcal{F})$ contains the soft variables of $\mathcal{F}$. We will use $x \notin \mathcal{S}(\mathcal{F})$ as shorthand for $x \in \textsc{Var}(\mathcal{F}) \setminus \mathcal{S}(\mathcal{F})$. An MCS of $\mathcal{F}$ is a subset $S \subseteq \mathcal{S}(\mathcal{F})$ for which $\mathcal{F}\big|_{\neg(\mathcal{S}(\mathcal{F}) \setminus S)}$ is satisfiable and $\mathcal{F}\big|_{\neg(\mathcal{S}(\mathcal{F}) \setminus S')}$ is unsatisfiable for all $S' \subset S$. A truth assignment τ is a solution to $\mathcal{F}$ if $\tau(\mathcal{F}) = 1$. Overloading notation, we denote by $\mathcal{S}(\tau)$ the set of

[2] These may also be called blocking, assumption or label variables.

soft variables of $\mathcal{F}$ assigned to true in τ. A solution τ is minimal if $\mathcal{S}(\tau) \in \text{MCS}(\mathcal{F})$.

Example 3 *Consider the instance $\mathcal{F}$ from Example 1. The literal-centric view $\mathcal{F}^E$ of $\mathcal{F}$ has* $\text{CLAUSES}(\mathcal{F}) = \{(y \vee x), (x \vee l_1), (\neg x \vee l_2), (\neg y \vee l_3)\}$, $\mathcal{S}(\mathcal{F}) = \{l_1, l_2, l_3\}$ *with* $w^{\mathcal{F}}(l_1) = w^{\mathcal{F}}(l_2) = 1$ *and* $w^{\mathcal{F}}(l_3) = 2$. *The set $M = \{l_2\}$ is an MCS of $\mathcal{F}$ as* $\mathcal{F}\big|_{\neg(\mathcal{S}(\mathcal{F}) \setminus M)} = \{(y \vee x), (x), (\neg x \vee l_2), (\neg y)\}$ *is satisfied by* $\tau = \{\neg y, x, l_2\}$ *and* $\mathcal{F}\big|_{\neg(\mathcal{S}(\mathcal{F}) \setminus \emptyset)} = \{(y \vee x), (x), (\neg x), (\neg y)\}$ *is unsatisfiable. The solution τ has $\mathcal{S}(\tau) = \{l_2\} \in \text{MCS}(\mathcal{F})$, so it is minimal.*

2.4 MaxSAT Preprocessing Techniques

We will next overview the preprocessing techniques we focus on. For this, let $\mathcal{F}$ be a (literal-centric) MaxSAT instance. The resolvent $C \bowtie_x D$ of two clauses $C = (C^1 \vee x) \in \mathcal{F}$ and $D = (D^1 \vee \neg x) \in \mathcal{F}$ is $C^1 \vee D^1$. Resolution is lifted to sets of clauses $S_x \subset \mathcal{F}$ ($S_{\neg x} \subset \mathcal{F}$) all containing the literal x ($\neg x$) by $S_x \bowtie_x S_{\neg x} = \{C \bowtie_x D \mid C \in S_x, D \in S_{\neg x}\}$. We will focus on the following preprocessing techniques [6, 10, 13, 28].

Bounded variable elimination **(BVE)** eliminates a variable $x \notin \mathcal{S}(\mathcal{F})$ from $\mathcal{F}$ by substituting all clauses containing the variable x with the pairwise resolvents of those clauses. More specifically, $\text{BVE}(\mathcal{F}) = (\mathcal{F} \setminus (\text{CL}(\mathcal{F}, x) \cup \text{CL}(\mathcal{F}, \neg x))) \cup (\text{CL}(\mathcal{F}, x) \bowtie_x \text{CL}(\mathcal{F}, \neg x))$.[3]

Blocked clause elimination **(BCE)** A clause C is blocked on a literal l if $C \bowtie_l D$ is a tautology for all $D \in \text{CL}(\mathcal{F}, \neg l)$. BCE eliminates a clause $C \in \mathcal{F}$ that is blocked on a literal $l \notin \mathcal{S}(\mathcal{F})$, i.e., $\text{BCE}(\mathcal{F}) = \mathcal{F} \setminus \{C\}$.

Subsumption elimination **(SE)** A clause $C \in \mathcal{F}$ is subsumed if there is another clause $D \in \mathcal{F}$ for which $D \subseteq C$. SE eliminates subsumed clauses from $\mathcal{F}$, i.e., $\text{SE}(\mathcal{F}) = \mathcal{F} \setminus \{C\}$.

Self-subsuming resolution **(SSR)** removes literals from clauses subsumed by resolvents. More precisely, let $l \notin \mathcal{S}(\mathcal{F})$ and assume $\mathcal{F}$ contains two clauses $C = C' \vee l$ and $D = D' \vee \neg l$ for which $(C \bowtie_l D) \subseteq D$. Then $\text{SSR}(\mathcal{F}) = \mathcal{F} \setminus \{D\} \cup \{C \bowtie_l D\}$.

Group-subsumed label elimination **(GSLE)** is a MaxSAT-specific technique. A soft variable $x \in \mathcal{S}(\mathcal{F})$ is subsumed by a group $S \subseteq (\mathcal{S}(\mathcal{F}) \setminus \{x\})$ if i) $\text{CL}(\mathcal{F}, x) \subseteq \text{CL}(\mathcal{F}, S)$ and ii) $w^{\mathcal{F}}(x) \geq \sum_{y \in S} w^{\mathcal{F}}(y)$. GSLE removes soft variables that are subsumed. More precisely, $\text{GSLE}(\mathcal{F}) = \mathcal{F}\big|_{\{\neg x\}}$.

2.5 Solution Reconstruction

An important detail in applying preprocessing is that some of the most central preprocessing techniques require applying *solution reconstruction* for obtaining a solution to the original instance from a solution to the preprocessed instance. This is due to the fact that preprocessing techniques such as BVE and BCE preserve (on the level of SAT) only equisatisfiability, not logical equivalence; in particular, the preprocessed instances may have solutions which are not solutions of the original instance. However, there is a general linear-time solution reconstruction technique [27, 25] that can be applied in conjunction with essentially any preprocessing technique. Solution reconstruction will be important for some of the main observations made in this work. Hence we shortly define the solution reconstruction steps for each of the preprocessing techniques considered.

