

On Computing Preferred MUSes and MCSes

Joao Marques-Silva^{1,2} and Alessandro Previti¹

¹ CASL, University College Dublin, Ireland

² IST/INESC-ID, Technical University of Lisbon, Portugal

Abstract. Minimal Unsatisfiable Subsets (MUSes) and Minimal Correction Subsets (MCSes) are essential tools for the analysis of unsatisfiable formulas. MUSes and MCSes find a growing number of applications, that include abstraction refinement in software verification, type debugging, software package management and software configuration, among many others. In some applications, there can exist preferences over which clauses to include in computed MUSes or MCSes, but also in computed Maximal Satisfiable Subsets (MSSes). Moreover, different definitions of preferred MUSes, MCSes and MSSes can be considered. This paper revisits existing definitions of preferred MUSes, MCSes and MSSes of unsatisfiable formulas, and develops a preliminary characterization of the computational complexity of computing preferred MUSes, MCSes and MSSes. Moreover, the paper investigates which of the existing algorithms and pruning techniques can be applied for computing preferred MUSes, MCSes and MSSes. Finally, the paper shows that the computation of preferred sets can have significant impact in practical performance.

1 Introduction

Reasoning about over-constrained systems finds a wide range of practical application, and has been the subject of extensive research over the last three decades (e.g. [35,19,29,22,38]). For the concrete case of propositional formulas, Minimal Unsatisfiable Subsets (MUSes), Minimal Correction Subsets (MCSes) and Maximal Satisfiable Subsets (MSSes) are essential tools for the analysis of unsatisfiable propositional formulas in Conjunctive Normal Form (CNF). MUSes, MCSes and MSSes find a growing number of applications, that include abstraction refinement in software verification (e.g. [23]), software (re-)configuration (e.g. [20]), software package management (e.g. [18]), type debugging (e.g. [3]), abstract argumentation frameworks (e.g. [39]), fault localization in C code (e.g. [21]), design debugging (e.g. [10]), post-silicon debugging (e.g. [40]), among many others. MSSes/MCSes are also tightly related with Maximum Satisfiability (MaxSAT) (e.g. [1,30]). In recent years, there has been extensive work on

* This work is partially supported by SFI grant BEACON (09/IN.1/I2618), by FCT grant POLARIS (PTDC/EIA-CCO/123051/2010), and INESC-IDs multiannual PIDDAC funding PEst-OE/EEI/LA0021/2011.

developing efficient algorithms for computing MUSes [16,28,5,27,24,31] and MCSes/MSSes [3,33,14,32,26]. Despite all the new algorithms and techniques proposed, the issue of computing preferred sets, namely MUSes, MCSes and MSSes, has seldom been investigated. The main exception is [22], that studies preferred explanations (MUSes in our context) and relaxations (MSSes in our context) for the case of generalized constraints. Nevertheless, the importance of preferences in a number of fields is well-known (e.g. [8,36] and references therein). Moreover, preferences in SAT have also been the subject of comprehensive research in recent years, which is summarized for example in [13]. To our best knowledge, the computation of preferred MUSes, MCSes or MSSes, their complexity, their algorithms and associated techniques, have not been studied before.

This paper represents a first step towards a systematic analysis of computing preferred MUSes, MCSes and MSSes. The first topic addressed by the paper are the possible definitions of what a preferred set (either MUS, MCS or MSS) represents. This paper focuses solely on the definitions of preference considered in [22]. Given these definitions, the paper then studies the computational complexity of computing preferred sets (either MUSes, MCSes or MSSes), focusing on hardness results. As the paper shows, computing MUSes or MCSes for the natural (lexicographic) definition of preference is hard for the second level of the polynomial hierarchy¹. This in part explains the alternative definition of preference (i.e. anti-lexicographic) used in [22] for MUSes. For this alternative definition of preferred set, computing a preferred MUS (or MCS) is in FP^{NP} , and so can be computed with a polynomial number of calls to an NP oracle. Similarly, as the paper shows, computing a (lexicographically) preferred MSS is in FP^{NP} , and so it is *easier*² than computing a lexicographic preferred MUS (or MCS). Moreover, the paper investigates whether well-known algorithms for computing MUSes/MCSes/MSSes can be extended for computing preferred sets, and concludes that some algorithms cannot be extended for computing some preferred sets. In addition, the paper investigates whether well-known and widely used techniques for reducing the number of NP oracle calls can be extended for computing preferred sets and, again, concludes that several techniques cannot be used for computing preferred sets. Finally, the paper assesses the performance impact of computing preferred sets in state of the art tools for computing MUSes/MCSes/MSSes. Although the paper addresses propositional formulas in CNF, the results can be generalized to more expressive domains, including CSP and SMT.

The paper is organized as follows. Section 2 introduces the notation and definitions used in the rest of the paper. Section 3 develops preliminary complexity results regarding the computation of preferred MUSes/MCSes/MSSes. Section 4 surveys well-known algorithms and investigates which can be extended

¹ The complexity classes considered in this paper are briefly reviewed in Section 2. Similarly, the paper develops complexity characterizations assuming an NP oracle which, for the purposes of this paper, can be seen as equivalent to a SAT solver call.

² Under the assumption the polynomial hierarchy does not collapse. This is a generally accepted conjecture that is also (implicitly) assumed throughout the paper.

for computing preferred sets. Similarly, Section 5 surveys some of the best known techniques for reducing the number of NP oracle calls, and investigates which can be extended for computing preferred sets. Section 6 assesses the practical impact of some of the paper’s results. The paper concludes in Section 7.

2 Preliminaries

This section introduces the notation and definitions used throughout the paper.

