Literal-Based MCS Extraction

Paper #1149

Abstract

Given an over-constrained system, a Maximal Satisfiable Subset (MSS) denotes a maximal set of constraints that are consistent. A Minimal Correction Subset (MCS, or co-MSS) is the complement of an MSS. MSSes/MCSes find a growing range of practical applications, including optimization, configuration and diagnosis. A number of MCS extraction algorithms have been proposed in recent years, enabling very significant performance gains. This paper builds on earlier work and proposes a finer-grained view of the MCS extraction problem, one that reasons in terms of literals instead of clauses. This view is inspired by the relationship between MCSes and backbones of propositional formulas, which is further investigated, and allows for devising a novel algorithm. Also, the paper develops a number of techniques to approximate (weighted partial) MaxSAT by a selective enumeration of MCSes. Empirical results show substantial improvements over the state-of-the-art in MCS extraction and indicate that MCS-based MaxSAT approximation is very effective in practice.

1 Introduction

For over-constrained systems [Jampel *et al.*, 1996], i.e. sets of inconsistent constraints, a Maximal Satisfiable Subset (MSS) denotes a maximal set of constraints that are consistent. A Minimal Correction Subset (MCS, or co-MSS) is the complement of an MSS, i.e. it is a set-wise minimal set of constraints whose removal renders the system consistent. MSSes/MCSes find an ever increasing number of practical applications, representing an essential tool in the analysis of over-constrained systems [Junker, 2004; Felfernig *et al.*, 2012]. One concrete example is Maximum Satisfiability (MaxSAT), since a maximum set of satisfiable constraints is the largest MSS (and so corresponds to the smallest MCS). MCSes are also related with Minimal Unsatisfiable Subsets (MUSes), by a well-known minimal hitting set duality property that can be traced to the seminal work of Reiter [Reiter, 1987]. Thus, enumeration of MCSes often serves for computing the MUSes of an over-constrained system. Additional application domains include the computation of minimal models [Ben-Eliyahu and Dechter, 1996] and interactive constraint satisfaction and configuration [O'Callaghan *et al.*, 2005], among many others.

A number of MCS extraction and enumeration algorithms have been proposed in recent years [O'Callaghan *et al.*, 2005; Bailey and Stuckey, 2005; Liffiton and Sakallah, 2008; Felfernig *et al.*, 2012; Nöhrer *et al.*, 2012; Marques-Silva *et al.*, 2013; Malitsky *et al.*, 2014; Grégoire *et al.*, 2014; Bacchus *et al.*, 2014], resulting in ever more efficient algorithms, capable of solving increasingly more challenging problem instances. The number and significance of practical applications of MCS extraction and enumeration motivates investigating more efficient algorithms.

This paper builds on recent work in the domain of Boolean satisfiability [Marques-Silva *et al.*, 2013; Grégoire *et al.*, 2014; Bacchus *et al.*, 2014], and proposes a finer-grained view of the MCS extraction problem, by reasoning in terms of literals instead of clauses.This alternative view of the MCS extraction problem is inspired by a fundamental relationship between MCSes and backbones of propositional formulas, which is investigated in this paper, and enables developing a novel MCS extraction algorithm. The new algorithm exhibits a worst-case query complexity that can be characterized in terms of the number of variables in the formula.

Moreover, recent work [Marques-Silva *et al.*, 2013] has shown that restricted enumeration of MCSes is effective at approximating MaxSAT. This paper extends this earlier work, by developing a set of new techniques which enable far better approximation quality than earlier work.

Experimental results, obtained on representative problem instances, demonstrate that the new MCS extraction algorithm consistently outperforms the approaches that currently represent the state-of-the-art. In addition, the results confirm that MaxSAT approximation by restricted MCS enumeration can provide high-quality solutions to the MaxSAT problem within short run times.

The paper is organized as follows. Section 2 introduces the notation and the definitions used throughout the paper. Section 3 provides a brief account of earlier work, highlighting the most effective approaches. The novel MCS extraction algorithm is detailed in Section 4, and the new techniques for approximating MaxSAT are described in Section 5. Section 6 presents the results from the experimental study, both for MCS extraction and for MaxSAT approximation. Section 7 concludes the paper, and identifies research directions.

2 Background

We assume familiarity with propositional logic [Biere *et al.*, 2009] and consider propositional Boolean formulas in Conjunctive Normal Form (CNF). A CNF formula $\mathcal{F}$ is defined over a set of Boolean variables $X = \{x_1, ..., x_n\}$ as a conjunction of clauses $(c_1 \wedge ... \wedge c_m)$. A clause c_i is a disjunction of literals $(l_{i1} \vee ... \vee l_{ik_i})$ and a literal l is either a variable x or its negation $\neg x$. We refer to the set of literals appearing in $\mathcal{F}$ as $L(\mathcal{F})$. Formulas can be alternatively represented as sets of clauses, and clauses as sets of literals. Occasionally, expressions of the form $\vee_{\mathcal{L}}$ will be used to indicate a clause made of the literals in a set $\mathcal{L}$.

A truth assignment, or interpretation, is a mapping $\mu : X \rightarrow \{0, 1\}$. If all the variables in X are assigned a truth value, μ is referred to as a *complete* assignment. Interpretations can be also seen as conjunctions or sets of literals. Truth valuations are lifted to clauses and formulas as follows: μ satisfies a clause c if it contains at least one of its literals, whereas μ falsifies c if it contains the complements of all its literals. Given a formula $\mathcal{F}$, μ satisfies $\mathcal{F}$ (written $\mu \vDash \mathcal{F}$) if it satisfies all its clauses, being μ referred to as a *model* of $\mathcal{F}$.

