On Computing Minimal Correction Subsets

Joao Marques-Silva
CASL/UCD, Ireland
IST/INESC-ID, Portugal
jpms@ucd.ie

Federico Heras
CASL/UCD, Ireland
federico.heras@ucd.ie

Mikolas Janota
INESC-ID, Portugal
mikolas.janota@gmail.com

Alessandro Previti
CASL/UCD, Ireland
alessandro.previti@ucdconnect.ie

Anton Belov
CASL/UCD, Ireland
anton.belov@ucd.ie

Abstract

A set of constraints that cannot be simultaneously satisfied is over-constrained. Minimal relaxations and minimal explanations for over-constrained problems find many practical uses. For Boolean formulas, minimal relaxations of over-constrained problems are referred to as Minimal Correction Subsets (MCSes). MCSes find many applications, including the enumeration of MUSes. Existing approaches for computing MCSes either use a Maximum Satisfiability (MaxSAT) solver or iterative calls to a Boolean Satisfiability (SAT) solver. This paper shows that existing algorithms for MCS computation can be inefficient, and so inadequate, in certain practical settings. To address this problem, this paper develops a number of novel techniques for improving the performance of existing MCS computation algorithms. More importantly, the paper proposes a novel algorithm for computing MCSes. Both the techniques and the algorithm are evaluated empirically on representative problem instances, and are shown to yield the most efficient and robust solutions for MCS computation.

1 Introduction

A set of constraints is *over-constrained* if the constraints cannot be simultaneously satisfied [Meseguer *et al.*, 2003]. For over-constrained problems, one is often interested in finding a maximal set of constraints that can be satisfied or, equivalently, a minimal set of constraints that need to be relaxed (among those constraints that can be relaxed) (e.g. [Junker, 2004]). A related objective is to compute a minimal subset of constraints that *explains* why the initial set of constraints cannot be simultaneously satisfied. For Boolean formulas, the former are referred to as Minimal Correction Subsets (MCSes) and the latter as Minimally Unsatisfiable Subsets (MUSes) (e.g. [Liffiton and Sakallah, 2008])[1]. Motivated by the seminal work of Reiter [Reiter, 1987], several important results relating MCSes and MUSes have been established [Birnbaum and Lozinskii, 2003; Bailey and Stuckey, 2005; Liffiton and Sakallah, 2008], including the well-known minimal hitting set duality between MCSes and MUSes.

The identification of MCSes finds many practical applications. For example, the MCS with the smallest number of clauses (or smallest weight) represents the solution of the Maximum Satisfiability (MaxSAT) problem. Another example is to enumerate MCSes and use minimal hitting set duality for computing the MUSes of a formula (e.g. [Bailey and Stuckey, 2005; Liffiton and Sakallah, 2008]). One additional example is the computation of minimal models (e.g. [Ben-Eliyahu and Dechter, 1996; Soh and Inoue, 2010]). This paper shows that MCSes can also be used for effectively approximating the solution of the MaxSAT problem.

Different approaches have been proposed over the years for computing MCSes. Earlier work proposed modifications to the basic Boolean Satisfiability (SAT) DPLL algorithm [Birnbaum and Lozinskii, 2003]. This work was later shown to be inefficient in practice when compared with an approach based on MaxSAT [Liffiton and Sakallah, 2008]. A different solution consists of iteratively calling a SAT solver [Bailey and Stuckey, 2005]. Results in [Liffiton and Sakallah, 2008] indicate that the MaxSAT approach is more efficient. However, recent work [Nöhrer *et al.*, 2012] proposes important optimizations. An alternative approach was recently developed in the context of model-based diagnosis [Felfernig *et al.*, 2012]. This recent algorithm, FASTDIAG, adapts the QUICKXPLAIN algorithm [Junker, 2004] for finding MUSes. Finally, MCSes can also be computed with SAT with preferences, e.g. [Rosa *et al.*, 2010]. As the experimental results in this paper show, although acceptably efficient for MCS enumeration, existing MCS computation algorithms can be fairly inefficient when computing single MCSes for harder instances. A single MCS can, for example, be used for approximating the solution for very hard instances of MaxSAT, i.e. instances for which tight upper bounds (on the cost of falsified clauses) are currently unknown. In these cases, existing algorithms for computing MCSes are ineffective.

This paper develops new techniques for improving recent

[1]Although this paper addresses Boolean formulas, the work can be lifted to more general constraints.

MCS computation algorithms [Bailey and Stuckey, 2005; Nöhrer *et al.*, 2012; Felfernig *et al.*, 2012]. Moreover, the paper develops a new algorithm for computing MCSes that is inspired by the work of [Birnbaum and Lozinskii, 2003]. This new algorithm also exploits the proposed novel techniques. The paper conducts an extensive experimental evaluation, organized according to three different scenarios. The first scenario evaluates raw performance when computing a single MCS. The second scenario evaluates how each algorithm performs when enumerating all MCSes. To conclude, the third scenario evaluates the enumeration of MCSes (within a time bound) to approximate MaxSAT. The experimental results demonstrate that the new algorithms significantly outperform the existing state of the art in several ways. First, the new algorithms allow computing MCSes for problem instances for which existing approaches are unable to. Second, the new algorithms are capable of enumerating all the MCSes for problem instances for which existing approaches are unable to. Finally, the new algorithms are shown to consistently outperform incomplete algorithms for MaxSAT, providing better quality bounds than what had been achieved until now.

The paper is organized as follows. Section 2 introduces the definitions used throughout the paper. A brief overview of algorithms for computing MCSes is provided in Section 3. Sections 4 and 5 present the main contributions of the paper. The experimental evaluation is presented in Section 6. The paper concludes in Section 7.

2 Preliminaries

Standard definitions for propositional logic are assumed (e.g. [Biere *et al.*, 2009]). Formulas are defined over a set of Boolean variables $X = \{x_1, \ldots, x_n\}$, and are represented in Conjunctive Normal Form (CNF). A CNF formula is a conjunction of disjunctions of literals (clauses), and a literal is either a variable or its negation. A CNF formula can also be interpreted as a set of sets of literals. Both representations are used interchangeably. CNF formulas are denoted by letters with calligraphic fonts, e.g. $\mathcal{F}$, $\mathcal{S}$, $\mathcal{U}$, $\mathcal{C}$, $\mathcal{T}$, etc. When necessary, subscripts are used. Clauses are represented by c or c_i, $i = 1, \ldots, m$, where $m = |\mathcal{F}|$. For a formula $\mathcal{T}$ and a range $i..j$, where $\mathcal{T}$ is viewed as a sequence $\langle c_1, c_2, \ldots, c_{|\mathcal{T}|}\rangle$, $\mathcal{T}_{i..j}$ denotes the clauses in the range $i..j$, i.e. $\{c_i, c_{i+1}, \ldots, c_j\}$ for $j \geq i$, or $\emptyset$ if $j < i$.