[3] In practice, BVE is not allowed to increase the number of clauses. This detail does not affect the theoretical analysis of this work.

Assume that we have a solution τ to a preprocessed instance $\mathcal{P}(\mathcal{F})$. Solution reconstruction works by modifying τ iteratively in the inverse order of the preprocessing steps made to obtain to obtain a solution to the original instance $\mathcal{F}$. We will denote the thereby reconstructed solution by $\text{REC}(\tau)$. The reconstruction works as follows for each step of the individual preprocessing techniques.

BVE: Let $x \notin \mathcal{S}(\mathcal{F})$ be the eliminated variable. $\text{REC}(\tau)$ is $\tau \cup \{x\}$ if $\tau(\text{CL}(\mathcal{F}, x)) = 0$, and $\tau \cup \{\neg x\}$ otherwise.
BCE: Let C be blocked on $l \notin \mathcal{S}(\mathcal{F})$. $\text{REC}(\tau)$ is $(\tau \setminus \{\neg l\}) \cup \{l\}$ if $\tau(C) = 0$ and τ otherwise.
SE and SSR: $\text{REC}(\tau)$ is τ.
GSLE: Let $l \in \mathcal{S}(\mathcal{F})$ be subsumed. $\text{REC}(\tau)$ is $\tau \cup \{\neg l\}$.

Example 4 *Consider the instance $\mathcal{F}^E$ from Example 3. BVE can be applied to eliminate the variable $x \notin \mathcal{S}(\mathcal{F})$ in order to obtain the (literal-centric) instance* $\text{BVE}(\mathcal{F}^E) = \{(y \vee l_2), (l_1 \vee l_2), (\neg y \vee l_3)\}$ *with* $\mathcal{S}(\text{BVE}(\mathcal{F}^E)) = \{l_1, l_2, l_3\}$. *The solution* $\tau = \{\neg y, l_1, l_2, \neg l_3\}$ *to* $\text{BVE}(\mathcal{F})$ *does not satisfy* $(x \vee y) \in \text{CL}(\mathcal{F}^E, x)$ *so* $\text{REC}(\tau) = \{\neg y, l_1, l_2, \neg l_3, x\}$.

3 PREPROCESSING AND SOLUTION COST

We aim to better understand how applying preprocessing before MaxSAT search influences the interpretation of the costs of solutions seen during search.

To date, correctness arguments for different preprocessing techniques for MaxSAT [6, 10] have focused on establishing a well-defined connection between the costs of optimal solutions before and after preprocessing. However, both complete and incomplete solvers use information from *non-optimal* solutions seen during search to expedite the search process, using the costs of the non-optimal solutions e.g. for bounding (as in the case of complete solving in the style of the implicit hitting set approach to MaxSAT [41, 4, 18, 19, 17]) and the so-called hardening rule [1] applied with stratification in core-guided MaxSAT solvers [40, 37, 38], heuristic guidance (as in the case of stochastic local search for MaxSAT [14, 33, 15]) and linear-search (model improving) algorithms whose search is guided by the cost of solutions found [20, 36, 29].

In this section, we will show that, both in theory and in practice, applying preprocessing before MaxSAT search can in fact have a drastic effect on the perceived cost of non-optimal solutions seen during search. In particular, we will show that subtleties in how preprocessing alters MaxSAT instances result in MaxSAT algorithms making overestimations the costs of non-optimal solutions, thereby potentially weakening the overall performance unnecessarily when preprocessing is applied. After the analysis, we will in Section 4 propose a solution for dealing with this issue in practice in the context of stochastic local search for MaxSAT.

3.1 Apparent vs Actual Cost

Central to our analysis is the notion of *apparent cost* of (non-optimal) solutions, reflecting how the cost of solutions is perceived—potentially incorrectly—by MaxSAT algorithms after preprocessing.

Definition 1 *Let $\mathcal{F}$ be a (standard) MaxSAT instance. The apparent cost* $\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau)$ *of a solution τ to a preprocessed MaxSAT instance $\mathcal{P}(\mathcal{F})$ is* $\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau) = \sum_{x \in \mathcal{S}(\mathcal{F})} \tau(x) w(x)$.

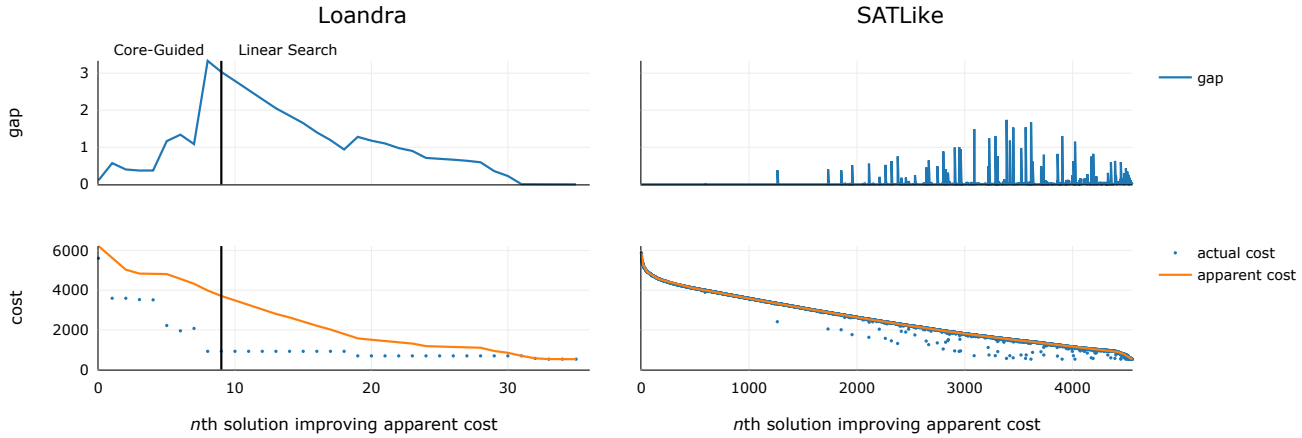

Figure 1. Difference between actual and apparent costs of solutions: Loandra (left) and SATLike (right) on instance ram_k3_n12.ra1.wcnf.