Given two formulas $\mathcal{F}$ and $\mathcal{G}$, $\mathcal{F}$ entails $\mathcal{G}$ (written $\mathcal{F} \vDash \mathcal{G}$) iff all the models of $\mathcal{F}$ are also models of $\mathcal{G}$. $\mathcal{F}$ and $\mathcal{G}$ are equivalent (written $\mathcal{F} \equiv \mathcal{G}$) iff $\mathcal{F} \vDash \mathcal{G}$ and $\mathcal{G} \vDash \mathcal{F}$.

A formula $\mathcal{F}$ is satisfiable ($\mathcal{F} \nvDash \bot$) if there exists a model for it. Otherwise it is unsatisfiable ($\mathcal{F} \vDash \bot$). SAT is the decision problem of determining the satisfiability of a propositional formula. This problem is NP-*complete* [Cook, 1971].

Several interesting function problems characterize propositional formulas. One of great importance in AI is computing the *backbone* of $\mathcal{F}$ [Kilby *et al.*, 2005], defined as the set of literals appearing in all its models. Formally:

Definition 1 *The backbone $B(\mathcal{F}) \subseteq L(\mathcal{F})$ of $\mathcal{F}$ is the set of all literals l such that $F \vDash l$.*

A number of algorithms have been proposed for computing backbones [Janota *et al.*, 2015]. There are also alternative and more general definitions that capture the notion of equality between literals [Codish *et al.*, 2013].

Given an unsatisfiable formula $\mathcal{F}$, the following subsets represent different notions regarding (set-wise) minimal unsatisfiability and maximal satisfiability [Liffiton and Sakallah, 2008; Marques-Silva *et al.*, 2013]:

Definition 2 *$\mathcal{M} \subseteq \mathcal{F}$ 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 3 *$\mathcal{C} \subseteq \mathcal{F}$ 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 4 *$\mathcal{S} \subseteq \mathcal{F}$ 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.*

Note that an MSS is the complement of an MCS. MUSes and MCSes are closely related by the well-known hitting set duality [Reiter, 1987; Bailey and Stuckey, 2005; Birnbaum and Lozinskii, 2003; Slaney, 2014]: Every MCS (MUS) is an irreducible hitting set of all MUSes (MCSes) of the formula. In the worst case, there can be an exponential number of MUSes and MCSes [Liffiton and Sakallah, 2008; O'Sullivan *et al.*, 2007]. Besides, MCSes are related to the MaxSAT problem, which consists in finding an assignment satisfying as many clauses as possible. The smallest MCS (largest MSS) represents an optimal solution to MaxSAT.

Given the practical significance of handling soft constraints [Meseguer *et al.*, 2003], we consider that formulas may be partitioned into sets of hard and soft clauses, i.e., $\mathcal{F} = \{\mathcal{F}_H, \mathcal{F}_S\}$. Hard clauses must be satisfied, while soft clauses can be relaxed if necessary. Thus, an MCS will be a subset of $\mathcal{F}_S$. Clauses may also have (positive integer) weights. These elements define different versions of the MaxSAT problem. It is said *partial* if $\mathcal{F}_H \neq \emptyset$ (otherwise it is *plain*), and *weighted* if there are weights different from 1. MaxSAT, then, is reformulated as the problem of computing an assignment minimizing the sum of the weights of the soft clauses it falsifies.

3 Related Work

Stemming from the influential work of [Reiter, 1987], computing MCSes has been the subject of a considerable body of research in the fields of model-based diagnosis, constraint satisfaction and Boolean satisfiability. Here, we focus on the most recent and efficient approaches.

Most modern methods rely on iteratively making calls to a complete solver to check the satisfiability of different subformulas. In general, these algorithms handle a partition $\{\mathcal{S}, \mathcal{U}, \mathcal{C}\}$ of $\mathcal{F}$, where $\mathcal{S}$ is a satisfiable subformula, $\mathcal{C}$ contains clauses proved to be inconsistent with $\mathcal{S}$ (i.e. $\forall c \in \mathcal{C}$, $\mathcal{S} \cup \{c\} \vDash \bot$), and $\mathcal{U}$ consists of the remaining clauses of $\mathcal{F}$. Usually, a first call to the solver computes a complete assignment μ satisfying $\mathcal{F}_H$. This serves to initialize the partition with $\mathcal{S}$ (resp. $\mathcal{U}$) containing the clauses satisfied (resp. falsified) by μ, and $\mathcal{C}$ as the empty set. Note $\mathcal{F}_H \subseteq \mathcal{S}$. Eventually, $\mathcal{S}$ ($\mathcal{C}$) will be an MSS (MCS) of $\mathcal{F}$ and $\mathcal{U}$ will become empty.

The simplest algorithm, LINEAR SEARCH [Bailey and Stuckey, 2005], removes one clause c from $\mathcal{U}$ at a time and calls the solver on $\mathcal{S} \cup \{c\}$. If it is satisfiable c is added to $\mathcal{S}$. Otherwise, it is added to $\mathcal{C}$. Other approaches include DICHOTOMIC SEARCH ([O'Callaghan *et al.*, 2005; Li *et al.*, 2013]) and FASTDIAG ([Felfernig *et al.*, 2012]), which mimics the divide-and-conquer QUICKXPLAIN algorithm for MUS extraction [Junker, 2004].

These algorithms can be enhanced with optimizations, such as exploiting models of satisfiable subformulas ([Nöhrer *et al.*, 2012]), which consists in moving from $\mathcal{U}$ to $\mathcal{S}$ all the clauses satisfied by the model obtained after a satisfiable call. Other optimizations are the use of *backbone literals* and *disjoint unsatisfiable cores* ([Marques-Silva *et al.*, 2013]). Some of these techniques are also used by the three best performing algorithms, described below.