A truth assignment is a mapping $\mu : X \to \{0, 1\}$. In this paper assignments are always complete, i.e. the mapping is total. Assignments are extended to clauses and to formulas, where the valuation of a clause (or a formula) is obtained by standard logic operations. Thus, $\mu(c)$ and $\mu(\mathcal{F})$ denote, respectively, the valuation of clause c and of formula $\mathcal{F}$.

The algorithms described in the paper use extensively SAT solver calls. These can be viewed as calls to a SAT oracle that can return more information than the standard true/false outcome. A SAT solver call is denoted by $(st, \mu, \mathcal{V}) \leftarrow \text{SAT}(\mathcal{F})$, where *st* denotes the outcome (i.e. true or false) and the remaining (optional) arguments denote, respectively, the computed model (if the outcome is true) and the computed unsatisfiable subformula (if the outcome is false).

The following definitions serve to characterize unsatisfiable subformulas, and are used throughout.

```
Function BLS ($\mathcal{F}$)
    Global: none
1   $(\mathcal{S},\mathcal{U}) \leftarrow$ InitialAssignment($\mathcal{F}$)
2   foreach $c \in \mathcal{U}$ do
3       if SAT($\mathcal{S} \cup \{c\}$) then $\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$
4
5   return $\mathcal{F} \setminus \mathcal{S}$                    // MCS of $\mathcal{F}$
```

Algorithm 1: Basic linear search (BLS)

Definition 1 *$\mathcal{M} \subseteq \mathcal{F}$ is a* Minimally Unsatisfiable Subset *(MUS) of $\mathcal{F}$ iff $\mathcal{M}$ is unsatisfiable and $\forall c \in \mathcal{M}, \mathcal{M} \setminus \{c\}$ is satisfiable.*

Definition 2 *$\mathcal{C} \subseteq \mathcal{F}$ is a* Minimal Correction Subset *(MCS) iff $\mathcal{F} \setminus \mathcal{C}$ is satisfiable and $\forall c \in \mathcal{C}, \mathcal{F} \setminus (\mathcal{C} \setminus \{c\})$ is unsatisfiable.*

Definition 3 *$\mathcal{S} \subseteq \mathcal{F}$ is a* Maximal Satisfiable Subset *(MSS) iff $\mathcal{S}$ is satisfiable and $\forall c \in \mathcal{F} \setminus \mathcal{S}, \mathcal{F} \cup \{c\}$ is unsatisfiable.*

It is well-known that for an MCS $\mathcal{C}$, $\mathcal{F} \setminus \mathcal{C}$ denotes an MSS of $\mathcal{F}$ (e.g. [Liffiton and Sakallah, 2008]). Given a formula $\mathcal{F}$, an unsatisfiable subformula (or core) [Zhang and Malik, 2003] of $\mathcal{F}$ is any subset $\mathcal{U}$ of $\mathcal{F}$ that is also unsatisfiable. Hence, an unsatisfiable core contains one or more MUSes.

The MaxSAT problem consists of finding an assignment that satisfies the *maximum* number of clauses of an unsatisfiable formula. Thus, a largest MSS represents a solution to the MaxSAT problem, which can also be represented as the smallest MCS. More general variants of the MaxSAT problem consider *hard* clauses (that must be satisfied) or *weighted* clauses (incurring a cost if not satisfied) (e.g. [Li and Manyà, 2009]). The algorithms described in this paper address CNF formulas where all clauses are *soft*, i.e. can be falsified (or *relaxed*). However, extensions to the more general case of formulas with hard (or weighted) clauses are immediate, and are highlighted when necessary.

3 Related Work

This section overviews work related with (or easily adapted to) computing MCSes of CNF formulas. A review of the large body of related work on model-based diagnosis (e.g. [Reiter, 1987] or, for recent references [Felfernig *et al.*, 2012]) is omitted due to space constraints. In the area of constraints, the analysis of over-constrained problems (with soft constraints) is often associated with solving MaxCSP (e.g. [Jampel *et al.*, 1996; Meseguer *et al.*, 2003; 2006; van Hoeve, 2011]).

A simple approach for computing one MCS consists of a (linear) search through the clauses of the formula [Bailey and Stuckey, 2005]. This approach corresponds to Algorithm 1, where line 1 is replaced by the assignment $\mathcal{U} \leftarrow \mathcal{F}$. An immediate improvement is to start from an assignment that (preferably) falsifies as few clauses as possible [Nöhrer *et al.*, 2012]. This is illustrated by line 1 in Algorithm 1. Another improvement consists of adding to $\mathcal{S}$ all clauses satisfied at each SAT solver call [Nöhrer *et al.*, 2012]. This improvement is analyzed in greater detail in Section 4. In the worst case, linear search requires $\mathcal{O}(m)$ calls to a SAT solver. Observe that the extension to partial MaxSAT is easily achieved

```
Function BFD ($\mathcal{R}, \mathcal{T} \subseteq \mathcal{R}$, hasD)
    Global: none
    if $hasD \wedge \mathtt{SAT}(\mathcal{R})$ then return $\emptyset$

    if $|\mathcal{T}| = 1$ then return $\mathcal{T}$

    $m \leftarrow \lfloor \frac{|\mathcal{T}|}{2} \rfloor$
    $(\mathcal{T}_1, \mathcal{T}_2) \leftarrow (\mathcal{T}_{1..m}, \mathcal{T}_{m+1..|\mathcal{T}|})$
    $\mathcal{D}_2 \leftarrow \mathrm{BFD}(\mathcal{R} \setminus \mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_1 \neq \emptyset)$
    $\mathcal{D}_1 \leftarrow \mathrm{BFD}(\mathcal{R} \setminus \mathcal{D}_2, \mathcal{T}_1, \mathcal{D}_2 \neq \emptyset)$
    return $\mathcal{D}_1 \cup \mathcal{D}_2$            // Clauses of MCS
```

Algorithm 2: Basic FASTDIAG (BFD)

by requiring the hard clauses to be included in set $\mathcal{S}$ initially. Thus, line 1 in Algorithm 1 involves a SAT solver call to guarantee that the hard clauses are satisfied. A similar solution is used in all algorithms described in the following sections. Moreover, enumeration of MCSes is implemented by *blocking* each computed MCS, with a hard clause that disables repeating the same MCS. This hard clause is defined as the disjunction of all the literals in the clauses of the MCS (see Section 5 for a justification). The same approach is assumed in all of the MCS algorithms described in this paper.