First, recall that minimal solutions correspond to minimal corrections sets. In earlier work [6, 13], establishing the correctness of preprocessing in terms of optimal solutions, it has been shown that the apparent cost of any *minimal* solution τ to a preprocessed MaxSAT equals the actual cost of the solution $\text{REC}(\tau)$ obtained via solution reconstruction to the original instance.

Proposition 1 *([6, 13]) Let $\mathcal{F}$ be a (standard) MaxSAT instance, $\mathcal{P}(\mathcal{F})$ a preprocessed instance and τ a minimal solution to $\mathcal{P}(\mathcal{F})$. Then* $\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau) = \text{COST}(\mathcal{F}, \text{REC}(\tau))$.

However, this does *not* hold for solutions to preprocessed instances in general. In particular, the following example demonstrates that Proposition 1 does not hold for non-minimal solutions.

Example 5 *Consider the instances $\mathcal{F}$, $\mathcal{F}^E$ and $\text{BVE}(\mathcal{F}^E)$ as well as the solutions τ and $\text{REC}(\tau)$ discussed in Examples 1, 3 and 4. For these* $\text{APPAR-COST}(\text{BVE}(\mathcal{F}^E), \tau) = \text{APPAR-COST}(\mathcal{F}^E, \text{REC}(\tau)) = 2 \neq 1 = \text{COST}(\mathcal{F}, \text{REC}(\tau))$.

In other words, preprocessing does not preserve the costs of the non-optimal solutions of MaxSAT instances in general. In order to understand why this is an issue in MaxSAT solving in practice, note that solvers in general straightforwardly use the notion of apparent cost for interpreting the costs of solutions of the preprocessed instance encountered during search, either explicitly or implicitly via unit soft clauses containing the negations of soft variables. This implies that they may incorrectly evaluate the costs of solutions encountered when preprocessing is applied prior to search.

This observation holds not only in theory, but is also witnessed in practice. We demonstrate this using two recent incomplete MaxSAT solvers, the SAT-based solver Loandra [7] based on combining core-guided and linear-search algorithms, and the stochastic local search solver SATLike [30]. These solvers were the two best-performing incomplete solvers in the 2019 MaxSAT evaluation [5]. Figure 1 shows the apparent costs of solutions as perceived during search by Loandra and SATLike on a MaxSAT instance from MaxSAT Evaluation 2019 after first preprocessing the instance with the MaxPre preprocessor [28], as well as the *actual* cost of the same solutions. Specifically, for each solution τ_n that improved the currently best found solution, we computed $\text{REC}(\tau_n)$ and recorded both the apparent cost $\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau_n)$ and the actual cost $\text{COST}(\mathcal{F}, \text{REC}(\tau_n))$. Furthermore, the *gap* between the apparent and actual costs as shown in the upper plots of Figure 1 is $(\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau)/\text{COST}(\mathcal{F}, \text{REC}(\tau))) - 1$.

We can observe that preprocessing results in the solvers making drastic overapproximations of the costs of solutions seen during search, thereby potentially significantly slowing down the search by being misled into attempting to improve the apparent cost instead of the actual cost. For example, the 10th solution τ_{10} found by Loandra had $\text{COST}(\mathcal{F}, \text{REC}(\tau_{10})) = 919$ and $\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau_{10}) = 3982$. The following 10 solutions found, i.e., all τ_n for $n = 11..20$ had $\text{COST}(\mathcal{F}, \text{REC}(\tau_n)) = 919$. However, Loandra, operating with $\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau_n)$, still considered them improvements as $\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau_n) > \text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau_{n+1})$ holds for all $n = 10..20$. Even worse, we also found examples of both Loandra and SATLike improving the apparent cost of a solution while worsening the actual cost. For example, the 8th solution τ_8 found by Loandra had $\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau_8) = 4572$ and $\text{COST}(\mathcal{F}, \text{REC}(\tau_8)) = 1953$. The next solution τ_9 considered an improvement by Loandra had $\text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau_9) = 4324 < \text{APPAR-COST}(\mathcal{P}(\mathcal{F}), \tau_8)$ but $\text{COST}(\mathcal{F}, \text{REC}(\tau_9)) = 2073 > \text{COST}(\mathcal{F}, \text{REC}(\tau_8))$.

Notice that, as shown in Figure 1, the apparent and actual costs differ both within the core-guided and linear-search phases of Loandra, which suggests that similar observations can be made for other solvers implementing core-guided or linear-search types of algorithms, including complete solvers. Furthermore, as another central MaxSAT solving paradigm, this observation also extends to implicit hitting set solvers which make use of the costs of optimal solutions encountered during search for e.g. termination and bounds-based fixing of variables [4, 18, 41].

We hypothesize that this issue at least partially explains the reported modest impact of preprocessing in the context of MaxSAT solving [8, 6, 11] (in particular when compared to the impact of preprocessing in the context of SAT solving).

Evidently, to fully take advantage of preprocessing for speeding up MaxSAT solving, the issue of misinterpreting the costs of solutions during search needs to be rectified. By Proposition 1, the apparent cost of a minimal solution equals its actual cost. Hence a hypothetical solver that only computes minimal solutions will always evaluate the costs of solutions it sees correctly; however, ensuring that a solution is minimal is computationally hard, as this requires proving that the solution corresponds to an MCS, which in itself is NP-hard [24]

(although the underlying hitting set problem may be of small size in practice). Alternatively, one may rectify the issue by applying solution reconstruction at each iteration of search. However, while solution reconstruction is linear-time in the number of preprocessing steps performed, this can cause a significant overhead as modern MaxSAT solvers, especially stochastic local search solvers, typically find a significant number of solutions during search.

A natural question to ask is if there are other properties of solutions, more relaxed than minimal solutions, for which apparent cost equals actual cost in general. In the following we will show that what we refer to as *locally minimal solutions* characterize such a larger set of solutions for various central preprocessing techniques.