Motivated by the work of [Birnbaum and Lozinskii, 2003], the CLAUSED algorithm (CLD) [Marques-Silva *et al.*, 2013] works differently from the previous methods. It checks whether $\mathcal{S}$ can be extended with some clause in $\mathcal{U}$, calling the solver on $\mathcal{S} \cup \{D\}$, where $D = \vee_{L(\mathcal{U})}$ is a clause expressing the disjunction of the clauses in $\mathcal{U}$. If it is satisfiable, CLD updates $\mathcal{S}$ and $\mathcal{U}$ from the computed model and repeats the process until it gets an unsatisfiable call, indicating that the clauses still in $\mathcal{U}$ constitute an MCS.

CMP (*Computational Method for Partitioning*) [Grégoire *et al.*, 2014] is a recent algorithm that exploits the fact that

Table 1: Complexity of MCS extractors: $m = |\mathcal{F}_S|$ and k (p) is the size of the largest (smallest) MCS of $\mathcal{F}$.

Algorithm	**Query complexity**
Linear Search	$\mathcal{O}(m)$
Dichotomic Search	$\mathcal{O}(k \log(m))$
FastDiag	$\mathcal{O}(k + k \log(\frac{m}{k}))$
Clause D	$\mathcal{O}(m - p)$
CMP	$\mathcal{O}(km)$

an MCS is a minimal hitting set of all MUSes. It iteratively searches for *transition clauses*, those that belong to all MUSes of a given subformula, adding them to the MCS. CMP works with an auxiliary set $\mathcal{X} \subseteq \mathcal{U}$ such that $\mathcal{S} \cup \mathcal{X} \vDash \bot$ is known. Initially $\mathcal{X} = \mathcal{U}$ and, repeatedly until $\mathcal{U}$ becomes empty, it selects a clause $c \in \mathcal{X}$ and calls the solver on $\mathcal{S} \cup (\mathcal{X} \setminus \{c\})$. If it is satisfiable, $\mathcal{S}$ and $\mathcal{U}$ are updated exploiting the computed model, c is moved to the MCS $\mathcal{C}$ and $\mathcal{X}$ is reset to $\mathcal{U}$ in order to look for the next transition clause. Otherwise, c is removed from $\mathcal{X}$. Despite its quadratic worst-case complexity, CMP was shown very effective in practice. As other algorithms, CMP is enhanced with optimizations, including some new ones. One of them consists in making a call, in a way similar to CLD, at the beginning of each iteration. This is done by a procedure termed `extendSatPart`, which checks the satisfiability of $\mathcal{S} \cup \{\vee_{L(\mathcal{X})}\}$. If it is satisfiable, $\mathcal{S}$ is extended with the satisfied clauses and $\mathcal{X}$ and $\mathcal{U}$ are filtered accordingly. Otherwise, all the clauses in $\mathcal{X}$ are moved to $\mathcal{C}$ and $\mathcal{X}$ is set to $\mathcal{U}$. So, regardless of the outcome of the solver, the `extendSatPart` procedure removes at least one clause from $\mathcal{U}$. One in every two calls CMP makes to the solver is made inside this procedure, so in the worst case, CMP would require $2 \times m$ calls. This yields the following new result:

Proposition 1 *CMP using the* `extendSatPart` *procedure has a query complexity of $\mathcal{O}(m)$, with $m = |\mathcal{F}_S|$.*

Finally, building on previous work on SAT with preferences [Rosa *et al.*, 2010], Relaxation Search (RS) [Bacchus *et al.*, 2014] is a new general algorithm to handle optional clauses that can be used to extract MCSes. It consists in using a modified SAT solver so that preferences in the variable selection heuristic can be expressed. An MCS is computed by making a single call on an alternative satisfiability problem consisting of (i) the hard clauses together with (ii) for each soft clause c_i, the clauses resulting from $\neg c_i \leftrightarrow b_i$, where b_i is a fresh Boolean variable. The solver is set to first branch on the b_i variables preferring to set them to *false*, i.e., *activating* the associated soft clauses. If the set of hard clauses is satisfiable, it will always return a model, an MCS being the soft clauses c_i such that its associated variable b_i was set to *true*.

Except for RS, the algorithms presented herein make a worst-case number of calls depending on the number of soft clauses in $\mathcal{F}$, as summarized in Table 1. In contrast, RS requires a dedicated solver, which is called once.

4 Literal-Based MCS Extraction

This section first investigates the relationship between MCSes and backbones, which enables a novel viewpoint of the MCS extraction problem. Then, it introduces a new algorithm that exploits these results.

4.1 MCSes, MSSes and Backbones

In [Marques-Silva *et al.*, 2013], the authors introduced a technique termed *backbone literals*. This technique was derived from the observation that, whenever a clause c is proven to belong to the MCS being extracted, i.e. $\mathcal{S} \cup \{c\} \vDash \bot$, the complements of its literals are backbone literals of all the MSSes containing $\mathcal{S}$. Backbone literals can then be added to the working formula as unit clauses, so that each future call to the SAT solver is expected to be easier.

A deeper analysis reveals a closer relationship between the backbone of an MSS and its corresponding MCS. This is illustrated in the following result.