MaxSAT [Liffiton and Sakallah, 2008] represents the most widely used approach for computing MCSes, e.g. for MUS enumeration. In this approach, a new relaxation variable is added to each clause. These variables are then used for iteratively reducing the cost of the relaxation variables assigned value 1 (i.e. that relax clauses). This procedure terminates when the cost of the relaxation variables assigned value 1 cannot be further reduced. This represents a MaxSAT solution, and so it is an MCS of the original formula. Recent improvements to MaxSAT-based MCS enumeration are reported in [Morgado *et al.*, 2012]. Despite very good results for MCS enumeration, the use of MaxSAT for computing MCSes has a few drawbacks. For example, since this approach requires *solving* MaxSAT, it is inadequate for *approximating* the solution of hard MaxSAT instances.

Recent work in model-based diagnosis proposed the FASTDIAG algorithm [Felfernig *et al.*, 2012]. This algorithm mimics the well-known QUICKXPLAIN algorithm [Junker, 2004]. However, whereas QUICKXPLAIN computes explanations, FASTDIAG computes diagnoses (or relaxations). Algorithm 2 details the organization of FASTDIAG. The algorithm accepts three arguments. The first argument is a set of clauses one aims to maximally satisfy, i.e. the reference set. The second argument represents the clauses that can be extracted when attempting to satisfy the clauses in the first argument, i.e. the target set. Finally, the third argument indicates whether a valid correction set already exists. If it does, then the clauses in the reference set can be checked for satisfiability. For plain MaxSAT, Algorithm 2 is initially called with sets $\mathcal{R}$ and $\mathcal{T}$ corresponding to all clauses in the formula. In contrast, for partial MaxSAT, the algorithm is initially called with set $\mathcal{R}$ corresponding to both the hard and soft clauses of the formula, and set $\mathcal{T}$ corresponding solely to the soft clauses. A similar solution is used in any the variant of the BFD algorithm described in the following sections.

```
Function DISJOINTUNSATISFIABLECORES ($\mathcal{F}$)
    Output: $\mu$: Assignment; $\mathcal{F}$: Subformula; $U$: Disjoint cores
    $(U, st) \leftarrow (\emptyset, \mathsf{false})$
    while not st do
        $(st, \mu, \mathcal{V}) \leftarrow \mathtt{SAT}(\mathcal{F})$
        if not st then
            $\mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{V}$
            $U \leftarrow U \cup \{\mathcal{V}\}$
    return $(\mu, \mathcal{F}, U)$
```

Algorithm 3: Computing disjoint unsatisfiable cores

This algorithm has the same complexity as QUICKXPLAIN. If k is the size of the *largest* MCS, then the algorithm requires $\mathcal{O}(k + k \log \frac{m}{k})$ calls to a SAT solver in the worst case, where $m = |\mathcal{F}|$. If the largest MCS is much smaller than the original formula (and often it is), then FASTDIAG can be expected to outperform linear search.

Approaches for computing MCSes that do not use repeated calls to a SAT solver are either based on modifying the DPLL SAT solver [Birnbaum and Lozinskii, 2003], or by adapting a CDCL SAT solver for implementing SAT with preferences [Rosa *et al.*, 2010]. The work in [Birnbaum and Lozinskii, 2003] requires a fixed order of the variables, and standard SAT techniques cannot be used. These approaches were shown to be quite inefficient in [Liffiton and Sakallah, 2008]. The results in Section 6 show that the use of SAT with preferences is also ineffective in practice.

4 New MCS Techniques

This section develops three techniques that serve to reduce the number of calls to a SAT solver and to simplify each call. The first one is based on finding a *disjoint set of unsatisfiable cores*. The second one exploits *backbone literals* (e.g. [Kilby *et al.*, 2005]. Finally, the third one exploits clauses satisfied by satisfying assignments. The third technique is used in the recently proposed PICOMCS algorithm [Nöhrer *et al.*, 2012]. However, we show that the technique can be used with other algorithms for computing MCSes.

Algorithm 3 shows how a set of disjoint unsatisfiable cores can be computed. The following result is used when exploiting disjoint unsatisfiable cores for computing MCSes.

Proposition 1 *Let $\mathcal{F}$ be unsatisfiable, and let P denote a partition of $\mathcal{F}$ as follows, $P = \{\mathcal{G}, \mathcal{C}_1, \ldots \mathcal{C}_r\}$, where each $\mathcal{C}_i$, $i = 1, \ldots, r$ is an unsatisfiable core of $\mathcal{F}$. Then, r represents a lower bound on the size of any MCS of $\mathcal{F}$.*

Proof. Since P is a partition of $\mathcal{F}$, the unsatisfiable cores are *disjoint*. Given any unsatisfiable core $\mathcal{C}_i$, any MCS $\mathcal{M}$ of $\mathcal{F}$ must include at least one clause from each $\mathcal{C}_i$, as otherwise $\mathcal{C}_i \subseteq \mathcal{F} \setminus \mathcal{M}$, and so $\mathcal{F} \setminus \mathcal{M} \vDash \bot$. Therefore, an MCS of $\mathcal{F}$ must include at least r clauses, and so r denotes a lower bound on the size of any MCS of $\mathcal{F}$. □

Consider an unsatisfiable formula $\mathcal{F}$ and an assignment μ. Then μ induces a partition of $\mathcal{F}$, $\{\mathcal{S}, \mathcal{U}\}$, where $\mathcal{S}$ denotes the satisfied clauses and $\mathcal{U}$ denotes the falsified clauses. Thus,

Proposition 2 *There exist an MSS $\mathcal{W}$ and an MCS $\mathcal{M}$ of $\mathcal{F}$ such that $\mathcal{S} \subseteq \mathcal{W}$ and $\mathcal{U} \supseteq \mathcal{M}$.*