3.2 Locally Minimal Solutions

Example 5 provides simplistic intuition on locally minimal solutions. The solution τ satisfies the clause $(l_1 \vee l_2)$ due to both of the soft variables l_1 and l_2 being set to true, thus unnecessarily incurring apparent cost on both. Similarly, the solution $\mathrm{REC}(\tau)$ satisfies the clause $(x \vee l_1) \in \mathcal{F}^E$ both on x and l_1, incurring apparent cost on l_1 unnecessarily. In other words, the difference between $\textsc{appar-cost}(\mathcal{F}^E, \mathrm{REC}(\tau)) = \textsc{appar-cost}(\mathcal{P}(\mathcal{F}^E), \tau) = 2$ and $\textsc{Cost}(\mathcal{F}, \mathrm{REC}(\tau)) = 1$ is due to soft variables being set to true unnecessarily. Locally minimal solutions are those in which no soft variables are assigned unnecessarily to true, i.e., under which no soft variable can be flipped from true to false while keeping the assigned values to other soft variables and maintaining satisfiability.

Definition 2 *Let $\mathcal{F}$ be a (literal-centric) MaxSAT instance. A solution τ to $\mathcal{F}$ is locally minimal if $(\tau \setminus S) \cup (\neg S)$ falsifies $\mathcal{F}$ for all non empty $S \subset \mathcal{S}(\tau)$.*

In particular, consider a (standard) MaxSAT instance $\mathcal{F}$ and let $\mathcal{F}^E$ be the (literal-centric) instance obtained by adding soft variables to the soft clauses as described in Section 2.2. We have that $\textsc{appar-cost}(\mathcal{F}^E, \tau) = \textsc{Cost}(\mathcal{F}, \tau)$ holds for any locally minimal solution τ to $\mathcal{F}^E$. Importantly, this also holds for applications of preprocessing techniques for which $\mathrm{REC}(\tau)$ is a locally minimal solution to $\mathcal{F}^E$ whenever τ is a locally minimal solution to $\mathcal{P}(\mathcal{F})$.

Definition 3 *A preprocessing technique $\mathcal{P}$ preserves locally minimal solutions if the following holds for any (standard) MaxSAT instance $\mathcal{F}$: If τ is a locally minimal solution to $\mathcal{P}(\mathcal{F})$, then $\mathrm{REC}(\tau)$ is a locally minimal solution to $\mathcal{F}^E$, the literal-centric instance obtained by adding soft variables to the soft clauses of $\mathcal{F}$.*

In particular, applying a preprocessing technique that preserves locally minimal solutions in conjunction with a MaxSAT solver guaranteed to only compute locally minimal solutions during search, the solver will perceive correctly the costs of all solutions encountered during search.

Proposition 2 *The following holds for any preprocessing technique $\mathcal{P}$ that preserves locally minimal solutions. Given a (standard) MaxSAT instance $\mathcal{F}$ and a locally minimal solution τ to $\mathcal{P}(\mathcal{F})$, we have that $\textsc{Cost}(\mathcal{F}, \mathrm{REC}(\tau)) = \textsc{appar-cost}(\mathcal{P}(\mathcal{F}), \tau)$.*

This gives rise to two questions: (i) Which preprocessing techniques preserve locally minimal solutions? (ii) How to guarantee that a MaxSAT solver computes only locally minimal solutions during search? As an answer to (i), we will next show that all of the considered preprocessing techniques—except for BCE—in fact preserve locally minimal solutions. Towards a solution to (ii) in the context of stochastic local search for MaxSAT, we will in Section 4 describe an SLS approach to MaxSAT which is guaranteed to consider only locally minimal solutions.

Lemma 1 *Let τ be a locally minimal solution to an instance $\mathcal{F}$. For each $l \in \mathcal{S}(\tau)$ there is a clause $(C \vee l) \in \mathcal{F}$ such that $\tau(C) = 0$.*

Proof. If not, then $(\tau \setminus \{l\})$ is also a solution to $\mathcal{F}$. ■

Proposition 3 *$\mathcal{P}$ preserves locally minimal solutions for each $\mathcal{P} \in \{\mathrm{BVE}, \mathrm{SE}, \mathrm{SSR}, \mathrm{GSLE}\}$.*

Proof. Let $\mathcal{F}$ be a (literal centric) MaxSAT instance and τ a locally minimal solution to $\mathcal{P}(\mathcal{F})$. The statement follows trivially if $\mathcal{S}(\tau) = \emptyset$. Consider hence any $l \in \mathcal{S}(\tau)$ and apply Lemma 1 to obtain a clause $(C \vee l) \in \mathcal{P}(\mathcal{F})$ for which $\tau(C) = 0$. We prove that $\mathrm{REC}(\tau)$ is a locally minimal solution to $\mathcal{F}$ by showing that $\mathrm{REC}(\tau) \setminus \{l\}$ is not a solution to $\mathcal{F}$. If $(C \vee l) \in \mathcal{F}$, the statement follows from $\tau \subseteq \mathrm{REC}(\tau)$. Otherwise, i.e., if $(C \vee l) \notin \mathcal{F}$, the specific argument depends on $\mathcal{P}$.

- For BVE, $(C \vee l) = D \vee E$ for some $(D \vee x) \in \mathcal{F}$ and $(E \vee \neg x) \in \mathcal{F}$. Assume $\mathrm{REC}(\tau) = \tau \cup \{x\}$ (the other case is symmetric). Then $\mathrm{REC}(\tau)(D \vee x) = 1$. However, as E is satisfied by τ at most on l, it is not satisfied by $\tau \setminus \{l\}$ and thus not by $\mathrm{REC}(\tau) \setminus \{l\}$ either.
- For SSR, $(C \vee l) = (D \bowtie_x E)$ for two clauses $(D \vee x)$ and $(E \vee \neg x)$ of $\mathcal{F}$, where $(D \bowtie_x E) \subset D$. As $\mathrm{REC}(\tau) = \tau$ satisfies $(D \bowtie_x E)$ only on l, it follows that $\mathrm{REC}(\tau) \setminus \{l\}$ falsifies either $(E \vee \neg x)$ or $(D \vee x)$.
- For GSLE, $(C \vee l_2 \vee l) \in \mathcal{F}$ for a subsumed soft variable $l_2 \in \mathcal{S}(\mathcal{F})$. Now $\mathrm{REC}(\tau) = \tau \cup \{\neg l_2\}$ and as such $\mathrm{REC}(\tau) \setminus \{l\}$ does not satisfy $(C \vee l_2 \vee l)$ as $\tau(C) = 0$.
- For SE, the assumption $(C \vee l) \notin \mathcal{F}$ does not apply. ■

Proposition 3 implies that, used in conjunction with a MaxSAT solver that is guaranteed to encounter only locally minimal solutions during search, all of the preprocessing techniques BVE, SE, SSR, GSLE can be applied without fear of overapproximating the costs of solutions.