Proposition 2 *Let $\mathcal{S} \subset \mathcal{F}$ be an MSS of $\mathcal{F}$, $\mathcal{C} = \mathcal{F} \setminus \mathcal{S}$ an MCS and $c \in \mathcal{F}$. Then, $c \in \mathcal{C}$ if and only if $B(\mathcal{S})$ falsifies c.*

Proof. Both parts are proved by contradiction.
(*If*). Suppose $c \notin \mathcal{C}$. Then $c \in \mathcal{S}$ and so $\mathcal{S} \equiv \mathcal{S} \cup \{c\}$. As $B(\mathcal{S})$ falsifies c, $\forall l \in c$, $\neg l \in B(\mathcal{S})$, so $\forall l \in c$, $\mathcal{S} \vDash \neg l$, and thus $\mathcal{S} \cup \{c\} \vDash \bot$, contradicting the satisfiability of $\mathcal{S}$.
(*Only if*). Suppose $c \in \mathcal{C}$ is not falsified by $B(\mathcal{S})$. Then there exists a literal $l \in c$ such that $\neg l \notin B(\mathcal{S})$, that is, $\mathcal{S} \nvDash \neg l$. So, there exists an assignment μ such that $\mu \vDash \mathcal{S} \cup \{l\}$, so $\mu \vDash \mathcal{S} \cup \{c\}$ holds, contradicting the maximality (resp. minimality) of $\mathcal{S}$ (resp. $\mathcal{C}$). □

Interestingly, Proposition 2 suggests the possibility to compute an MCS by identifying the subset of the backbone literals of the MSS that falsifies the MCS. In spite of the fact that the MSS is unknown in advance, the following results will be useful for computing these backbone literals and the MSS at the same time.

Proposition 3 *Let $\mathcal{S}' \subseteq \mathcal{S}$, where $\mathcal{S}$ is an MSS of $\mathcal{F}$. Then $B(\mathcal{S}') \subseteq B(\mathcal{S})$.*

Proof. Let $l \in B(\mathcal{S}')$, i.e. $\mathcal{S}' \vDash l$. As $\mathcal{S}' \subseteq \mathcal{S}$, due to monotonicity of logical entailment, $\mathcal{S} \vDash \mathcal{S}' \vDash l$. So $l \in B(\mathcal{S})$. □

Proposition 4 *Let $\mathcal{B} \subseteq B(\mathcal{S}')$, with $\mathcal{S}' \subseteq \mathcal{S}$ and $\mathcal{S}$ an MSS of $\mathcal{F}$. If $c \in \mathcal{F} \setminus \mathcal{S}'$ is falsified by $\mathcal{B}$, then c belongs to the MCS $\mathcal{C} = \mathcal{F} \setminus \mathcal{S}$.*

Proof. Firstly, by Proposition 3, $\mathcal{B} \subseteq B(\mathcal{S})$. Then as $\mathcal{B}$ falsifies c, $B(\mathcal{S})$ falsifies c, so by Proposition 2, $c \in \mathcal{C}$. □

4.2 LBX: A New MCS Extraction Algorithm

The results above allow us to devise a new MCS extraction algorithm that iterates over the literals of $\mathcal{F}$ searching for backbone literals of subsequent satisfiable subformulas. This method, termed Literal-Based eXtractor (LBX), is depicted in Algorithm 1. Besides the sets of clauses $\mathcal{S}$ and $\mathcal{U}$, it works with two sets of literals: $\mathcal{L}$, containing candidates to be tested, and $\mathcal{B}$, with those known to belong to $B(\mathcal{S})$.

Similarly to previous approaches, LBX first initializes the partition $\{\mathcal{S}, \mathcal{U}\}$ from a complete assignment. As the MCS will not contain any clause in $\mathcal{S}$, it suffices to test the literals in $\mathcal{U}$, and so $\mathcal{L}$ is initialized with $L(\mathcal{U})$. Then, LBX enters a loop where a literal l is selected and removed from $\mathcal{L}$ at a time and the solver is called on $\mathcal{S} \cup \mathcal{B} \cup \{l\}$. Note that adding $\mathcal{B}$ does not alter the outcome, but has the potential of easing the SAT problem. If it is satisfiable, $\mathcal{S}$ and $\mathcal{U}$ are updated from the computed model and all the literals that are no longer in $\mathcal{U}$ are filtered out from $\mathcal{L}$. Otherwise, $\neg l$ is added to $\mathcal{B}$ as a backbone literal. LBX terminates when $\mathcal{L}$ becomes empty, returning the MCS $\mathcal{F} \setminus \mathcal{S}$.

LBX can exploit the structure of the formula by selecting the literal $l \in \mathcal{L}$ having most occurrences in $\mathcal{U}$, so that, if the call is satisfiable, more clauses could be expected to be removed from $\mathcal{U}$, reducing $\mathcal{L}$ to a greater extent. Regardless of this choice, LBX is guaranteed to return an MCS and, interestingly, within a number of calls bounded by the number of variables.

Proposition 5 *LBX always returns an MCS of $\mathcal{F}$ and has a query complexity given by $\mathcal{O}(|X|)$.*

Proof. (Sketch)
(*Termination and Complexity*). As, initially, $\mathcal{F}$ is split from a complete assignment, $|\mathcal{L}| \leq |X|$. Then, at each iteration, at least one literal is removed from $\mathcal{L}$. Hence, LBX always terminates making no more than $|X| + 1$ satisfiability tests.
(*Correctness*). We show that, upon termination, $\mathcal{S}$ constitutes an MSS of $\mathcal{F}$. To this aim, it suffices to note that at the beginning of every iteration the following invariant holds: $\mathcal{S}$ is a satisfiable subformula, $\mathcal{U} = \mathcal{F} \setminus \mathcal{S}$ and for all $l \in L(\mathcal{U}) \setminus \mathcal{L}$, $\neg l \in \mathcal{B}$. Eventually, $\mathcal{L} = \emptyset$ and so, for all $l \in L(\mathcal{U})$, $\neg l \in \mathcal{B}$. Therefore, $\mathcal{B} \subseteq B(\mathcal{S})$ falsifies every $c \in \mathcal{U}$. Thus, by Proposition 4, $\mathcal{U}$ is contained in all the MCSes not containing any clause in $\mathcal{S}$. Since $\mathcal{U} = \mathcal{F} \setminus \mathcal{S}$, it is minimal and constitutes an MCS, and so $\mathcal{S}$ an MSS of $\mathcal{F}$. □

LBX admits some further refinements. On the one hand, as $\mathcal{F}$ is unsatisfiable, there are situations where we can detect some backbone literals without making a call to the SAT solver. This happens at any step of the algorithm where $\mathcal{B}$ does not falsify yet any clause in $\mathcal{F}$ and there exists a literal l that is included in all the clauses of $\mathcal{U}$. Then, we can add $\neg l$ to $\mathcal{B}$. This is a direct consequence of the following more general property.