If $\mathcal{F}$ is partitioned into $\{\mathcal{G}, \mathcal{C}_1, \ldots, \mathcal{C}_r\}$, such that $\mathcal{G}$ is satisfiable, and $\mathcal{C}_1, \ldots, \mathcal{C}_r$ are disjoint unsatisfiable cores, then any assignment μ that satisfies $\mathcal{G}$ partitions each unsatisfiable core $\mathcal{C}_i$ into two disjoint sets $\mathcal{S}_i$ and $\mathcal{U}_i$ of satisfied and falsified clauses, respectively. Let $\mathcal{S} = \mathcal{G} \cup \cup_{i=1}^{r} \mathcal{S}_i$, with $\mathcal{S} \subseteq \mathcal{F}$, denote the clauses satisfied by some assignment μ. Clearly, by Proposition 2, $\mathcal{S} \subseteq \mathcal{W}$ where $\mathcal{W}$ is an MSS of $\mathcal{F}$. Set $\mathcal{S}$ is to be iteratively updated with additional clauses. Let $\mathcal{S}^q$ denote the q^{th} update, $\mathcal{S} \equiv \mathcal{S}^0 \subset \mathcal{S}^1 \subset \ldots \subset \mathcal{S}^q \subset \ldots \subseteq \mathcal{W}$, with the invariant that each set $\mathcal{S}^q$ is satisfiable, and so represents a subset of an MSS $\mathcal{W}$ of $\mathcal{F}$. The set of falsified clauses from unsatisfiable core $\mathcal{C}_1$ is $\mathcal{U}_1$ (with $\mathcal{U}_1 \neq \emptyset$, since $\mathcal{C}_1$ is unsatisfiable). Next, consider each clause c in $\mathcal{U}_1$, taking into account that the clauses in $\mathcal{S}^q$ must be satisfied. If $\mathcal{S}^q \cup \{c\}$ is satisfiable, then $\mathcal{S}^{q+1} = \mathcal{S}^q \cup \{c\}$. Otherwise, since $\mathcal{S}^q \cup \{c\} \vDash \bot$, c *must* be included in any MCS $\mathcal{M}$ of $\mathcal{F}$, such that the MSS $\mathcal{W} = \mathcal{F} \setminus \mathcal{M}$ contains $\mathcal{S}^q$. After analyzing all clauses in $\mathcal{U}_1$, the clauses not added to some set $\mathcal{S}^q$ must be included in any MCS $\mathcal{M}$ of $\mathcal{F}$ such that the MSS $\mathcal{W} = \mathcal{F} \setminus \mathcal{M}$ contains $\mathcal{S}^q$. The process is repeated for each unsatisfiable core, at each step either updating some satisfiable set $\mathcal{S}^q$ with a clause c, and so $\mathcal{S}^{q+1} = \mathcal{S}^q \cup \{c\}$, such that $\mathcal{S}^{q+1}$ is satisfiable, or declaring c to be included in an MCS of $\mathcal{F}$. Since any MCS $\mathcal{F}$ integrates at least one clause from each unsatisfiable core (Proposition 1), the above process can be terminated when $|\mathcal{U}_i| = 1$, thus saving one SAT solver call. Each unsatisfiable core is said to be *corrected locally*, since it is analyzed separately. Given the above, disjoint unsatisfiable cores can be used for two purposes. First, to provide a lower bound on the size of any MCS. Second, an MCS can be constructed by correcting locally each unsatisfiable core. More importantly, *all* the algorithms described in this paper can either work on the complete formula $\mathcal{F}$ or on each core.

For enumerating MCSes, Algorithm 3 can be used iteratively, being required to satisfy the *blocking* clauses that prevent repeated MCSes (see Section 3). After enumerating all the MCSes identified by a set of disjoint unsatisfiable cores, additional disjoint unsatisfiable cores are computed, but previously computed ones are kept. The algorithm terminates when a computed unsatisfiable core is empty. This indicates that all MCSes have been enumerated. Moreover, the approach for using disjoint unsatisfiable cores for computing the next MCS remains unchanged.

The second technique exploits *backbone literals* [Kilby *et al.*, 2005]. Let $\mathcal{M}$ be an MCS of $\mathcal{F}$. Then, by definition of MCS, for any clause $c \in \mathcal{M}$, $c = (l_1 \vee l_2 \vee \ldots \vee l_k)$, $(\mathcal{F} \setminus \mathcal{M}) \cup \{c\} \vDash \bot$. Hence, $\mathcal{F} \setminus \mathcal{M} \vDash \bar{c}$, i.e. for each literal $l_i \in c$, $\mathcal{F} \setminus \mathcal{M} \vDash \neg l_i$. Thus, $\neg l_i$ is a backbone literal of $\mathcal{F} \setminus \mathcal{M}$. Let $\mathcal{L}$ denote the set of clauses decided to be included in some MCS of $\mathcal{F}$. Then for every MCS $\mathcal{M}$ of $\mathcal{F}$ such that $\mathcal{L} \subseteq \mathcal{M}$, the complements of the literals of each of the clauses of $\mathcal{L}$ represent backbone literals of $\mathcal{F} \setminus \mathcal{M}$. As a result, *any* algorithm for MCS computation can exploit backbone literals, by adding the negation of each clause that is decided to be included in a MCS of $\mathcal{F}$.