Theorem 1 *For any (standard) MaxSAT instance $\mathcal{F}$ and combination $\mathcal{P}$ of preprocessing techniques among* BVE*,* SE*,* SSR*, and* GSLE*, we have $\textsc{appar-cost}(\mathcal{P}(\mathcal{F}), \tau) = \textsc{Cost}(\mathcal{F}, \mathrm{REC}(\tau))$ for any locally minimal solution τ to $\mathcal{P}(\mathcal{F})$.*

Finally, we show that, in contrast to the other preprocessing techniques, BCE does not preserve locally minimal solutions; that is, BCE is more problematic in terms of misleading MaxSAT solvers in terms of the perceived costs of solutions during search.

Theorem 2 *BCE does not preserve locally minimal solutions.*

Proof. Consider the MaxSAT instance $\mathcal{F} = \{(\neg x \vee y), (x \vee \neg y), (\neg x \vee l)\}$ with $\mathcal{S}(\mathcal{F}) = \{l\}$ and $w^{\mathcal{F}}(l) = 1$. The clause $(\neg x \vee y)$ is blocked on $\neg x \notin \mathcal{S}(\mathcal{F})$ so BCE can be applied to obtain the instance $\mathrm{BCE}(\mathcal{F}) = \{(x \vee \neg y), (\neg x \vee l)\}$ with $\mathcal{S}(\mathrm{BCE}(\mathcal{F})) = \{l\}$ and $w^{\mathcal{P}(\mathcal{F})}(l) = 1$. The solution $\tau = \{l, x, \neg y\}$ is locally minimal to $\mathrm{BCE}(\mathcal{F})$. However, the solution $\mathrm{REC}(\tau) = \{l, \neg x, \neg y\}$ is not locally minimal to $\mathcal{F}$ since $\{\neg x, \neg y\}$ also satisfies $\mathcal{F}$. ■

Finally, we note that the underlying reason for misinterpreting solutions costs is not the literal-centric view; using BCE on standard MaxSAT instances (which is possible also without the use of soft variables [6]) misleads solvers.

Example 6 *Consider the standard MaxSAT instance* $\mathcal{F} = (F_h, F_s, w)$ *with* $F_h = \{(\neg x \vee y), (x \vee \neg y)\}$, $F_s = \{(\neg x)\}$ *and* $w((\neg x)) = 1$. *The clause* $(\neg x \vee y)$ *is blocked on* $\neg x$ *so BCE can be applied in order to obtain the (standard) instance* $\mathcal{P}(\mathcal{F}) = (F_h \setminus \{(\neg x \vee y)\}, F_s, w)$. *The solution* $\tau = \{x, \neg y\}$ *to* $\mathcal{P}(\mathcal{F})$ *has* $\textsc{Cost}(\mathcal{P}(\mathcal{F}), \tau) = 1$. *Since* $\tau((\neg x \vee y)) = 0$ *the reconstruction* $\textsc{REC}(\tau)$ *is* $(\tau \setminus \{x\}) \cup \{\neg x\}$ *and has* $\textsc{Cost}(\mathcal{F}, \textsc{REC}(\tau)) = 0$.

4 LOCAL SEARCH OVER LOCALLY MINIMAL SOLUTIONS

We describe a MaxSAT stochastic local search algorithm that is guaranteed to search over locally minimal solutions, thereby avoiding the misinterpretation of solution costs seen during search (recall Figure 1). While in principle one could apply solution reconstruction at each solution, this is in practice too time-consuming especially in the context of SLS; we propose a more practically suited approach. Though based on similar ideas as recent SLS methods [30], a key difference in our approach is to ensure locally minimal solutions via grouping clauses of soft variables according to the soft variables they contain, and using this structural information to obtain locally minimal solutions. In particular, starting from a (standard) MaxSAT instance $\mathcal{F}$ as input, we partition the clauses of the instance $\mathcal{P}(\mathcal{F})$ (obtained by applying preprocessing methods that preserve locally minimal solutions) into sets H and S, where $H = \{C \in \mathcal{P}(\mathcal{F}) \mid C \cap \mathcal{S}(\mathcal{F}) = \emptyset\}$ and $S = \{C \in \mathcal{P}(\mathcal{F}) \mid C \cap \mathcal{S}(\mathcal{F}) \neq \emptyset\}$. Further, the clauses of S are partitioned into disjoint groups $S_g = \{G^{x_c} \mid x_c \in \mathcal{S}(\mathcal{F})\}$. A clause $C \in S$ is included in group G^{x_c} if x_c is a soft variable in C with the lowest weight over all the soft variables in C (ties broken randomly).[4] We identify each clause $C \in H$ with the group $\{C\}$ and denote their collection by H_g. We say that a group G is satisfied by an assignment τ if $(\tau \setminus \mathcal{S}(\mathcal{F}))\,(G) = 1$, and otherwise unsatisfied.