Proposition 6 *Let $\mathcal{S} \subset \mathcal{F}$ be a satisfiable subformula, and $\mathcal{X} \subseteq \mathcal{F} \setminus \mathcal{S}$ such that $\mathcal{S} \cup \mathcal{X} \vDash \bot$. If there exists $l \in L(\mathcal{X})$ s.t. $l \in c$ for all $c \in \mathcal{X}$, then $\neg l \in B(\mathcal{S})$.*

Proof. Suppose $\neg l \notin \mathcal{B}$. Then $\mathcal{S} \cup l \nvDash \bot$ and, since l belongs to all the clauses in $\mathcal{X}$, $\mathcal{S} \cup \mathcal{X} \nvDash \bot$, a contradiction. □

On the other hand, LBX can also incorporate an optimization similar to the `extendSatPart` procedure used in CMP. This consists in issuing, at the beginning of each iteration, a call on $\mathcal{S} \cup \mathcal{B} \cup \{\vee_{\mathcal{L}}\}$. If it is satisfiable, $\mathcal{S}, \mathcal{U}$ and $\mathcal{L}$ are updated exploiting the model as before. Otherwise, it means that the complements of all the literals in $\mathcal{L}$ are backbone literals of $\mathcal{S}$, thus $\mathcal{L}$ becomes empty, leading LBX to termination. It can be easily proved that this call changes neither the correctness, not the query complexity result from Proposition 5, as every call results in at least one literal being removed from $\mathcal{L}$. Furthermore, a similar reasoning that led to Proposition 1, reveals that using this technique introduces an independent

```
Function LBX ($\mathcal{F}$)
    $(\mathcal{S},\mathcal{U}) \leftarrow$ InitialAssignment($\mathcal{F}$)
    $(\mathcal{L},\mathcal{B}) \leftarrow (L(\mathcal{U}), \emptyset)$
    while $\mathcal{L} \neq \emptyset$ do
        $l \leftarrow$ RemoveLiteral($\mathcal{L}$)
        $(st, \mu) =$ SAT($\mathcal{S} \cup \mathcal{B} \cup \{l\}$)
        if st then
            $(\mathcal{S},\mathcal{U}) \leftarrow$ UpdateSatClauses($\mu, \mathcal{S}, \mathcal{U}$)
            $\mathcal{L} \leftarrow \mathcal{L} \cap L(\mathcal{U})$
        else
            $\mathcal{B} \leftarrow \mathcal{B} \cup \{\neg l\}$
    return $\mathcal{F} \setminus \mathcal{S}$                    // MCS of $\mathcal{F}$
```
Algorithm 1: Literal-Based eXtractor (LBX)

new upper bound on the number of calls LBX would need to perform, which allows us to conclude that it has a query complexity of $\mathcal{O}(\min\{m, |X|\})$, with $m = |\mathcal{F}_S|$.

Finally, it is worth mentioning that LBX presents an important advantage from a technical point of view. Literals can be tested using assumptions, so an incremental SAT solver can be interfaced without the need of creating fresh Boolean variables as clause selectors. This results in no memory overload (as there are no additional variables) and in a hopefully more effective use of propagation and clause learning.

5 MaxSAT Approximation

Enumerating MCSes represents a promising way of approximating MaxSAT [Marques-Silva *et al.*, 2013]. Arguably, an MCS constitutes a form of local optimal solution, as it is not possible to satisfy any of its clauses without falsifying at least one contained in the respective MSS. So, the average quality of the solutions associated to MCSes can be expected to be higher than that of just random assignments satisfying $\mathcal{F}_H$. MCS enumeration is done by iteratively running an MCS extraction algorithm over $\mathcal{F}$, which is augmented with the hard clause $\vee_{L(\mathcal{C})}$ after computing the MCS $\mathcal{C}$. This way, $\mathcal{C}$ is *blocked*, preventing the algorithm from finding it again. The process is repeated until $\mathcal{F}_H \vDash \bot$, meaning that all the MCSes have been enumerated, and the best solution found, corresponding to a cost-min MCS, represents a proven optimum.

This section presents three techniques for improving MCS-based MaxSAT approximation. These introduce a heuristic component for making a selective enumeration of MCSes and avoid the computation of some MCSes that cannot improve the current best solution. The first one can only be used with LBX. Here, $w(c)$ denotes the weight of the clause c, $w(\mathcal{X})$ the sum of the weights of the clauses in $\mathcal{X}$ and UB the cost of the best solution found so far.

Literal Selection Heuristic

LBX makes a series of non-deterministic choices regarding the literal in $\mathcal{L}$ it selects at each iteration. The first technique defines a selection policy aimed at biasing the process towards MCSes with small costs. For this purpose, it selects the literal resulting from the expression $\arg\max\{\sum_{(l \in c; c \in \mathcal{U})} w(c); l \in \mathcal{L}\}$, i.e. for each literal it computes an aggregated weight as the sum of the weights of the clauses in $\mathcal{U}$ it appears in, selecting that with the maximum value. Ties are broken arbitrarily. Note that in the case of unweighted formulas, the selected literal would be the one with most occurrences in $\mathcal{U}$. This technique contributes moderately to guiding the search, as LBX exploits models adding the satisfied clauses to the MSS under construction. However, deactivating this feature would deteriorate its performance, an so its ability to enumerate a large number of MCSes in a short time. The next technique introduces an additional component to achieve a proper balance to this respect.