2.1 CNF Formulas, MUSes, MCSes and MSSes

Standard propositional logic definitions apply (e.g. [6]). CNF formulas are defined over a set of propositional variables. A CNF formula $\mathcal{F}$ is a conjunction of clauses, also interpreted as a set of clauses. A clause is a disjunction of literals, also interpreted as a set of literals. A literal is a variable or its complement. $m \triangleq |\mathcal{F}|$ represents the number of clauses. The set of variables associated with a CNF formula $\mathcal{F}$ is denoted by $X \triangleq \text{var}(\mathcal{F})$. The clauses of $\mathcal{F}$ are *soft*, meaning that these clauses can be relaxed (i.e. not satisfied). In contrast, an optional set of *hard* clauses $\mathcal{H}$ may be specified. (For simplicity it is assumed that $\mathcal{F} \cap \mathcal{H} = \emptyset$. Moreover, $\mathcal{H}$ is implicit in some definitions.) MUSes, MCSes and MSSes are defined over $\mathcal{F}$ taking into account the hard clauses in $\mathcal{H}$ as follows:

Definition 1 (Minimal Unsatisfiable Subset (MUS)). *Let $\mathcal{F}$ denote the set of soft clauses and $\mathcal{H}$ denote the set of hard clauses, such that $\mathcal{H} \cup \mathcal{F}$ is unsatisfiable. $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{H} \cup \mathcal{M}$ is unsatisfiable and $\forall \mathcal{M}' \subsetneq \mathcal{M}$, $\mathcal{H} \cup \mathcal{M}'$ is satisfiable.*

Definition 2 (Minimal Correction Subset (MCS)). *Let $\mathcal{F}$ denote the set of soft clauses and $\mathcal{H}$ denote the set of hard clauses, such that $\mathcal{H} \cup \mathcal{F}$ is unsatisfiable. $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{H} \cup \mathcal{F} \setminus \mathcal{C}$ is satisfiable and $\forall \mathcal{C}' \subsetneq \mathcal{C}$, $\mathcal{H} \cup \mathcal{F} \setminus \mathcal{C}'$ is unsatisfiable.*

Definition 3 (Maximal Satisfiable Subset (MSS)). *Let $\mathcal{F}$ denote the set of soft clauses and $\mathcal{H}$ denote the set of hard clauses, such that $\mathcal{H} \cup \mathcal{F}$ is unsatisfiable. $\mathcal{S} \subseteq \mathcal{F}$ is a Maximal Satisfiable Subset (MSS) iff $\mathcal{H} \cup \mathcal{S}$ is satisfiable and $\forall \mathcal{S}' \supsetneq \mathcal{S}$, $\mathcal{H} \cup \mathcal{S}'$ is unsatisfiable.*

2.2 Set Membership Problems

To establish some of the complexity results in Section 3, the following set membership/non-membership decision problems are defined.

Definition 4 (MUS Membership). *Given $c \in \mathcal{F}$, the MUS membership problem is to decide whether there exists an MUS $\mathcal{M} \subseteq \mathcal{F}$ such that $c \in \mathcal{M}$.*

Definition 5 (MCS Membership). *Given $c \in \mathcal{F}$, the MCS membership problem is to decide whether there exists an MCS $\mathcal{C} \subseteq \mathcal{F}$ such that $c \in \mathcal{C}$.*

Definition 6 (MSS Non-membership). *Given $c \in \mathcal{F}$, the MSS non-membership problem is to decide whether there exists an MSS $\mathcal{S} \subseteq \mathcal{F}$ such that $c \notin \mathcal{S}$.*

Additional membership/non-membership problems could be defined, but are unnecessary for the purposes of the paper. Clearly, the above definitions can be extended to a set of target clauses $\mathcal{T} \subseteq \mathcal{F}$, such that membership (resp. non-membership) consists in deciding whether all clauses in $\mathcal{T}$ are (resp. are not) in some set.

2.3 Preferred MUSes, MCSes & MSSes

This section revisits the definitions from [22] adapted to CNF formulas.

Definition 7 (Precedence Operator, $\prec$). *Let $c_1, c_2 \in \mathcal{F}$. $c_1 \prec c_2$ denotes that clause c_1 is preferred to clause c_2 .*

A strict total order $<$ is assumed between the clauses of $\mathcal{F}$. This can be viewed as a linearization of the partial order induced by the $\prec$. (See [22] for a detailed discussion of the linearization $<$). As a result, the clauses of $\mathcal{F}$ are listed as $\langle c_1, c_2, \dots, c_m \rangle$, respecting the strict total order $<$. Throughout the paper, sequences are used to represent preferences among clauses in a set, e.g. $\langle c_1, c_2, \dots, c_m \rangle$.

Definition 8 (Lexicographic preference, L-preference). *Given a strict total order $<$ on $\mathcal{F}$, a set $\mathcal{A} \subseteq \mathcal{F}$ is lexicographically preferred to another set $\mathcal{B} \subseteq \mathcal{F}$, denoted $\mathcal{A} <_{\text{lex}} \mathcal{B}$, iff $\exists 1 \leq k \leq m$, $c_k \in \mathcal{A} \setminus \mathcal{B}$ and $\mathcal{A} \cap \{c_1, \dots, c_{k-1}\} = \mathcal{B} \cap \{c_1, \dots, c_{k-1}\}$.*

Definition 9 (Anti-lexicographic preference, A-preference). *Given a strict total order $<$ on $\mathcal{F}$, a set $\mathcal{A} \subseteq \mathcal{F}$ is anti-lexicographically preferred to another set $\mathcal{B} \subseteq \mathcal{F}$, denoted $\mathcal{A} <_{\text{antilex}} \mathcal{B}$, iff $\exists 1 \leq k \leq m$, $c_k \in \mathcal{B} \setminus \mathcal{A}$ and $\mathcal{A} \cap \{c_{k+1}, \dots, c_m\} = \mathcal{B} \cap \{c_{k+1}, \dots, c_m\}$.*

Definition 10 (L-preferred/A-preferred MUSes/MCSes/MSSes). *An MUS/MCS/MSS $\mathcal{A}$ is L-preferred (resp. A-preferred) if for all MUS/MCS/MSS $\mathcal{B} \neq \mathcal{A}$, $\mathcal{A} <_{\text{lex}} \mathcal{B}$ (resp. $\mathcal{A} <_{\text{antilex}} \mathcal{B}$).*

For simplicity, throughout the paper we use the expression *preferred set* to denote L-preferred or A-preferred MUSes, MCSes or MSSes. Moreover, given a strict total order for a given set of clauses, there exists a unique preferred set.

Example 1. Let $\mathcal{H} = \emptyset$ and let $\mathcal{F}$ be given by the following (soft) clauses:

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\bar{x}_1 \vee \bar{x}_2)$	(x_1)	$(x_5 \vee x_6)$	$(\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$	(x_3)	(x_4)	(x_2)

where the strict total order is: $\langle c_1, c_2, \dots, c_7 \rangle$. The MSSes, MCSes and MUSes of this CNF formula are shown in Table 1, which also shows which are L-preferred or A-preferred. Observe that an L-preferred set preferably includes the most preferred clauses, and so captures the more intuitive notion of preference. In contrast, an A-preferred set preferably discards the least preferred clauses.