Our approach ignores soft variables during search, and instead tries to find assignments τ which satisfy the groups in H_g. Note that if the group G^{x_c} contains only one soft variable—namely the variable x_c—then any solution τ that leaves at least one $C \in G^{x_c}$ unsatisfied must have $\tau(x_c) = 1$. Thus to not incur cost from x_c, each clause in G^{x_c} needs to be satisfied (a la Group MaxSAT [23]). Moreover, if each group contains at most one soft variable, any solution τ for which the invariant "$\tau(x_c) = 1$ iff G^{x_c} is unsatisfied" holds is locally minimal as no solution can set $\tau(x_c) = 0$ when G^{x_c} is unsatisfied. Each assigment τ can be modified to make the invariant hold by setting $\tau(x_c) = 0$ if G^{x_c} is satisfied and $\tau(x_c) = 1$ otherwise. For groups containing clauses with more than one soft variable, a solution τ for which the invariant holds is not necessarily locally minimal. To remedy this issue, we generate a locally minimal assignment $\tau' \subseteq \tau$ by first setting $\tau' := \tau$ and then iteratively setting $\tau'(x_c) = 0$ if $\tau' \setminus \{x_c\}$ is a solution, considering the variables $x_c \in \mathcal{S}(\tau)$ in decreasing order of $w^{\mathcal{F}}(x_c)$.

We refer to the proposed SLS method as LMS-SLS. The pseudocode of LMS-SLS is outlined in Algorithm 1. Given a standard instance $\mathcal{F}$, LMS-SLS begins by preprocessing the instance to obtain the literal-centric instance $\mathcal{P}(\mathcal{F})$ (line 1), which is then partitioned into groups (line 2).

After grouping, an initial assignment is constructed (lines 3–8). For groups the clauses of which contain at most one soft variable, locally minimal solutions are already guaranteed by lines 4–6, which is followed by unit propagation based decimation [16].

In the main search loop (lines 9–22), if the current assignment τ is a solution, a locally minimal $\tau' \subseteq \tau$ is computed (line 11) greedily as described above. The current incumbent solution τ^* is updated to τ' if $\textsc{appar-cost}(\mathcal{P}(\mathcal{F}), \tau')$ is lower than $\textsc{appar-cost}(\mathcal{P}(\mathcal{F}), \tau^*)$. A greedy step (lines 13–15) is performed whenever possible. That is, when the set D of variables with a positive score is non-empty, one such variable is chosen to be flipped by the *best from multiple selection* (BMS) strategy of [33, 30]. The BMS strategy samples t variables with replacement from D and returns the variable v with the highest score, breaking ties randomly. In particular, we adapt a dynamic weighting scheme proposed earlier in the context of MaxSAT SLS [34, 14, 33, 16, 30] to the level of groups, assigning scores to non-soft variables. Specifically, the weights of all groups in H_g are initialized to 1 and the weight of each $G^{x_c} \in S_g$ to $w^{\mathcal{F}}(x_c)$. During search the weight of each unsatisfied group $G \in H_g$ is increased by a user-defined parameter h_inc. The score of a variable $x \notin \mathcal{S}(\mathcal{F})$ with respect to an assignment τ is then the increase in the sum of the weights of the groups that would be satisfied by flipping x.

In case a greedy step is not possible, we update group weights (line 17) and select uniformly at random a currently unsatisfied group G to be satisfied by the SATISFY subroutine, preferring hard groups over soft groups (lines 18– 20). If $G \in H_g$, SATISFY flips a variable in G with the highest (least negative) score in it, breaking ties randomly (line 21). Otherwise, to satisfy $G \in S_g$, SATISFY employs a

Algorithm 1: LMS-SLS

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

1 Preprocess $\mathcal{F}$ to obtain $\mathcal{P}(\mathcal{F})$
2 Form the groups H_g and S_g
3 $\tau :=$ a random initial assignment for $\mathcal{P}(\mathcal{F})$
4 **for** $x_c \in \mathcal{S}(\mathcal{F})$ **do**
5 **if** G^{x_c} *is satisfied* **then** $\tau(x_c) = 0$
6 **else** $\tau(x_c) := 1$
7 $\tau :=$ decimation on τ and $\mathcal{P}(\mathcal{F})$ without soft variables
8 $\tau^* := \tau$
9 **while** *no termination criteria is met* **do**
10 **if** τ *is a solution* **then**
11 $\tau' :=$ greedily computed locally minimal solution for which $\tau' \subseteq \tau$
12 **if** $\textsc{appar-cost}(\mathcal{P}(\mathcal{F}), \tau') < \textsc{appar-cost}(\mathcal{P}(\mathcal{F}), \tau^*)$ **then** $\tau^* := \tau'$
13 **if** $D := \{v \notin \mathcal{S}(\mathcal{F}) \mid \textsc{Score}(v) > 0\} \neq \emptyset$ **then**
14 $v :=$ a variable in D chosen by the BMS strategy
15 $\tau := \tau$ with v flipped
16 **else**
17 update group weights
18 **if** $\exists G \in H_g$ *that is unsatisfied* **then**
19 $G :=$ an unsatisfied group from H_g
20 **else** $G :=$ an unsatisfied group from S_g
21 $\tau := \textsc{Satisfy}(\tau, G)$
22 update $\tau(x_c)$ for all $x_c \in \mathcal{S}(\mathcal{F})$
23 **if** τ^* *is a solution* **then**
24 **return** $\textsc{REC}(\tau^*), \textsc{Cost}(\mathcal{F}, \textsc{REC}(\tau^*))$
25 **else return** no solution found

[4] We note that over 68% of the weighted instances in the MaxSAT Evaluation 2019 incomplete track consist solely of clauses with at most one soft variable after preprocessing.

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)**
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)**
InterpretableClassifiers	15	**10 (0.946)**	5 (0.769)	0 (0.559)	**15 (1.000)**	**11 (0.994)**	9 (0.984)
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)**
RBAC maintenance	15	**15 (1.000)**	1 (0.753)	5 (0.818)	**10 (0.917)**	**8 (0.962)**	7 (0.987)
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)**
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)**

simple local search algorithm which either flips a variable in G randomly with probability $1 - gp$ or a variable with the highest *make* with probability gp. Here the make of a variable is defined as the number of unsatisfied clauses in G that would become satisfied by flipping the value of x in the current assignment. This process is iterated until G is satisfied. Finally, on line 22 we modify τ to guarantee that the invariant "$\tau(x_c) = 1$ iff G^{x_c} is unsatisfied" holds. At termination (line 24) we reconstruct the current incumbent solution for reporting the best solution found and its cost.