Stratified MCS Extraction
The second technique exploits the fact that MCSes can be computed in a recursive manner by initially splitting the set of soft clauses as $\mathcal{F}_S = \{\mathcal{F}_{S_1}, ..., \mathcal{F}_{S_k}\}$. Then, $\mathcal{C} = \cup_{(i=1..k)}\mathcal{C}_i$ is an MCS of $\mathcal{F}$, where $\mathcal{C}_1$ is an MCS of $\mathcal{F}_H \cup \mathcal{F}_{S_1}$ and, for $i = 2..k$, $\mathcal{C}_i$ is an MCS of $\mathcal{F}_H \cup \cup_{(j=1..i-1)}(\mathcal{F}_{S_j} \setminus \mathcal{C}_j) \cup \mathcal{F}_{S_i}$. This result can be exploited to establish preferences regarding the kind of MCSes to be computed first. Motivated by the work of [Ansótegui *et al.*, 2012], we define the partition so that clauses with the same weight are included in the same subset, and subsets with largest weights are processed first in the MCS extraction process. This way, the MCS extractor is forced to try to first satisfy clauses with larger weights, as no clause in a subset is considered until all the preceding subsets have been completely processed.

Search Space Pruning
The third technique stems from the observation that we can avoid the enumeration of some MCSes without losing the possibility of finding an optimal solution to MaxSAT. This is a direct consequence of the following result:

Proposition 7 *Let $\mathcal{C}$ be an MCS of $\mathcal{F}$ and $\mathcal{C}' \subseteq \mathcal{C}$ a subset s.t. $w(\mathcal{C}') \geq UB$. Adding to $\mathcal{F}_H$ the clause $\vee_{L(\mathcal{C}')}$ (blocking $\mathcal{C}'$) does not prevent an MCS enumerator from finding any MCS whose cost is less than UB.*

Proof. Adding $\vee_{L(\mathcal{C}')}$ to $\mathcal{F}_H$ guarantees that from then on, all the computed MSSes will contain at least one clause from $\mathcal{C}'$. Also, every MCS improving UB cannot contain $\mathcal{C}'$. □

This can be exploited in two ways: First, it allows us to stop the computation of an MCS whenever its cost exceeds UB. Second, we can always block a subset, avoiding the computation of all the MCSes containing it, which cannot lead to improving UB. After computing an MCS $\mathcal{C}$, or part of it exceeding UB, we compute its cost-minimum subset $\mathcal{C}' \subseteq \mathcal{C}$ such that $w(\mathcal{C}') \geq UB$ and add the hard clause $\vee_{L(\mathcal{C}')}$ to $\mathcal{F}$.

6 Experimental Study

This experimental study evaluates the proposed Literal-Based eXtractor algorithm (LBX) with respect to the state-of-the-art. It consists of two parts; the first one devoted to computing one MCS and the second one to MaxSAT approximation. LBX includes the refinements described at the end of Section 4.2. All the experiments were run on a Linux cluster (2 GHz, 64-bits), setting a memory limit to 4 GB. Our prototype was implemented in C++ interfacing the SAT solver Minisat 2.2 [Eén and Sörensson, 2003].

6.1 MCS Extraction

Regarding MCS extraction, we compare LBX with CLD, which was the best-performing algorithm among the methods presented in [Marques-Silva *et al.*, 2013], RS [Bacchus *et al.*, 2014] and CMP (with all its optimizations) [Grégoire *et al.*, 2014]. In all cases, the original binaries provided by the authors were run according to their instructions. In these experiments we set a time limit of 1800 seconds.

Table 2: MCS extraction. Instances solved by 30 min.

	ijcai13-bench	*mus-bench*	*hard-bench*
# Inst.	866	295	179
CLD	812	271	70
RS	826	258	75
CMP	844	292	103
LBX	**846**	**293**	**138**
VBS	848	294	145

We considered three sets of unsatisfiable instances: The first one, referred to as *ijcai13-bench*, contains 866 instances[1] taken from [Marques-Silva *et al.*, 2013], both plain and partial formulas. The second one, proposed in [Grégoire *et al.*, 2014], includes 295 plain instances from the 2011 MUS competition. Finally, as most of these instances were solved swiftly by all the algorithms (in many cases, a fraction of a second), we propose a new set of hard instances. It contains all the instances from the 2014 MaxSAT evaluation[2] as well as from other sources such that either CMP or LBX took more than 90s to solve, filtering out those where none of these algorithms was able to complete a single call to the SAT solver. The resulting set, *hard-bench*, contains 179 instances.

Table 2 reports, for each algorithm and benchmark set, the number of instances solved by the time limit. Figures 1 and 2 show, respectively, the running times of the methods on the combination of *ijcai13-bench* and *mus-bench* and on the *hard-bench* sets of instances. VBS (*Virtual Best Solver*) emulates a portfolio including all the algorithms.

In the first two sets, LBX is the best method, closely followed by CMP, which is clearly better than RS and CLD. LBX is able to solve 3 more instances than CMP, and many more than both RS and CLD. Also, Fig. 1 shows clear differences in the running times in favor of LBX over CMP. It is important to remark that for these instances there is not much room for improvement: most of the unsolved *ijcai13* instances are extremely hard for SAT solving, while in *mus-bench* there is only one instance that no algorithm can solve.