Table 1: MSSes, MCSes & MUSes for Example 1

Set	Type	L or A -preferred?
$\{c_1, c_3, c_4, c_5, c_6, c_7\}$	MSS	$\times$
$\{c_1, c_2, c_3, c_4, c_5\}$	MSS	$\checkmark$ (L & A)
$\{c_1, c_2, c_3, c_4, c_6\}$	MSS	$\times$
$\{c_1, c_2, c_3, c_5, c_6\}$	MSS	$\times$
$\{c_2, c_3, c_4, c_5, c_7\}$	MSS	$\times$
$\{c_2, c_3, c_4, c_6, c_7\}$	MSS	$\times$
$\{c_2, c_3, c_5, c_6, c_7\}$	MSS	$\times$
$\{c_2\}$	MCS	$\checkmark$ (A)
$\{c_6, c_7\}$	MCS	$\times$
$\{c_5, c_7\}$	MCS	$\times$
$\{c_4, c_7\}$	MCS	$\times$
$\{c_1, c_6\}$	MCS	$\times$
$\{c_1, c_5\}$	MCS	$\times$
$\{c_1, c_4\}$	MCS	$\checkmark$ (L)
$\{c_1, c_2, c_7\}$	MUS	$\checkmark$ (L)
$\{c_2, c_4, c_5, c_6\}$	MUS	$\checkmark$ (A)

Example 2. To further highlight the differences between L-preferred and A-preferred sets, consider a set $\mathcal{F} = \{c_1, c_2, c_3, c_4\}$ with the strict total order $\langle c_1, c_2, c_3, c_4 \rangle$, and let the minimal sets of interest be $\mathcal{S}_1 = \{c_1, c_4\}$ and $\mathcal{S}_2 = \{c_2, c_3\}$. Thus, $\mathcal{S}_1$ is L-preferred, whereas $\mathcal{S}_2$ is A-preferred.

2.4 Computational Complexity

Standard computational complexity definitions are used throughout the paper [15,34]. The notation is adapted from [34] and more recent papers (e.g. [25]). Besides the well-known complexity classes P, NP, coNP and D^P , the following additional complexity classes of decision problems are used in the paper:

- $P^{NP} = \Delta_2^P$: class of decision problems solvable in deterministic polynomial time by executing a polynomial number of calls to an NP oracle.
- $NP^{NP} = \Sigma_2^P$: class of decision problems solvable in non-deterministic polynomial time by executing a polynomial number of calls to an NP oracle.

Moreover, the following classes of function problems are used in the paper (see [34] for a definition of function problem):

- FP: class of function problems solvable in deterministic polynomial time.
- FP^{NP} : class of function problems solvable in deterministic polynomial time by executing a polynomial number of calls to an NP oracle.
- $FP^{NP}[\mathcal{O}(\log n)]$: class of function problems solvable in deterministic polynomial time by executing a logarithmic number of calls to an NP oracle.

- $\text{FP}^{\Sigma_2^P}$: class of function problems solvable in deterministic polynomial time by executing a polynomial number of calls to a Σ_2^P oracle.

The classes listed above can be extended to other levels of the polynomial hierarchies of decision and function problems (e.g. [34]).

2.5 Related Work

Preferences have been extensively studied in the recent past (e.g. [8,36,13]). The computation of preferred sets of explanations and relaxations in the area of constraints is studied in [22]. To our best knowledge, there is no work studying the computational complexity of computing preferred MUSes, MCSes and MSSes, nor investigating the restrictions/limitations of existing algorithms and techniques for computing preferred sets. There exists a large body of work on extracting MUSes, MCSes and MSSes. Recent work includes [28,14,5,32,27,26,24,31]. Well-known past work includes [35,12,11,4,22,3,33]. The use of MaxSAT for computing MCSes/MSSes is well-known. Recent work is surveyed in [1,30]. Complexity results on computing MSSes are available from [9].

3 Complexity Results

This section develops computational complexity results for computing L-preferred and A-preferred MUSes, MCSes and MSSes. This characterization is relevant, because it allows us to understand which preferred sets we can expect to be able to compute with at most a polynomial number of calls to an NP oracle. We start in Section 3.1 by developing some results which are used in later sections.

3.1 Membership Tests

We first characterize the computational complexity of testing whether a clause is in some MUS, in some MCS, not in some MSS, and in some MSS.

Proposition 1 (MUS Membership). *The MUS membership problem is Σ_2^P -complete [25].*

Proof. The proof in [25, Theorem 4, page 216], regarding deciding clause membership in an irredundant subformula, can be used, since the proof of hardness is actually for an unsatisfiable formula. $\square$

Proposition 2 (MCS Membership). *The MCS membership problem is Σ_2^P -complete.*

Proof. (Sketch) MCS membership is in Σ_2^P . Let c be the target clause. Guess $\mathcal{C} \subseteq \mathcal{F}$, with $c \in \mathcal{C}$, such that $\mathcal{F} \setminus \mathcal{C}$ is satisfiable, and $\mathcal{C}$ is irreducible. To prove Σ_2^P -hardness, we reduce MUS membership to MCS membership. By minimal hitting set duality between MUSes and MCSes (e.g. [7,3], but also [35]), a clause c is in some MUS iff c is in some MCS. Thus, MUS membership can be decided with MCS membership. $\square$

Proposition 3 (MSS Non-membership). *The MSS non-membership problem is Σ_2^P -complete.*

Proof. Similarly to MCS membership, MSS non-membership is clearly in Σ_2^P . To prove Σ_2^P -hardness, we reduce MSS non-membership to MCS membership. Let $c \in \mathcal{F}$ be a target clause. c is in some MCS of $\mathcal{F}$ iff c is not in some MSS of $\mathcal{F}$. But deciding whether c is not in some MSS of $\mathcal{F}$ is the MSS non-membership problem. $\square$

Proposition 4. *The MUS/MCS membership problems and the MSS non-membership problem extended to a target set $\mathcal{T} \subsetneq \mathcal{F}$ are Σ_2^P -complete.*

Proof. (Sketch) MUS/MCS membership and MSS non-membership of a single clause (see Proposition 1, Proposition 2 and Proposition 3) are restrictions of the general case when $|\mathcal{T}| = 1$. The proof of membership generalizes the argument in the proof of Proposition 2. $\square$

In contrast, MSS membership is relatively easier to decide, as shown below.