The third technique consists of exploiting clauses satisfied by models of subsets of $\mathcal{F}$. Let $\mathcal{F}$ be split into a subset $\mathcal{S}$ of an MSS of $\mathcal{F}$ and a superset $\mathcal{U}$ of an MCS of $\mathcal{F}$. Any MCS algorithm analyzes the clauses in $\mathcal{U}$. For each such clause c, an MCS algorithm either decides there is an MSS that includes $\mathcal{S} \cup \{c\}$, and so c is added to $\mathcal{S}$, or decides that $\mathcal{S} \cup \{c\}$ is not contained in any MSS, and so c is added to a set $\mathcal{L}$, representing a subset of an MCS of $\mathcal{F}$. Now, suppose a SAT solver is called on $\mathcal{S} \cup \mathcal{T}$, where $\mathcal{T} \subseteq \mathcal{U}$, and suppose the SAT solver returns a satisfying assignment μ. Moreover, let $\mathcal{V}$ denote a subset of $\mathcal{U} \setminus \mathcal{T}$ of clauses satisfied by μ. Then, there *must* exist an MSS of $\mathcal{F}$ that includes $\mathcal{S} \cup \mathcal{T} \cup \mathcal{V}$. Thus, all clauses in $\mathcal{T} \cup \mathcal{V}$ can be added to $\mathcal{S}$. The technique of moving satisfied clauses to set $\mathcal{S}$, and requiring them to continue to be satisfied is used in PICOMCS [Nöhrer *et al.*, 2012]. Below, we show that the technique can be used both in an enhanced version of FASTDIAG [Felfernig *et al.*, 2012] and in the new MCS computation algorithm (see Section 5).

```
Function EFD ($\mathcal{R}, \mathcal{T} \subseteq \mathcal{R}$, hasD)
    Global: $\mathcal{S}, \mathcal{B}, \mathcal{U}_1, \ldots, \mathcal{U}_r$
1   if hasD then
2       $(st, \mu) \leftarrow \texttt{SAT}(\mathcal{R} \cup \mathcal{S} \cup \mathcal{B})$
3       if st then
4           $\texttt{UpdateSatisfiedClauses}(\mu)$
5           return $\emptyset$
6   if $|\mathcal{T}| = 1$ then
7       $\texttt{UpdateBackboneLiterals}(\mathcal{T})$
8       return $\mathcal{T}$
9   $m \leftarrow \lfloor \frac{|\mathcal{T}|}{2} \rfloor$
10  $(\mathcal{T}_1, \mathcal{T}_2) \leftarrow (\mathcal{T}_{1..m}, \mathcal{T}_{m+1..|\mathcal{T}|})$
11  $\mathcal{D}_2 \leftarrow \text{EFD}(\mathcal{R} \setminus \mathcal{T}_1, \mathcal{T}_2 \setminus \mathcal{S}, \mathcal{T}_1 \neq \emptyset)$
12  $\mathcal{D}_1 \leftarrow \text{EFD}(\mathcal{R} \setminus \mathcal{D}_2, \mathcal{T}_1 \setminus \mathcal{S}, \mathcal{D}_2 \neq \emptyset)$
13  return $\mathcal{D}_1 \cup \mathcal{D}_2$              // Clauses of MCS
```
Algorithm 4: Enhanced FASTDIAG (EFD)

Algorithm 4 shows an enhanced version of FASTDIAG, EFD, which integrates the new techniques described in this section, and differs somewhat from BFD (see Algorithm 2). The algorithm uses a number of *global* sets of clauses. Set $\mathcal{B}$ contains the unit clauses representing already computed backbone literals, and set $\mathcal{S}$ denotes the current subset of an MSS of $\mathcal{F}$. Sets $\mathcal{U}_1, \ldots, \mathcal{U}_r$ represent the falsified clauses of each disjoint unsatisfiable core (if being used). Set $\mathcal{B}$ is updated each time a clause is added to the MCS (line 7). Set $\mathcal{S}$ is updated each time the SAT solver returns satisfiable (line 4). Both sets are used when calling the SAT solver (line 2). Moreover, the number of recursive calls is reduced by filtering the clauses in set $\mathcal{S}$ (lines 11 and 12). Finally, as remarked earlier, the handling of disjoint unsatisfiable cores requires no modifications; it suffices to invoke Algorithm 4 on the falsified clauses of each disjoint unsatisfiable core, i.e. $\mathcal{U}_i$, while maintaining the global sets $\mathcal{S}$ and $\mathcal{B}$, and aggregating the clauses from each core that cannot be added to $\mathcal{S}$.

5 New MCS Algorithm

This section develops a new algorithm for computing a single MCS. The algorithm exploits a (simple) observation from [Birnbaum and Lozinskii, 2003] (and used in algorithm AMC1 in [Birnbaum and Lozinskii, 2003]: *the clauses falsified by any complete truth assignment do not have complemented literals*. This remark forms the basis of Algorithm 5, the clause D based (or CLD) algorithm. Consider a truth as-

```
Function CLD ($i$)
    Input: $i$: Target unsatisfiable core
    Global: $\mathcal{S}, \mathcal{B}, \mathcal{U}_1, \ldots, \mathcal{U}_r$
1   $st \leftarrow$ true
2   while $st \wedge (|\mathcal{U}_i| > 1)$ do
3       $D \leftarrow (\vee_{c \in \mathcal{U}_i} c)$
4       $(st, \mu) \leftarrow$ SAT$(\mathcal{S} \cup \mathcal{B} \cup \{D\})$
5       if $st$ then
6           UpdateSatisfiedClauses$(\mu)$
7   UpdateBackboneLiterals$(\mathcal{U}_i)$
8   return $\mathcal{U}_i$                          // Clauses of MCS
```

Algorithm 5: Computing one MCS with clause D (CLD)

signment that splits the clauses in $\mathcal{F}$ into a set of satisfied clauses $\mathcal{S}$ and a set of falsified clauses $\mathcal{U}_i$ (assume $i = 1$, i.e. no disjoint cores considered). Given that the clauses in $\mathcal{U}_i$ do not have complemented literals, we can create their disjunction as a new clause D, and call a SAT solver on $\mathcal{S} \cup \{D\}$. If this formula is satisfiable, that means it is possible to satisfy at least one clause from $\mathcal{U}_i$ while satisfying all the clauses in $\mathcal{S}$. Thus, $\mathcal{S}$ does not yet represent an MSS of $\mathcal{F}$ and $\mathcal{U}_i$ does not yet represent an MCS of $\mathcal{F}$. In this case, the sets $\mathcal{S}$ and $\mathcal{U}_i$ are updated by removing from set $\mathcal{U}_i$ the satisfied clauses and moving them to set $\mathcal{S}$. However, if the formula $\mathcal{S} \cup \{D\}$ is unsatisfiable, then no additional clauses from $\mathcal{U}_i$ can be satisfied given $\mathcal{S}$. Thus, $\mathcal{S}$ represents an MSS of $\mathcal{F}$, and $\mathcal{U}_i$ represents an MCS of $\mathcal{F}$ (observe that disjoint cores are not being considered). Similarly to algorithm EFD, algorithm CLD integrates the techniques described in Section 4. The global sets $\mathcal{S}$ and $\mathcal{B}$ capture, respectively, satisfied clauses and backbones. These are updated the same way as in EFD. Moreover, CLD can also be readily used with disjoint unsatisfiable cores. Instead of considering a single set $\mathcal{U}$ of falsified clauses, Algorithm 5 is called for each set of falsified clauses associated with each unsatisfiable core, i.e. $\mathcal{U}_i$. In the case no disjoint unsatisfiable cores are used, then the CLD can be started by first creating sets $\mathcal{S}$ and $\mathcal{U}$ (see line 1 in Algorithm 1).