The results from the third benchmark show more important differences. As we can observe in Figure 2, LBX performs significantly better than any other algorithm. Remarkably, LBX is able to solve 35, 63 and 68 more instances than CMP, RS and CLD respectively.

Finally, VBS, to which LBX contributes substantially, confirms the conclusion of previous works that a combined approach can lead to improvements. It results in a reduction of the running times and in the ability to solve some more instances. This can be expected, as the algorithms that form VBS are essentially different from each other, so an instance that may be hard for one method, could be easy for another one, and vice versa.

[1][Bacchus *et al.*, 2014] and [Grégoire *et al.*, 2014] considered 1343 instances from [Marques-Silva *et al.*, 2013]. We only consider the 866 instances intended for MCS extraction, as the remaining ones were proposed for MCS enumeration, since these are easy.

[2]http://www.maxsat.udl.cat/14/index.html

Table 3: MaxSAT approximation. Comparison with state-of-the-art methods.

		CLD			WPM			Dist			BC			# WINS			
Set	**# Inst.**	**L**	**D**	**W**	**L**	**D**	**W**	**L**	**D**	**W**	**L**	**D**	**W**	**CLD**	**WPM**	**Dist**	**LBX**
ms-ind	14	2	1	**11**	4	0	**10**	2	0	**12**	3	0	**11**	2	3	0	**9**
pms-ind	55	19	9	**27**	6	4	**45**	13	2	**40**	19	5	**31**	19	8	12	**28**
wpms-ind	47	12	3	**32**	19	3	**25**	10	4	**33**	8	4	**35**	12	**20**	8	15
pms-crafted	36	6	13	**17**	3	8	**25**	**19**	0	17	2	0	**34**	8	6	**19**	15
wpms-crafted	71	8	2	**61**	20	7	**44**	20	3	**48**	2	2	**67**	2	22	22	**37**
weighted	118	20	5	**93**	39	10	**69**	30	7	**81**	10	6	**102**	14	42	30	**52**
unweighted	105	27	23	**55**	13	12	**80**	34	2	**69**	24	5	**76**	29	17	31	**52**
Total	223	47	28	**148**	52	22	**149**	64	9	**150**	34	11	**178**	43	59	61	**104**

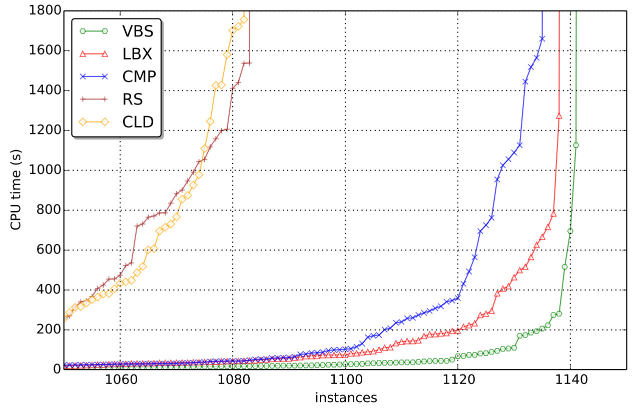

Figure 1: Running times from *ijcai13* and *mus* instances. Note that the plot is zoomed into the range [1050, 1150]

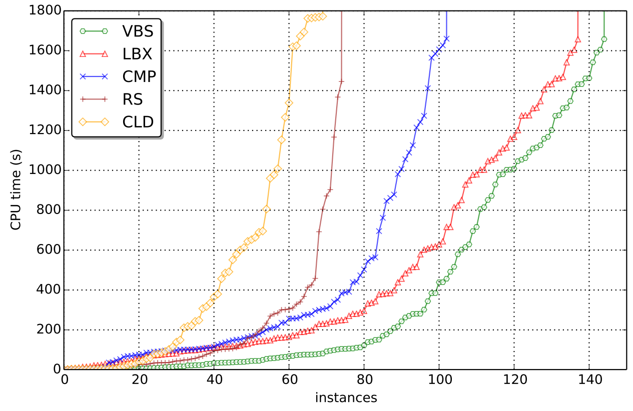

Figure 2: Running times from *hard* instances.

6.2 MaxSAT Approximation

The second series of experiments is focused on MaxSAT approximation. We compare LBX, using the techniques presented in Section 5, with three state-of-the-art approximation methods: CLD (in approximation mode), which was shown to outperform IROTS [Tompkins and Hoos, 2004] and SAT4J [Berre and Parrain, 2010], Dist [Cai *et al.*, 2014], which ranked first in the (weighted) partial crafted incomplete tracks in the 2014 MaxSAT evaluation and *wpm2014.in* (WPM), which ranked first in the (weighted) partial industrial tracks.

The comparison takes place on instances that may be very hard for exact solvers. So, for each crafted and industrial sets of benchmarks of the 2014 MaxSAT evaluation, we considered the instances such that the winner of the respective complete track took more than 500 seconds (or aborted in 30 min.). These exact solvers are Eva [Narodytska and Bacchus, 2014], ISAC+ [Ansótegui *et al.*, 2014] and Open-WBO [Martins *et al.*, 2014a; Martins *et al.*, 2014b]. In all the experiments, the approximation algorithms were given 60 seconds.

Table 3 reports, for each benchmark set, a comparison of LBX with the three mentioned algorithms. L, D and W show, respectively, the number of instances where LBX returned a worse, equal or better solution than the method it is compared to by the time limit. We include BC, that comprises the best upper bound returned by the winner of the respective complete track by a time limit of 30 min. The last columns (# WINS) show the number of instances each method reached the best solution found by all the algorithms (excluding BC).