Proposition 5 (MSS Membership). *MSS membership is in P if $\mathcal{H} = \emptyset$ and $\mathcal{F}$ has no empty clauses. If $\mathcal{H} \neq \emptyset$, and $\mathcal{H}$ is an arbitrary set of clauses, then MSS membership is NP-complete.*

Proof. Let $\mathcal{H} = \emptyset$. The proof is by contradiction. A clause $c \in \mathcal{F}$ is in some MSS of $\mathcal{F}$ iff it is not in every MCS of $\mathcal{F}$. Assume c was included in every MCS of $\mathcal{F}$. Then, by minimal hitting set duality [7], there would be an MUS with a single non-empty clause c ; a contradiction. We now consider the case when $\mathcal{H} \neq \emptyset$. MSS membership is clearly in NP. Let c be the target clause. Guess a truth assignment. If clause c and the hard clauses are satisfied, then c is in some MSS. Completeness is straightforward, by reducing SAT to MSS membership. Given a CNF formula $\mathcal{G}$, just pick any clause $c_j \in \mathcal{G}$, make c_j soft and the other clauses of $\mathcal{G}$ hard, i.e. $\mathcal{F} = \{c_j\}$, $\mathcal{H} = \mathcal{G} \setminus \{c_j\}$, and check the membership of c_j in some MSS of $\mathcal{F}$ given $\mathcal{H}$. Note that $\mathcal{H}$ may be unsatisfiable. This also indicates there is no MSS for the constructed formula, and so c_j is not in some MSS of $\mathcal{F}$. $\square$

3.2 L-Preferred & A-Preferred MUSes, MCSes & MSSes

Algorithms for computing A-preferred MUSes can be adapted from [22]. Moreover, the relaxation algorithm outlined in [22] shows that computing an L-preferred MSS is in FP^{NP} . This result is revisited later in this section.

Proposition 6. *Computing an L-preferred MUS (resp. MCS) is hard for Σ_2^P .*

Proof. We reduce the MUS (resp. MCS) membership problem to the problem of computing an L-preferred MUS (resp. MCS). Let $c \in \mathcal{F}$ be the target clause for MUS (resp. MCS) membership. Make c preferred over all other clauses in $\mathcal{F}$. c is in an MUS (resp. MCS) iff the L-preferred MUS (resp. MCS) contains c . $\square$

Proposition 7. *Computing an L-preferred MUS (resp. MCS) is in $\text{FP}^{\Sigma_2^P}$.*

Proof. We describe a polynomial time algorithm that uses an oracle for Σ_2^P to construct a preferred MUS (resp. MCS). Let $\mathcal{S}$ denote the set to be constructed, initially set to $\emptyset$. Analyze all clauses of $\mathcal{F}$, by decreasing preference given a total order on the clauses of $\mathcal{F}$. For each clause c_i check whether $\mathcal{S} \cup \{c_i\}$ is in some MUS (resp. MCS). By Proposition 4, this check can be done with a Σ_2^P oracle. If the outcome is true, then add c_i to $\mathcal{S}$. The algorithm executes a linear number of iterations given by the number of clauses of $\mathcal{F}$, each time invoking a Σ_2^P oracle. Thus, computing an L-preferred MUS (resp. MCS) is in $\text{FP}^{\Sigma_2^P}$. $\square$

Proposition 8. *Computing an A-preferred MSS is hard for Σ_2^P .*

Proof. We reduce MSS non-membership to computing an A-preferred MSS. Let $c \in \mathcal{F}$ be the target clause for MSS non-membership. Make c the least preferred clause over all clauses in $\mathcal{F}$. c is not in an MSS iff the A-preferred MSS does not contains c . $\square$

Proposition 9. *Computing an A-preferred MSS is in $\text{FP}^{\Sigma_2^P}$.*

Proof. The proof mimics that of Proposition 7. $\square$

In contrast, computing an L-preferred MSS is (believed to be) easier.

Proposition 10. *Computing an L-preferred MSS is in FP^{NP} .*

Proof. One possible algorithm considers the total order of clauses, in decreasing order of preference, for possible inclusion in a set $\mathcal{S}$. If the formula composed of $\mathcal{S}$ and target clause c is satisfiable, then c is added to $\mathcal{S}$. The proof that the computed MSS is L-preferred is by induction on the total order of clauses. For the basis case, let c_0 be the most preferred clause. If c_0 can be satisfied, then it is included in $\mathcal{S}$. Clearly, there can be no MSSes without c_0 that are preferred over an MSS that includes c_0 . The inductive step is similar, by considering the clauses already in $\mathcal{S}$. $\square$

It should be pointed out that the algorithms outlined in [22] can also be viewed as the sketch of a proof of FP^{NP} membership for the function problem of computing an L-preferred MSS. Moreover, the algorithm described in the proof above corresponds to the plain Grow procedure, as described in [3], and without the optimizations proposed in [32].

Proposition 11. *Computing an A-preferred MUS (resp. MCS) is in FP^{NP} .*

Proof. This is immediate from [22], since the QuickXplain algorithm is shown to compute (A-)preferred explanations (i.e. MUSes in our context). Moreover, the proof of Proposition 10 can easily be adapted for the case of A-preferred MCSes (but also as an alternative for A-preferred MUSes), by considering clauses from least to most preferred when deciding MSS membership. Clauses not included

Table 2: Summary of complexity results

	Membership			Hardness		
	MUS	MCS	MSS	MUS	MCS	MSS
L-preferred	$\text{FP}^{\Sigma_2^P}$	$\text{FP}^{\Sigma_2^P}$	FP^{NP}	Σ_2^P	Σ_2^P	D^P
A-preferred	FP^{NP}	FP^{NP}	$\text{FP}^{\Sigma_2^P}$	D^P	D^P	Σ_2^P

in the MSS are in an A-preferred MCS. $\square$

A summary of the results in this section is shown in Table 2. Observe that it is simple, but less significant, to prove that computing an A-preferred MUS/MCS and an L-preferred MSS are all hard for D^P .

Remark 1. It seems unlikely that computing L-preferred MSSes and A-preferred MUSes/MCSes could be in $\text{FP}^{\text{NP}}[\mathcal{O}(\log n)]$, since each clause must be considered separately for inclusion in a preferred set, given the strict order of clauses imposed by the preference. This issue is left for future research.

The results of this section, as well as the insights provided by Example 1, motivate the following result.