If p is the size of the *smallest* MCS of $\mathcal{F}$, then Algorithm 5 requires $m-p+1$ calls to a SAT solver, where $m = |\mathcal{F}|$. This complexity result suggests that Algorithm 5 operates quite differently from both the linear search algorithm and FASTDIAG. This observation is substantiated by the results in Section 6. A possible downside is that the disjunction clause D may result in instances of SAT that are harder than for the other algorithms.

6 Experimental Evaluation

The algorithms proposed in this paper were implemented in the MCSLS tool[2] for MCS computation and enumeration. MCSLS is written in C++ and integrates Minisat 2.2 [Eén and Sörensson, 2003] as the (incremental) SAT solver. The following algorithms have been implemented in MCSLS: BLS (Alg. 1); enhanced linear search, ELS (Alg. 1 with the techniques proposed in Section 4); CLD (Alg. 5 using disjoint unsatisfiable cores); BFD (Alg. 2); and EFD (Alg. 4 using disjoint unsatisfiable cores). All algorithms accept formulas with soft and (optionally) hard clauses. Thus, formulas are either considered *plain* MaxSAT or *partial* MaxSAT. Besides these algorithms, we also developed CAMUS2, an implementation of CAMUS that is able to solve partial MaxSAT instances. The following existing tools were included in the experiments: CAMUS [Liffiton and Sakallah, 2008; 2009], PICOMCS [Nöhrer *et al.*, 2012], NOPTSAT [Rosa *et al.*, 2010], and HYCAM (only for enumeration) [Grégoire *et al.*, 2007]. Most of the tools require formulas without hard clauses (i.e. plain CNF format), and so their evaluation is restricted to these formulas. All experiments were run on a Linux cluster, with a memory limit of 4GB for each process. The time limit was set depending on the experiment, as explained below. The problem instances used include a selection of small-sized industrial instances used in the SAT competitions (assuming all clauses are soft), and a selection of structured instances from MaxSAT evaluations (representing plain and partial MaxSAT instances). Additionally, instances representing the *minimal model* problem [Ben-Eliyahu and Dechter, 1996] were generated from satisfiable industrial instances. The resulting partial MaxSAT formulas are challenging for current MCS algorithms.

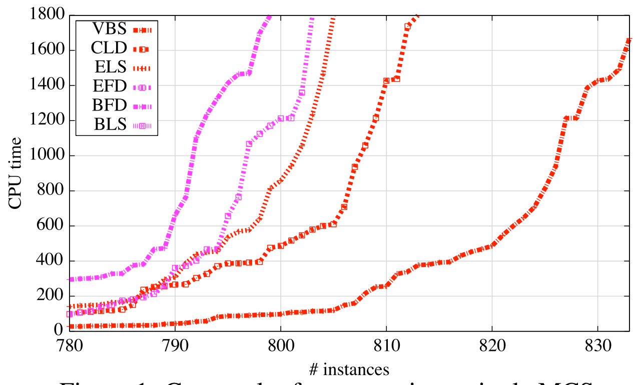

Figure 1: Cactus plot for computing a single MCS

Three experiments were carried out. The first one evaluates the existing and the new MCS algorithms on computing a single MCS. The second experiment evaluates these algorithms on enumerating all MCSes. The set of instances differs from the first experiment, since these need to be simpler. The final experiment compares one of the new algorithms with incomplete MaxSAT solvers, with the objective of evaluating the effectiveness of approximating MaxSAT solutions.

1 MCS. This experiment evaluates the raw performance for computing a single MCS. Preliminary experiments were executed to select instances (a total of 866) for which computing a single MCS was difficult (i.e. time consuming or not possible at all) for at least one of the tools. Table 1 shows the total number of instances for which each algorithm was able to compute one MCS within a time limit of 30 minutes. Existing tools (CAMUS/CAMUS2, PICOMCS and NOPTSAT) do not match the performance of the best performing tools, i.e. ELS, EFD and CLD. CLD is the best performing algorithm, being able to solve 8 more instances than the second best, ELS. The cactus plot in Figure 1 shows the running times for the best performing algorithms. Note that the running times for ELS, EFD and CLD are similar for most instances, with the ex-

[2]Available from http://logos.ucd.ie/wiki/doku.php?id=mcsls.

Set	$\#I$	BLS	BFD	ELS	EFD	CLD	CAMUS2	CAMUS	PICOMCS	NOPTSAT
Plain	269	181	245	**250**	248	248	155	151	193	197
Partial	597	555	553	554	554	**564**	410	N/A	N/A	N/A
Total	866	736	798	804	802	**812**	565	N/A	N/A	N/A

Table 1: Single MCS computation

Set	$\#I$	BLS	BFD	ELS	EFD	CLD	CAMUS2	CAMUS	PICOMCS	NOPTSAT	HYCAM
Plain	184	**89**	88	**89**	88	87	81	80	86	62	81
Partial	591	375	357	**380**	367	372	317	N/A	N/A	N/A	N/A
Total	775	464	445	**469**	455	459	398	N/A	N/A	N/A	N/A

Table 2: All MCS enumeration

ception of less than 20 instances. Both BLS and BFD perform significantly worse than their respective enhanced versions. BLS is orders of magnitude slower than ELS. As a result, it is not shown in Figure 1. BFD is significantly slower than EFD, in most cases by more than a factor of 2. These results indicate that the new techniques are paramount for improving the performance of MCS algorithms. The cactus plot also shows a VBS (Virtual Best Solver) that includes ELS, EFD and CLD. The performance gains of the VBS compared to the other algorithms is quite significant; it confirms that these algorithms complement each other, and motivates investigating portfolio solutions for computing MCSes.