Experiments. We evaluate the impact of ensuring that solutions considered during search are locally minimal on the practical performance of the proposed SLS approach. We consider the following variants.

default: the LMS-SLS approach in its full, with all of the preprocessing techniques that preserve locally minimal solutions applied before search.

+BCE: **LMS-SLS** with additionally BCE applied before search (recall that BCE does not preserve locally minimal solutions).

-G: **LMS-SLS** without clause grouping, i.e., treating each soft clause as a group of size 1 and thus not guaranteeing locally minimal solutions.

-G-pre: **-G** without preprocessing.

We implemented the approach in C++ on top of SATLike [30]. Following [30], we set $h\_inc = 300$ if the average weight of the soft clauses in the instance was higher than 10000, and 3 otherwise. We also set $t = 15$ and $gp = 0.8$ based on preliminary experiments. We used MaxPre [28] as the preprocessor. As benchmarks, we used the weighted instances used in the incomplete track of MaxSAT Evaluation 2019. The experiments were run on 2.4-GHz Intel Xeon E5-2680-v4, 256-GB computers with a per-instance time limit of 300 s, preprocessing time included, and a memory limit of 32 GB. We used a per-instance time limit of 10 s for preprocessing to ensure that preprocessing times do not dominate, while also generally achieving simplifications via MaxPre's internal technique scheduling.

A pairwise comparison of the relative performance of the four variants is shown in Table 1, grouped into comparisons (**-G-pre** vs **-G**), (**-G** vs **default**), and (**default** vs **+BCE**) wrt both the number of wins (best solution found) and the score metric used in MaxSAT Evaluations (a larger value is better). This allows us to make the following observations. First, we observe that without grouping, **-G-pre** performs overall better than **-G**, i.e., preprocessing in fact degrades performance when not only locally minimal solutions are considered during search. However, **default** improves considerably over **-G-pre**. This witnesses the practical potential of Theorem 1: by ensuring that only locally minimal solutions are considered during search in conjunction with applying preprocessing techniques that preserve locally minimal solutions, search performance can be improved. Note that **default** also outperforms **+BCE** overall. This would seem to suggest that BCE (that does not preserve locally minimal solutions by Theorem 2) indeed degrades search performance, which may be the result of applying BCE leading to misinterpreting costs of (even locally minimal) solutions during search. Finally, we note that the current prototype implementation of LMS-SLS is overall on par with SATLike. Interestingly, however, SATLike (with preprocessing) and **default** exhibit complementary performance; e.g. on lisbon-wedding SATLike wins 10 times against **default**'s 0 wins, while e.g. on min-width **default** wins 16 times against SATLike's 1 win.

5 CONCLUSIONS

We showed that central preprocessing techniques lead to MaxSAT solvers misinterpreting the costs of solutions during search both in theory and in practice. This may at least in part explain why the

performance gains obtained through preprocessing have appeared to be more modest in MaxSAT solving compared to SAT solving. Towards resolving this intricate issue in applying preprocessing as part of the MaxSAT solving workflow, we showed that, by assuming that MaxSAT solvers focus on so-called locally minimal solutions, the issue is fixed for most (but not all) preprocessing techniques. As the first steps towards designing MaxSAT solving techniques which focus on locally minimal solutions, we presented a stochastic local search approach with this sought-after property and empirically showed that search over locally minimal solutions indeed yields practical performance improvements. Adapting other types of MaxSAT algorithms, including SAT-based and branch-and-bound solvers [31, 32], to focus on locally minimal solutions is a promising direction of improving the practical impact of preprocessing for MaxSAT. Combinations of SAT-based preprocessing with MaxSAT preprocessing techniques based on MaxSAT resolution [22] could also be considered.

ACKNOWLEDGEMENTS

This work has been financially supported by Academy of Finland (grants 276412, 312662, 322869).