Proposition 12. *Consider a set of soft clauses $\mathcal{F}$, with a strict total order $<$, $\langle c_1, c_2, \dots, c_m \rangle$, and a inverse strict total order $<^{-1}$, $\langle c_m, c_{m-1}, \dots, c_1 \rangle$. Then,*

1. *$\mathcal{C}$ is an A-preferred MCS, given $<$, iff $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is an L-preferred MSS, given $<^{-1}$.*
2. *$\mathcal{C}$ is an L-preferred MCS, given $<$, iff $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is an A-preferred MSS, given $<^{-1}$.*

Proof. (Sketch)

1. Consider an MSS algorithm that analyzes each clause according to the strict total order $<^{-1}$, starting from c_m and terminating in c_1 , e.g. the Grow procedure from [3,32]. Selected clauses are included in the MSS and discarded clauses are included in the MCS. At step k , $m \geq k \geq 1$, if the algorithm selects the clause c_{m-k+1} to be in the MSS, then it is not included in the MCS. Any MCS that includes c_{m-k+1} (and shares the same clauses for smaller values of k is less preferred than the MCS selected by the algorithm. Moreover, the algorithm adds to the MSS the clauses most preferred given $<^{-1}$, and so the MSS is L-preferred for $<^{-1}$.
2. A similar argument can be used. $\square$

4 Algorithms for Computing Preferred Sets

This section samples some of the best known algorithms for computing preferred sets, namely MUSes, MCSes and MSSes, and investigates which can be used for computing preferred sets, namely L-preferred MSSes and A-preferred MUSes

Table 3: Summary of algorithm analysis

	$\times$	$\checkmark, \langle c_1, \dots, c_m \rangle$	$\checkmark, \langle c_m, \dots, c_1 \rangle$
A-preferred MUS	–	INS, Dicho, QXP	DEL, PRG
A-preferred MCS	MxSAT, CLD	QXP	Grow, PRG
L-preferred MSS	MxSAT, CLD	Grow, PRG	QXP

and MCSes. The analysis of preferred sets algorithms is mostly informal, due to lack of space and due to not including a formal description of the algorithms. The purpose of this section is thus to outline the key insights that explain why some algorithms cannot be used (at least as originally described) for computing preferred sets whereas others (possibly with additional constraints) can. The following algorithms are considered:

- For MUS extraction: (i) Deletion [4,11] (DEL); (ii) Insertion [12] (INS); (iii) Dichotomic [17] (Dicho); (iv) QuickXplain [22] (QXP); and (v) Progression [27] (PRG).
- For MCS & MSS extraction: (i) Grow procedure [3,32] (Grow); (ii) QuickXplain [22,33,14,27]; (iii) Clause D [26] (CLD); (iv) MaxSAT algorithms (e.g. [30,1]) (MxSAT); and (v) Progression [27] (PRG).

The main results of this section are summarized in Table 3. For each type of preferred set, Table 3 indicates which algorithms cannot be used ($\times$) for computing preferred sets and which can ($\checkmark$). Another distinction is the order in which the clauses need to be considered by each algorithm. As shown in Table 3, for all the algorithms considered, the order is either from most to least preferred or from least to most preferred. This is an immediate consequence Proposition 12 and highlights the apparent duality between computing an L-preferred MSS and an A-preferred MCS. The remainder of this section analyzes each of the algorithms listed in Table 3.

It is well-known that MaxSAT algorithms (e.g. [1,30]) can be used for computing MCSes and MSSes. In general MaxSAT algorithms do not analyze clauses according to a strict total order, and so these algorithms cannot be readily applied to computing A-preferred MCSes (or L-preferred MSSes). An alternative would be to analyze the clauses according to a strict total order (e.g. [2]), but this would correspond to an algorithm that analyzes each clause separately, e.g. Grow or Deletion (see below).

The recently proposed Clause D (CLD) algorithm [26] starts from an initial assignment, that satisfies some of the clauses and falsifies the other clauses; clearly, falsified clauses do not have complemented literals and so the algorithm operates on the disjunction of the falsified clauses, attempting to satisfy additional clauses. As shown in Proposition 14 (see list item 3), the removal of satisfied clauses hinders the ability to compute L-preferred or A-preferred sets. As a result, the Clause D algorithm cannot be used for computing an A-preferred MCS (nor an L-preferred MSS).

The Insertion (INS) algorithm for MUS extraction [12] can be organized to compute an A-preferred MUS provided the clauses are analyzed from most to least preferred. The INS algorithm adds clauses to a working set until the working set becomes unsatisfiable. The last clause added to the set is deemed necessary for the MUS and added to a set representing an under-approximation of the MUS. If the clauses are analyzed from most to least preferred, the selected clause is the most preferred among the clauses that could be declared necessary. An MUS that picks a less preferred clause is guaranteed not to be A-preferred. Thus, the INS algorithm computes an A-preferred MUS.

The Dichotomic algorithm for MUS extraction [17] can be organized to compute A-preferred MUS provided the clauses are analyzed from most to least preferred. The rationale is the same as for the Insertion algorithm. The QuickXplain (QXP) algorithm computes A-preferred explanations (MUSes in our context) [22], where clauses are sorted from most to least preferred. The rationale is as follows. The QXP algorithm splits a given target set in two halves and, given its organization, the algorithm is biased towards selecting clauses from the first set (i.e. the most preferred clauses). Moreover, this idea is applied recursively.

The Deletion (DEL) algorithm for MUS extraction [11,4] can be organized to compute an A-preferred MUS provided the clauses are analyzed from least to most preferred. The rationale is that the algorithm preferably drops clauses, and so it gives preference to drop the least preferred clauses. If a clause is deemed necessary for the MUS, then it is necessary for any other MUS given the set of selected clauses, and any other MUSes would have picked least preferred clauses.

The recently proposed Progression (PRG) algorithm for MUS extraction [27, Algorithm 3] can be organized to compute an A-preferred MUS provided the clauses are analyzed from least to most preferred. The operation of the PRG algorithm can be seen as similar to that of the DEL algorithm, with the exception that the set of clauses analyzed in each iteration grows exponentially, and binary search is used to find a necessary clause given the clauses to its *left* (i.e. more preferred clauses). Thus, the rationale is similar to the Deletion algorithm in that the PRG algorithm gives preference to drop the least preferred clauses.

The Grow procedure from [3] can be used for computing L-preferred MSSes provided the clauses are analyzed from most to least preferred. The computed MSS is L-preferred; no other MSS can include clauses that are preferred. By Proposition 12, the Grow procedure can also be used for computing A-preferred MCSes. The QXP algorithm can be adapted for computing A-preferred MCSes (and so also L-preferred MSSes) [33,14,27], being referred to as FastDiag in [14]. We assume the organization for computing minimal sets from [27, Algorithm 1]. The rationale is the same as for the QXP algorithm applied to computing A-preferred MUSes. The organization of the algorithm is biased towards selecting clauses from the first set, which contains the most preferred clauses, and this bias is applied recursively. By Proposition 12, the QXP algorithm can also be used for computing L-preferred MSSes.