MCS enumeration. This experiment evaluates the performance of the different algorithms for enumerating all MCSes. Preliminary experiments were executed to select a set of 775 problem instances, simpler than for one MCS. Enumeration of all MCSes was possible for around 50% of the instances with at least one of the approaches. Table 2 shows the total number of instances for which each algorithm was able to compute all MCSes within a time limit of 60 minutes. As can be observed, for the plain MaxSAT case, NOPTSAT performs significantly worse than the other approaches. The new algorithms have a slight performance edge over PICOMCS. The cactus plot in Figure 2 shows the running times for the best performing algorithms. Among these, CLD, BLS and ELS are the most efficient. BLS and ELS can solve 5 and 10 more instances than CLD, respectively. However, CLD is substantially faster for most instances. For 440 instances, CLD clearly outperforms the other approaches. This behavior is highlighted in the scatter plot of Figure 3 which compares the running times of CLD against ELS, and shows performance gains that in some cases exceed an order magnitude. The performance gap between CLD and the remaining algorithms is even larger. The cactus plot also shows a VBS of the same set of algorithms as above. Although not as significant as for the single MCS case, the performance gains also justify a portfolio approach.

One of the benchmark suites considered in this experiment is *DC* [Sinz *et al.*, 2003]. This suite consists of 84 instances from an automotive product configuration problem, and has been extensively studied in recent work on MCS/MUS enumeration [Liffiton and Sakallah, 2008; 2009]. For the *DC* benchmark suite, CLD is able to solve 51 instances, whereas the remaining approaches solve 49, and NOPTSAT solves 37. The two instances first solved by CLD correspond to $C210\_FW\_SZ\_128$ (284429 MCSes) and $C210\_FS\_SZ\_103$ (347685 MCSes).

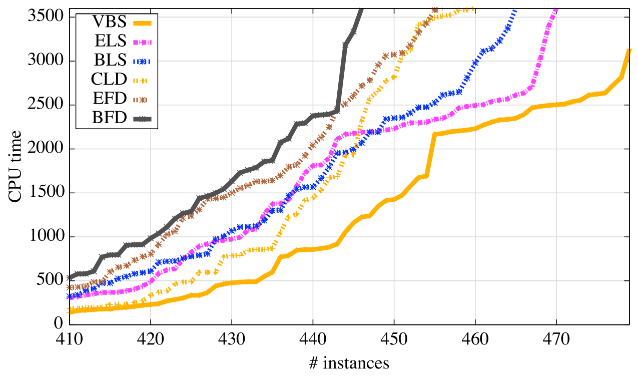

Figure 2: Cactus plot for enumerating all MCSes

MaxSAT Approximation. This experiment evaluates the enumeration of MCSes (within a time bound) to approximate MaxSAT. In some scenarios, it is necessary to quickly get a good quality solution for the MaxSAT problem, rather than waiting for a complete method to find the optimum solution. The size of the smallest computed MCS (within a given time limit) corresponds to a (non-optimal) MaxSAT solution which is compared to the solutions computed by SAT4J [Le Berre and Parrain, 2010] and IROTS [Tompkins and Hoos, 2004], two approximation tools submitted to the MaxSAT Evaluation 2012 [3]. SAT4J is a complete method based on iteratively calling a SAT solver and refining an upper bound. IROTS is an stochastic local search algorithm for MaxSAT.

For this experiment, the CLD algorithm was selected among the best performing algorithms for computing 1 MCS and for MCS enumeration. Nevertheless, any of the other proposed algorithms could be have been considered. In this experiment CLD was executed in enumeration mode and the size of the smallest MCS computed within the time limit corresponds to a (non-optimal) MaxSAT solution. CLD, SAT4J and IROTS were executed with a time limit of 120 seconds. Similar results were obtained for other time limits.

[3]http://maxsat.ia.udl.cat/introduction/.

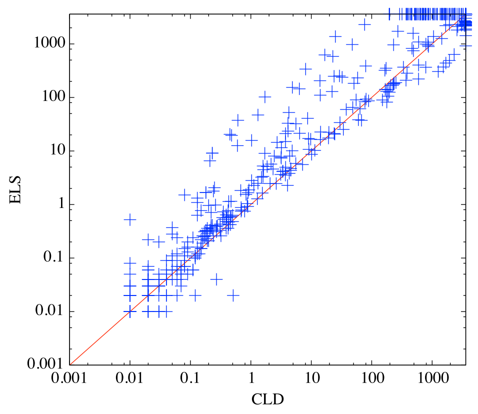

Figure 3: Scatter plot CLD vs ELS

Comparison		SAT4J&IROTS	SAT4J	IROTS	∅
$\#I$		158	115	219	68
CLD vs SAT4J	W	125	52	195	42
	D	13	15	0	0
	L	20	48	0	0
CLD vs IROTS	W	71	91	141	42
	D	9	0	10	0
	L	78	0	68	0
CLD TOs		0	24	24	26

Table 3: MaxSAT approximation

The 866 problem instances used for computing a single MCS were used in this experiment. Of these, SAT4J can find the optimum within the time limit for 306 instances. Since the optimum can be found within a small timeout, these instances are discarded. The results for the remaining 560 instances are summarized in Table 3. The first column shows which two solvers are being compared (CLD vs SAT4J, and CLD vs IROTS). The following columns consider the subsets of instances: (i) for which both SAT4J and IROTS can produce a valid upper bound within the time limit; (ii) for which only SAT4J (or (iii) only IROTS) produces a valid upper bound; and finally (iv) for which neither SAT4J nor IROTS can produce a valid upper bound (column ∅). Note that IROTS can return assignments that do not satisfy all hard clauses. These assignments are discarded as invalid upper bounds. The table shows how many times the MaxSAT approximation computed by CLD is better (W=win), is the same (D=draw) or is worse (L=loss) against the approximation computed by either SAT4J or IROTS. The last row shows the number of instances for which CLD times out without producing an approximation. The results indicate that CLD produces more accurate approximations in 6.09 (2.36) times more cases than SAT4J (IROTS). Thus, we conclude that CLD provides quite competitive approximations. It should be noted that CLD and the remaining MCS enumerators rely on calling a SAT solver, and for some instances it may not find any MCS within the time limit (as shown in Table 3, CLD times out for 74 instances). However, SAT4J and (IROTS) time out on even more instances, 219+68 and 115+68, respectively. Finally, it should be pointed out that a more detailed analysis of the actual quality of the computed MaxSAT approximations is harder to do, since these are instance-dependent.