The results reveal that LBX is very effective at approximating MaxSAT. It reaches better solutions for more instances than any other method in most sets, in many cases by far. The only exception is the pms-crafted set, where LBX shows a slightly worse behavior than Dist. Moreover, according to # WINS, LBX ranks first in three sets and second in the remaining two. Also, regardless of the instances being weighted or unweighted, LBX outperforms any other method, so it is quite stable. Overall, the ratio W/L ranges between 2.3 (for Dist) to 5.2 (for BC), which indicates that, compared to any other method, LBX returns better solutions many more times than worse solutions. Finally, it sums 2.42, 1.76 and 1.7 times more # WINS than CLD, WPM and Dist respectively.

7 Conclusions

MCS extraction is a fundamental step in the analysis of overconstrained systems, finding many applications in AI. This paper develops a novel algorithm (LBX) that exploits the tight relationship between MCSes and backbones of propositional formulas. Moreover, the paper develops novel insights on how to approximate MaxSAT with restricted MCS enumeration. The experimental results demonstrate that the novel MCS extraction algorithm outperforms what are currently the best performing approaches. In addition, they confirm that restricted MCS enumeration represents an effective approach for MaxSAT approximation, yielding for most (hard) problem instances, high-quality solutions within a small run time.

One natural research direction is to deploy the novel LBX algorithm in the growing range of applications of MCSes. Also, new literal-based algorithms could be devised by building on different strategies (e.g. dichotomic search). Finally, the results motivate further research on additional techniques for MCS-based MaxSAT approximation.

References

[Ansótegui *et al.*, 2012] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy. Improving SAT-based weighted MaxSAT solvers. In *CP*, pages 86–101, 2012.

[Ansótegui *et al.*, 2014] C. Ansótegui, Y. Malitsky, and M. Sellmann. MaxSAT by improved instance-specific algorithm configuration. In *AAAI*, pages 2594–2600, 2014.

[Bacchus *et al.*, 2014] F. Bacchus, J. Davies, M. Tsimpoukelli, and G. Katsirelos. Relaxation search: A simple way of managing optional clauses. In *AAAI*, pages 835–841, 2014.

[Bailey and Stuckey, 2005] J. Bailey and P. J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *PADL*, pages 174–186, 2005.

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

[Berre and Parrain, 2010] D. L. Berre and A. Parrain. The Sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.

[Biere *et al.*, 2009] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

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

[Cai *et al.*, 2014] S. Cai, C. Luo, J. Thornton, and K. Su. Tailoring local search for partial MaxSAT. In *AAAI*, pages 2623–2629, 2014.

[Codish *et al.*, 2013] M. Codish, Y. Fekete, and A. Metodi. Backbones for equality. In *HVC 2013*, pages 1–14, 2013.

[Cook, 1971] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[Eén and Sörensson, 2003] N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT*, pages 502–518, 2003.

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

[Grégoire *et al.*, 2014] É. Grégoire, J. Lagniez, and B. Mazure. An experimentally efficient method for (MSS, coMSS) partitioning. In *AAAI*, pages 2666–2673, 2014.

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

[Janota *et al.*, 2015] M. Janota, I. Lynce, and J. Marques-Silva. Algorithms for computing backbones of propositional formulae. *AI Commun.*, 28(2):161–177, 2015.

[Junker, 2004] U. Junker. QuickXplain: Preferred explanations and relaxations for over-constrained problems. In *AAAI*, pages 167–172, 2004.

[Kilby *et al.*, 2005] P. Kilby, J. K. Slaney, S. Thiébaux, and T. Walsh. Backbones and backdoors in satisfiability. In *AAAI*, pages 1368–1373, 2005.

[Li *et al.*, 2013] H. Li, H. Shen, Z. Li, and J. Guo. Reducing consistency checks in generating corrective explanations for interactive constraint satisfaction. *Knowl.-Based Syst.*, 43:103–111, 2013.

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

[Malitsky *et al.*, 2014] Y. Malitsky, B. O'Sullivan, A. Previti, and J. Marques-Silva. A portfolio approach to enumerating minimal correction subsets for satisfiability problems. In *CPAIOR*, pages 368–376, 2014.

[Marques-Silva *et al.*, 2013] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. On computing minimal correction subsets. In *IJCAI*, 2013.

[Martins *et al.*, 2014a] R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce. Incremental cardinality constraints for MaxSAT. In *CP*, pages 531–548, 2014.

[Martins *et al.*, 2014b] R. Martins, V. M. Manquinho, and I. Lynce. Open-WBO: A modular MaxSAT solver. In *SAT*, pages 438–445, 2014.

[Meseguer *et al.*, 2003] P. Meseguer, N. Bouhmala, T. Bouzoubaa, M. Irgens, and M. Sánchez. Current approaches for solving over-constrained problems. *Constraints*, 8(1):9–39, 2003.

[Narodytska and Bacchus, 2014] N. Narodytska and F. Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *AAAI*, pages 2717–2723, 2014.

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

[O'Callaghan *et al.*, 2005] B. O'Callaghan, B. O'Sullivan, and E. C. Freuder. Generating corrective explanations for interactive constraint satisfaction. In *CP*, pages 445–459, 2005.

[O'Sullivan *et al.*, 2007] B. O'Sullivan, A. Papadopoulos, B. Faltings, and P. Pu. Representative explanations for over-constrained problems. In *AAAI*, pages 323–328. AAAI Press, 2007.

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

[Rosa *et al.*, 2010] E. D. Rosa, E. Giunchiglia, and M. Maratea. Solving satisfiability problems with preferences. *Constraints*, 15(4):485–515, 2010.

[Slaney, 2014] J. Slaney. Set-theoretic duality: A fundamental feature of combinatorial optimisation. In *ECAI*, pages 843–848, 2014.

[Tompkins and Hoos, 2004] D. A. D. Tompkins and H. H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In *SAT*, 2004.