The PRG algorithm can be used for computing minimal sets subject to a monotone predicate [27], and so can be used for computing MCSes. The rationale

Table 4: Computing an L-preferred MSS and an A-preferred MCS with Grow [3,32]

L-preferred MSS				A-preferred MCS			
$\mathcal{R}$	c_i	$\mathcal{C}$	Decision	$\mathcal{R}$	c_i	$\mathcal{C}$	Decision
$\langle c_1, \dots, c_7 \rangle$	c_1	$\emptyset$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_1\}$	$\langle c_7, \dots, c_1 \rangle$	c_7	$\emptyset$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_7\}$
$\langle c_2, \dots, c_7 \rangle$	c_2	$\{c_1\}$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_2\}$	$\langle c_6, \dots, c_1 \rangle$	c_6	$\{c_7\}$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_6\}$
$\langle c_3, \dots, c_7 \rangle$	c_3	$\{c_1, c_2\}$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_3\}$	$\langle c_5, \dots, c_1 \rangle$	c_5	$\{c_6, c_7\}$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_5\}$
$\langle c_4, \dots, c_7 \rangle$	c_4	$\{c_1, c_2, c_3\}$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_4\}$	$\langle c_4, \dots, c_1 \rangle$	c_4	$\{c_5, c_6, c_7\}$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_4\}$
$\langle c_5, c_6, c_7 \rangle$	c_5	$\{c_1, \dots, c_4\}$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_5\}$	$\langle c_3, c_2, c_1 \rangle$	c_3	$\{c_4, \dots, c_7\}$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_3\}$
$\langle c_6, c_7 \rangle$	c_6	$\{c_1, \dots, c_5\}$	$\mathcal{C} \leftarrow \mathcal{C}$	$\langle c_2, c_1 \rangle$	c_2	$\{c_3, \dots, c_7\}$	$\mathcal{C} \leftarrow \mathcal{C}$
$\langle c_7 \rangle$	c_7	$\{c_1, \dots, c_5\}$	$\mathcal{C} \leftarrow \mathcal{C}$	$\langle c_1 \rangle$	c_1	$\{c_3, \dots, c_7\}$	$\mathcal{C} \leftarrow \mathcal{C} \cup \{c_1\}$
Final	MSS: $\{c_1, \dots, c_5\}$, MCS: $\{c_6, c_7\}$			Final	MSS: $\{c_1, c_3, \dots, c_7\}$, MCS: $\{c_2\}$		

for why the PRG algorithm computes an A-preferred MCS is the same as for computing an A-preferred MUS. By Proposition 12, the PRG algorithm can also be used for computing L-preferred MSSes.

Example 3. The operation of the Grow procedure [3,32] for computing an L-preferred MSS and an A-preferred MCS is illustrated in Table 4, using the set of clauses and the precedence relation from Example 1. $\mathcal{R}$ represents the set of clauses that the algorithm has yet to analyze. $\mathcal{C}$ represents the set of clauses the algorithm has decided to include in a preferred set. When the Grow procedure terminates, $\mathcal{C}$ represents an MSS of $\mathcal{F}$. Observe that, for the A-preferred MCS, the algorithm exploits Proposition 12.

5 Pruning Techniques for Computing Preferred Sets

This section analyzes techniques commonly used for reducing (or pruning) the number of SAT oracle calls when computing MUSes, MCSes or MSSes. Such pruning techniques are widely regarded as essential in some settings, including MUS and (more recently) MCS extraction. Concrete examples for MUS extraction include clause set refinement [4,5], and model rotation [28,5]. Concrete examples for MCS/MSS extraction include filtering of satisfied clauses [32,26], exploiting backbone literals [26], and disjoint unsatisfiable core identification [26].

We start by establishing a general result about deciding whether to remove or include a set of clauses while computing L-preferred MSSes and A-preferred MUSes/MCSes.

Definition 11 (Deletion-safe & Addition-safe Sets). *Let $\mathcal{F}$ be the set of target soft clauses. Let $\mathcal{S} \subseteq \mathcal{F}$ be the preferred set. Let $\mathcal{D} \subseteq \mathcal{F}$ be a set of clauses. If $\mathcal{D} \cap \mathcal{S} = \emptyset$, then $\mathcal{D}$ is deletion-safe. If $\mathcal{D} \subseteq \mathcal{S}$, then $\mathcal{D}$ is addition-safe.*

Let $\mathcal{F}$ be the set of target soft clauses, and let $\mathcal{S} \subseteq \mathcal{F}$ be a preferred set. Consider the execution of a preferred set algorithm, where $\mathcal{C} \subseteq \mathcal{S}$ represents the clauses from $\mathcal{S}$ already identified by the algorithm, and $\mathcal{R} \subseteq \mathcal{F}$ represents the clauses not yet decided to be included or not in a preferred set. Now, if $\mathcal{D} \subseteq \mathcal{R}$ is

deletion-safe, then $\mathcal{R}$ can be set to $\mathcal{R} \setminus \mathcal{D}$. Clearly, by definition of deletion-safe set, $\mathcal{D} \cap \mathcal{S} = \emptyset$, the clauses in $\mathcal{D}$ cannot be added to $\mathcal{S}$ and so can be dropped from $\mathcal{R}$. Similarly, if $\mathcal{D} \subseteq \mathcal{R}$ is addition-safe, then $\mathcal{R}$ can be refined to $\mathcal{R} \setminus \mathcal{D}$, and $\mathcal{C}$ be extended to $\mathcal{C} \cup \mathcal{D}$. By definition of addition-safe set, $\mathcal{D} \subseteq \mathcal{S}$ and so the clauses in $\mathcal{D}$ can be added to $\mathcal{C}$ and dropped from $\mathcal{R}$. Thus the following holds.

Proposition 13. *Given $\mathcal{F}$, $\mathcal{S}$, $\mathcal{C}$ and $\mathcal{R}$ as above, and $\mathcal{D} \subseteq \mathcal{R}$, then*

- *If $\mathcal{D}$ is deletion-safe, then $\mathcal{S} \subseteq \mathcal{C} \cup (\mathcal{R} \setminus \mathcal{D})$ and $\mathcal{R}$ can be updated to $\mathcal{R} \setminus \mathcal{D}$.*
- *If $\mathcal{D}$ is addition-safe, then $\mathcal{S} \subseteq (\mathcal{C} \cup \mathcal{D}) \cup (\mathcal{R} \setminus \mathcal{D})$, $\mathcal{R}$ can be updated to $\mathcal{R} \setminus \mathcal{D}$, and $\mathcal{C}$ updated to $\mathcal{C} \cup \mathcal{D}$.*

Depending on the pruning technique considered, the computed sets of clauses may or may not be deletion-safe or addition-safe. Some of the most well-known pruning techniques used for computing either MUSes, MCSes or MSSes are analyzed in the following proposition.