REFERENCES

[1] C. Ansótegui, M.L. Bonet, J. Gabàs, and J. Levy, 'Improving SAT-based weighted MaxSAT solvers', in *Proc. CP*, volume 7514 of *LNCS*, pp. 86–101. Springer, (2012).
[2] C. Ansótegui and J. Gabàs, 'Solving (weighted) partial MaxSAT with ILP', in *Proc. CPAIOR*, volume 7874 of *LNCS*, pp. 403–409. Springer, (2013).
[3] J. Argelich, C.M. Li, F. Manyà, and J. Planes, 'The first and second Max-SAT evaluations', *J. Satisfiability, Boolean Modeling and Computation*, **4**(2-4), 251–278, (2008).
[4] F. Bacchus, A Hyttinen, M. Järvisalo, and P. Saikko, 'Reduced cost fixing for maximum satisfiability', in *Proc. IJCAI*, pp. 5209–5213. ijcai.org, (2018).
[5] F. Bacchus, M. Järvisalo, and R. Martins, 'MaxSAT Evaluation 2018: New developments and detailed results', *J. Satisfiability, Boolean Modeling and Computation*, **11**, 99–131, (2019).
[6] A. Belov, A. Morgado, and J. Marques-Silva, 'SAT-based preprocessing for MaxSAT', in *Proc. LPAR*, volume 8312 of *LNCS*, pp. 96–111. Springer, (2013).
[7] J. Berg, E. Demirovic, and P.J. Stuckey, 'Core-boosted linear search for incomplete MaxSAT', in *Proc. CPAIOR*, volume 11494 of *LNCS*, pp. 39–56. Springer, (2019).
[8] J. Berg and M. Järvisalo, 'Impact of SAT-based preprocessing on core-guided MaxSAT solving', in *Proc. CP*, volume 9892 of *LNCS*, pp. 66–85. Springer, (2016).
[9] J. Berg and M. Järvisalo, 'Weight-aware core extraction in SAT-based MaxSAT solving', in *Proc. CP*, volume 10416 of *LNCS*, pp. 652–670. Springer, (2017).
[10] J. Berg and M. Järvisalo, 'Unifying reasoning and core-guided search for maximum satisfiability', in *Proc. JELIA*, volume 11468 of *LNCS*, pp. 287–303. Springer, (2019).
[11] J. Berg, P. Saikko, and M. Järvisalo, 'Improving the effectiveness of SAT-based preprocessing for MaxSAT', in *Proc. IJCAI*, pp. 239–245. AAAI Press, (2015).
[12] J. Berg, P. Saikko, and M. Järvisalo, 'Re-using auxiliary variables for MaxSAT preprocessing', in *Proc. ICTAI*, pp. 813–820, (2015).
[13] J. Berg, P. Saikko, and M. Järvisalo, 'Subsumed label elimination for maximum satisfiability', in *Proc. ECAI*, volume 285 of *FAIA*, pp. 630–638. IOS Press, (2016).
[14] S. Cai, C. Luo, J. Lin, and K. Su, 'New local search methods for partial MaxSAT', *Artif. Intell.*, **240**, 1–18, (2016).
[15] S. Cai, C. Luo, J. Thornton, and K. Su, 'Tailoring local search for partial MaxSAT', in *Proc. AAAI*, pp. 2623–2629. AAAI Press, (2014).
[16] S. Cai, C. Luo, and H. Zhang, 'From decimation to local search and back: A new approach to MaxSAT', in *Proc. IJCAI*, pp. 571–577, (2017).
[17] J. Davies and F. Bacchus, 'Solving MAXSAT by solving a sequence of simpler SAT instances', in *Proc. CP*, volume 6876 of *LNCS*, pp. 225–239. Springer, (2011).
[18] J. Davies and F. Bacchus, 'Exploiting the power of MIP solvers in MaxSat', in *Proc. SAT*, volume 7962 of *LNCS*, pp. 166–181. Springer, (2013).
[19] J. Davies and F. Bacchus, 'Postponing optimization to speed up MAXSAT solving', in *Proc. CP*, volume 8124 of *LNCS*, pp. 247–262. Springer, (2013).
[20] 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 *Proc. CP*, volume 11802 of *LNCS*, pp. 177–194. Springer, (2019).
[21] N. Eén and A. Biere, 'Effective preprocessing in SAT through variable and clause elimination', in *Proc. SAT*, volume 3569 of *LNCS*, pp. 61–75. Springer, (2005).
[22] F. Heras and D. Bañeres, 'The impact of Max-SAT resolution-based preprocessors on local search solvers', *J. Satisfiability, Boolean Modeling and Computation*, **7**(2-3), 89–126, (2010).
[23] F. Heras, A. Morgado, and J. Marques-Silva, 'An empirical study of encodings for group MaxSAT', in *Canadian Conference on AI*, volume 7310 of *LNCS*, pp. 85–96. Springer, (2012).
[24] M. Janota and J. Marques-Silva, 'On the query complexity of selecting minimal sets for monotone predicates', *Artif. Intell.*, **233**, 73–83, (2016).
[25] M. Järvisalo and A. Biere, 'Reconstructing solutions after blocked clause elimination', in *Proc. SAT*, volume 6175 of *LNCS*, pp. 340–345. Springer, (2010).
[26] M. Järvisalo, A. Biere, and M. Heule, 'Blocked clause elimination', in *Proc. TACAS*, volume 6015 of *LNCS*, pp. 129–144. Springer, (2010).
[27] M. Järvisalo, M. Heule, and A. Biere, 'Inprocessing rules', in *Proc. IJCAR*, volume 7364 of *LNCS*, pp. 355–370. Springer, (2012).
[28] T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo, 'MaxPre: An extended MaxSAT preprocessor', in *Proc. SAT*, volume 10491 of *LNCS*, pp. 449–456. Springer, (2017).
[29] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, 'QMaxSAT: A partial Max-SAT solver', *J. Satisfiability, Boolean Modeling and Computation*, **8**(1/2), 95–100, (2012).
[30] Z. Lei and S. Cai, 'Solving (weighted) partial MaxSAT by dynamic local search for SAT', in *Proc. IJCAI*, pp. 1346–1352. ijcai.org, (7 2018).
[31] C. M. Li, F. Manyà, N.O. Mohamedou, and J. Planes, 'Exploiting cycle structures in Max-SAT', in *Proc. SAT*, volume 5584 of *LNCS*, pp. 467–480. Springer, (2009).
[32] C. M. Li, F. Manyà, and J. Planes, 'Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers', in *Proc. CP*, volume 3709 of *LNCS*, pp. 403–414. Springer, (2005).
[33] C. Luo, S. Cai, K. Su, and W. Huang, 'CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability', *Artif. Intell.*, **243**, 26–44, (2017).
[34] C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su, 'CCLS: an efficient local search algorithm for weighted maximum satisfiability', *IEEE Trans. Computers*, **64**(7), 1830–1843, (2015).
[35] R. Martins, V. M. Manquinho, and I. Lynce, 'Open-wbo: A modular MaxSAT solver,', in *Proc. SAT*, volume 8561 of *LNCS*, pp. 438–445. Springer, (2014).
[36] R. Martins, V. M. Manquinho, and I. Lynce, 'Improving linear search algorithms with model-based approaches for MaxSAT solving', *J. Exp. Theor. Artif. Intell.*, **27**(5), 673–701, (2015).
[37] A. Morgado, C. Dodaro, and J. Marques-Silva, 'Core-guided MaxSAT with soft cardinality constraints', in *Proc. CP*, volume 8656 of *LNCS*, pp. 564–573. Springer, (2014).
[38] A. Morgado, F. Heras, M.H. Liffiton, J. Planes, and J. Marques-Silva, 'Iterative and core-guided MaxSAT solving: A survey and assessment', *Constraints*, **18**(4), 478–534, (2013).
[39] A. Morgado, F. Heras, and J. Marques-Silva, 'Model-guided approaches for MaxSAT solving', in *Proc. ICTAI*, pp. 931–938. IEEE Computer Society, (2013).
[40] N. Narodytska and F. Bacchus, 'Maximum satisfiability using core-guided MaxSAT resolution', in *Proc. AAAI*, pp. 2717–2723. AAAI Press, (2014).
[41] P. Saikko, J. Berg, and M. Järvisalo, 'LMHS: A SAT-IP hybrid MaxSAT solver', in *Proc. SAT*, volume 9710 of *LNCS*, pp. 539–546. Springer, (2016).