7 Conclusions

This paper studies the problem of computing minimal correction subsets of unsatisfiable CNF formulas, and makes the following main contributions. First, the paper proposes novel techniques for improving the performance of MCS computation algorithms. These include exploiting disjoint unsatisfiable cores and backbone literals. Second, the paper shows how these techniques can be integrated into existing MCS computation algorithms. Third, and finally, the paper develops a novel algorithm for computing MCSes. The experimental results indicate that the new techniques and algorithms significantly improve over existing algorithms, solving instances that could not be solved before, both for computing a single MCS and for enumerating MCSes. More importantly, the paper shows that the new algorithms for computing MCSes are effective at approximating the solution of (partial) MaxSAT instances. The experimental results indicate that, given a fixed timeout, computing MCSes provides better quality bounds than other incomplete approaches [Tompkins and Hoos, 2004; Le Berre and Parrain, 2010].

Future research work will address further improvements to the new algorithms, including selecting the algorithm to use depending on the size of each target unsatisfiable core. Motivated by the results of the VBS, that solely integrates the algorithms proposed in this paper, additional work includes developing a portfolio approach [Xu *et al.*, 2008; Kadioglu *et al.*, 2011] based on the new MCS algorithms. Another line of work is to exploit recently proposed algorithms for the minimal set over monotone predicate (MSMP) problem, and their specialization for MCSes [Marques-Silva *et al.*, 2013]. Finally, one additional line of work is the integration of the new algorithms for computing MCSes in recent approaches for partial MUS enumeration [Previti and Marques-Silva, 2013; Liffiton and Malik, 2013].

Acknowledgements.

The authors would like to thank the anonymous reviewers for the insightful comments. This work is partially supported by the SFI PI grant BEACON (09/IN.1/I2618), by the FCT grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/123051/2010), and by INESC-ID multiannual funding from the PIDDAC program funds.

References

[Bailey and Stuckey, 2005] James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Practical Aspects of Declarative Languages*, pages 174–186, 2005.

[Ben-Eliyahu and Dechter, 1996] Rachel Ben-Eliyahu and Rina Dechter. On computing minimal models. *Ann. Math. Artif. Intell.*, 18(1):3–27, 1996.

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[Birnbaum and Lozinskii, 2003] Elazar Birnbaum and Eliezer L. Lozinskii. Consistent subsets of inconsistent systems: structure and behaviour. *J. Exp. Theor. Artif. Intell.*, 15(1):25–46, 2003.

[Eén and Sörensson, 2003] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, pages 502–518, 2003.

[Felfernig *et al.*, 2012] Alexander Felfernig, Monika Schubert, and Christoph Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, 26(1):53–62, 2012.

[Grégoire *et al.*, 2007] Éric Grégoire, Bertrand Mazure, and Cédric Piette. Boosting a complete technique to find MSS and MUS thanks to a local search oracle. In *International Joint Conference on Artificial Intelligence*, pages 2300–2305, 2007.

[Jampel *et al.*, 1996] Michael Jampel, Eugene C. Freuder, and Michael J. Maher, editors. *Over-Constrained Systems*. Springer, 1996.

[Junker, 2004] Ulrich Junker. QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In *AAAI Conference on Artificial Intelligence*, pages 167–172, 2004.

[Kadioglu *et al.*, 2011] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *Principles and Practice of Constraint Programming*, pages 454–469, 2011.

[Kilby *et al.*, 2005] Philip Kilby, John K. Slaney, Sylvie Thiébaux, and Toby Walsh. Backbones and backdoors in satisfiability. In *AAAI Conference on Artificial Intelligence*, pages 1368–1373, 2005.

[Le Berre and Parrain, 2010] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010.

[Li and Manyà, 2009] Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*, pages 613–632. IOS Press, 2009.

[Liffiton and Malik, 2013] Mark Liffiton and Ammar Malik. Enumerating infeasibility: Finding multiple MUSes quickly. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2013.

[Liffiton and Sakallah, 2008] Mark Liffiton and Karem Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008.

[Liffiton and Sakallah, 2009] Mark Liffiton and Karem Sakallah. Generalizing core-guided Max-SAT. In *Theory and Applications of Satisfiability Testing*, pages 481–494, 2009.

[Marques-Silva *et al.*, 2013] Joao Marques-Silva, Mikolas Janota, and Anton Belov. Minimal sets over monotone predicates in Boolean formulae. In *Computer Aided Verification*, 2013.

[Meseguer *et al.*, 2003] Pedro Meseguer, Noureddine Bouhmala, Taoufik Bouzoubaa, Morten Irgens, and Martí Sánchez. Current approaches for solving over-constrained problems. *Constraints*, 8(1):9–39, 2003.

[Meseguer *et al.*, 2006] Pedro Meseguer, Francesca Rossi, and Thomas Schiex. *Handbook of Constraint Programming*, chapter Soft Constraints. Elsevier, 2006.

[Morgado *et al.*, 2012] Antonio Morgado, Mark Liffiton, and Joao Marques-Silva. MaxSAT-based MCS enumeration. In *Haifa Verification Conference*, 2012.

[Nöhrer *et al.*, 2012] Alexander Nöhrer, Armin Biere, and Alexander Egyed. Managing SAT inconsistencies with HUMUS. In *VaMoS*, pages 83–91, 2012.

[Previti and Marques-Silva, 2013] Alessandro Previti and Joao Marques-Silva. Partial MUS enumeration. In *AAAI Conference on Artificial Intelligence*, 2013.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[Rosa *et al.*, 2010] Emanuele Di Rosa, Enrico Giunchiglia, and Marco Maratea. Solving satisfiability problems with preferences. *Constraints*, 15(4):485–515, 2010.

[Sinz *et al.*, 2003] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin. Formal methods for the validation of automotive product configuration data. *AI EDAM*, 17(1):75–97, 2003.

[Soh and Inoue, 2010] Takehide Soh and Katsumi Inoue. Identifying necessary reactions in metabolic pathways by minimal model generation. In *European Conference on Artificial Intelligence*, pages 277–282, 2010.

[Tompkins and Hoos, 2004] Dave Tompkins and Holger Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Theory and Applications of Satisfiability Testing*, pages 306–320, 2004.

[van Hoeve, 2011] W.-J. van Hoeve. *Hybrid Optimization: the 10 years of CPAIOR*, chapter Over-Constrained Problems. Springer, 2011.

[Xu *et al.*, 2008] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res. (JAIR)*, 32:565–606, 2008.

[Zhang and Malik, 2003] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Design, Automation and Test in Europe Conference*, pages 10880–10885, 2003.