Proposition 14. *The following holds:*

1. *Clauses not included in an unsatisfiable core (i.e. the technique of clause set refinement [4,28]) are not guaranteed to be deletion-safe for A-preferred MUSes.*
2. *Clauses identified by model rotation [28,5] are addition-safe for A-preferred MUSes.*
3. *Clauses satisfied by some initial truth assignment (e.g. [32]) are not guaranteed to be addition-safe for L-preferred MSSes and for A-preferred MCSes.*
4. *Clauses satisfied (resp. falsified) by backbone literals [26] are addition-safe (resp. deletion-safe) for L-preferred MSSes and deletion-safe (resp. addition-safe) for A-preferred MCSes.*
5. *The identification of disjoint unsatisfiable cores [26] (and so the satisfied clauses that eventually result) are not guaranteed to be deletion-safe for A-preferred MCSes (and for L-preferred MSSes).*

Proof. (Sketch)

1. Given a computed unsatisfiable core, the clauses not in the core are not guaranteed to respect the strict total order of clauses, and so these clauses are not deletion-safe for A-preferred MUSes.
2. Model rotation identifies clauses that *must* be included in any MUS given that some other clauses are already included in an MUS. Since these clauses *must* be included in any MUS, they are addition-safe for A-preferred MUSes.
3. The clauses satisfied by an initial assignment need not respect the strict partial order of clauses, and so these clauses are not guaranteed to be addition-safe (for L-preferred MSSes) nor deletion-safe (for A-preferred MCSes).
4. Backbone literals represent literals that take a fixed value for some clauses in an MCS to be falsified. Any clause that is satisfied (resp. falsified) by these backbone literals cannot be included in any MCS (resp. MSS) and so must be included in some MSS (resp. MCS).
5. Unsatisfiable cores can be iteratively removed until a remaining set of clauses is satisfied. As above, these satisfied clauses need not respect the strict partial

order of clauses, and so these clauses are not guaranteed to be addition-safe (for L-preferred MSSes) nor deletion-safe (for A-preferred MCSes). $\square$

Example 4. Consider the clauses in Example 1, and the truth assignment $\{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1\}$, which satisfies the clauses $\{c_2, c_3, c_5, c_6, c_7\}$. Clearly, removing these clauses from the set of clauses to consider for the MCS prevents finding the A-preferred MCS $\{c_2\}$. Similarly, if the computed unsatisfiable core is $\{c_1, c_2, c_7\}$, then clause set refinement would drop the other clauses, thus preventing the identification of the A-preferred MUS $\{c_2, c_4, c_5, c_6\}$.

6 Assessing Practical Impact

The negative results of the previous sections impose important restrictions on the type of preferred sets we can expect to be able to compute efficiently. Given the objective of using a polynomial number of calls to a SAT oracle (as current algorithms do), one can only consider L-preferred MSSes and A-preferred MUSes and MCSes, as was also proposed in [22]. Moreover, the paper also identifies restrictions on the type of algorithms and techniques that can be used. As a result, the computation of preferred sets is expected to be significantly less efficient in practice than what is currently achieved by state of the art algorithms.

To evaluate the effect of the limitations identified in the previous sections, a set of 79 (non-trimmed) instances from the MUS track of the 2011 SAT competition³ were selected, such that the default run time of MUSer [5], i.e. with clause set refinement, did not exceed 50s. The experiments were run on a HPC cluster with 2xIntel E5-2620 @2GHz (6 core) nodes, each with 128 GB of memory, with a timeout of 3600 seconds. Figure 1 shows a scatter plot comparing the run times of MUSer with and without clause set refinement. (Observe that these represent plain MUS extraction instances, with no preferences specified.) As the plot shows, the performance difference is significant; for all instances the use of clause refinement improves the running time. In addition, without clause set refinement MUSer is unable to solve 46 instances (out of 79). For most of the instances solved, the run times increase by more than one order magnitude. As can be concluded, the limitation of not using clause set refinement has a significant impact in the ability of MUSer to solve problem instances within the given timeout. The main conclusion is that the computation of A-preferred MUSes (and similarly of A-preferred MCSes and L-preferred MSSes) should be restricted to small instances.

7 Discussion & Conclusions

In some settings, there exist preferences over which MUSes, MCSes or MSSes to compute. This paper revisits well-known definitions of preferred constraints and preferred explanations and relaxations for over-constrained sets of constraints [22], namely lexicographic (L-) and anti-lexicographic (A-) preferred

³ <http://www.satcompetition.org/2011/>.

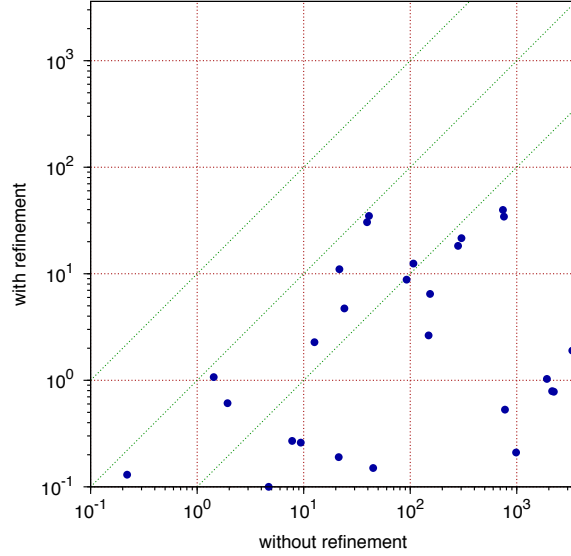


Fig. 1: Scatter plot for MUSer with and without clause set refinement

sets, and casts these definitions in the setting of unsatisfiable CNF formulas, namely MUSes, MCSes & MSSes. The paper shows that computing L-preferred MUSes/MCSes and A-preferred MSSes is hard for the second level of the polynomial hierarchy, whereas computing A-preferred MUSes/MCSes and L-preferred MSSes is in FP^{NP} , i.e. can be solved with a polynomial number of calls to an NP oracle. The practical significance of these results is that in practice, for efficiency reasons, a less natural definition of preferred set is used when computing MUSes or MCSes.

In addition, the paper investigates well-known algorithms for computing MUSes, MCSes and MSSes, and shows that some algorithms cannot be used for computing preferred sets. For example, the recently proposed CLD algorithm [26] cannot be used for computing preferred MCSes or MSSes. Moreover, the paper also investigates well-known, widely used, techniques for reducing the number of SAT solver calls, and shows that several of these techniques cannot be used when computing preferred sets. The consequences of these negative results are significant, since these prevent using well-known very effective pruning techniques when computing preferred sets. Experimental results on selected MUS problem instances illustrate the negative impact of this limitation.

A number of research directions can be envisioned. A natural line of research consists of analyzing other algorithms as well as other techniques for computing preferred sets. Moreover, a more comprehensive experimental evaluation will provide a better understanding of the practical consequences of the negative results proved in this paper. As a result, a main line of research is to develop practical solutions for computing preferred MUSes, MCSes and MSSes, that mitigate the negative results in this paper.

References

1. C. Ansótegui, M. L. Bonet, and J. Levy. SAT-based MaxSAT algorithms. *Artif. Intell.*, 196:77–105, 2013.
2. J. Argelich, I. Lynce, and J. Marques-Silva. On solving boolean multilevel optimization problems. In C. Boutilier, editor, *IJCAI*, pages 393–398, 2009.
3. J. Bailey and P. J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In M. V. Hermenegildo and D. Cabeza, editors, *PADL*, volume 3350 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2005.
4. R. R. Bakker, F. Dikker, F. Tempelman, and P. M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. In R. Bajcsy, editor, *IJCAI*, pages 276–281. Morgan Kaufmann, 1993.
5. A. Belov, I. Lynce, and J. Marques-Silva. Towards efficient MUS extraction. *AI Commun.*, 25(2):97–116, 2012.
6. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2009.
7. E. Birnbaum and E. L. Lozinskii. Consistent subsets of inconsistent systems: structure and behaviour. *J. Exp. Theor. Artif. Intell.*, 15(1):25–46, 2003.
8. C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res. (JAIR)*, 21:135–191, 2004.
9. C. Cayrol, M.-C. Lagasquie-Schiex, and T. Schiex. Nonmonotonic reasoning: From complexity to algorithms. *Ann. Math. Artif. Intell.*, 22(3-4):207–236, 1998.
10. Y. Chen, S. Safarpour, J. Marques-Silva, and A. G. Veneris. Automated design debugging with maximum satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 29(11):1804–1817, 2010.
11. J. W. Chinneck and E. W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *INFORMS Journal on Computing*, 3(2):157–168, 1991.
12. J. L. de Siqueira N. and J.-F. Puget. Explanation-based generalisation of failures. In *ECAI*, pages 339–344, 1988.
13. E. Di Rosa and E. Giunchiglia. Combining approaches for solving satisfiability problems with qualitative preferences. *AI Commun.*, 26(4):395–408, 2013.
14. A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, 26(1):53–62, 2012.
15. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
16. É. Grégoire, B. Mazure, and C. Piette. On approaches to explaining infeasibility of sets of Boolean clauses. In *ICTAI (1)*, pages 74–83. IEEE Press, 2008.
17. F. Hemery, C. Lecoutre, L. Sais, and F. Boussemart. Extracting MUCs from constraint networks. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *ECAI*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 113–117. IOS Press, 2006.
18. A. Ignatiev, M. Janota, and J. Marques-Silva. Towards efficient optimization in package management systems. In *ICSE*, May 2014.
19. M. Jampel, E. C. Freuder, and M. J. Maher, editors. *Over-Constrained Systems*, volume 1106 of *Lecture Notes in Computer Science*. Springer, 1996.
20. M. Janota, G. Botterweck, and J. Marques-Silva. On lazy and eager interactive reconfiguration. In P. Collet, A. Wasowski, and T. Weyer, editors, *VaMoS*. ACM, 2014.

21. M. Jose and R. Majumdar. Cause clue clauses: error localization using maximum satisfiability. In M. W. Hall and D. A. Padua, editors, *PLDI*, pages 437–446. ACM, 2011.
22. U. Junker. QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In D. L. McGuinness and G. Ferguson, editors, *AAAI*, pages 167–172. AAAI Press / The MIT Press, 2004.
23. A. Komuravelli, A. Gurfinkel, S. Chaki, and E. M. Clarke. Automatic abstraction in SMT-based unbounded software model checking. In Sharygina and Veith [37], pages 846–862.
24. J.-M. Lagniez and A. Biere. Factoring out assumptions to speed up MUS extraction. In M. Järvisalo and A. V. Gelder, editors, *SAT*, volume 7962 of *Lecture Notes in Computer Science*, pages 276–292. Springer, 2013.
25. P. Liberatore. Redundancy in logic I: CNF propositional formulae. *Artif. Intell.*, 163(2):203–232, 2005.
26. J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. On computing minimal correction subsets. In F. Rossi, editor, *IJCAI. IJCAI/AAAI*, 2013.
27. J. Marques-Silva, M. Janota, and A. Belov. Minimal sets over monotone predicates in boolean formulae. In Sharygina and Veith [37], pages 592–607.
28. J. Marques-Silva and I. Lynce. On improving MUS extraction algorithms. In K. A. Sakallah and L. Simon, editors, *SAT*, volume 6695 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2011.
29. P. Meseguer, N. Bouhmala, T. Bouzoubaa, M. Irgens, and M. Sánchez. Current approaches for solving over-constrained problems. *Constraints*, 8(1):9–39, 2003.
30. 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.
31. A. Nadel, V. Ryvchin, and O. Strichman. Efficient MUS extraction with resolution. In *FMCAD*, pages 197–200. IEEE, 2013.
32. A. Nöhrrer, A. Biere, and A. Egyed. Managing SAT inconsistencies with HUMUS. In U. W. Eisenecker, S. Apel, and S. Gnesi, editors, *VaMoS*, pages 83–91. ACM, 2012.
33. B. O’Callaghan, B. O’Sullivan, and E. C. Freuder. Generating corrective explanations for interactive constraint satisfaction. In P. van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 445–459. Springer, 2005.
34. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1993.
35. R. Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.
36. F. Rossi, K. B. Venable, and T. Walsh. *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*. Morgan & Claypool Publishers, 2011.
37. N. Sharygina and H. Veith, editors. *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*. Springer, 2013.
38. W.-J. van Hoeve. *Hybrid Optimization: the 10 years of CPAIOR*, chapter Over-Constrained Problems, pages 191–225. Springer, 2011.
39. J. P. Wallner, G. Weissenbacher, and S. Woltran. Advanced SAT techniques for abstract argumentation. In J. Leite, T. C. Son, P. Torroni, L. van der Torre, and S. Woltran, editors, *CLIMA*, volume 8143 of *Lecture Notes in Computer Science*, pages 138–154. Springer, 2013.

40. C. S. Zhu, G. Weissenbacher, and S. Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In P. Bjesse and A. Slobodová, editors, *FMCAD*, pages 63–66. FMCAD Inc., 2011.