Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Partial MUS Enumeration

Alessandro Previti
Complex and Adaptive Systems Laboratory
University College Dublin, Ireland
Email: alessandro.previti@ucdconnect.ie

Joao Marques-Silva
CASL, University College Dublin, Ireland
IST/INESC-ID, Portugal
Email: jpms@ucd.ie

Abstract

Minimal explanations of infeasibility find a wide range of uses. In the Boolean domain, these are referred to as Minimal Unsatisfiable Subsets (MUSes). In some settings, one needs to enumerate MUSes of a Boolean formula. Most often the goal is to enumerate all MUSes. In cases where this is computationally infeasible, an alternative is to enumerate some MUSes. This paper develops a novel approach for partial enumeration of MUSes, that complements existing alternatives. If the enumeration of all MUSes is viable, then existing alternatives represent the best option. However, for formulas where the enumeration of all MUSes is unrealistic, our approach provides a solution for enumerating some MUSes within a given time bound. The experimental results focus on formulas for which existing solutions are unable to enumerate MUSes, and shows that the new approach can in most cases enumerate a non-negligible number of MUSes within a given time bound.

Introduction

Minimal Unsatisfiable Subformulas (MUSes) of Boolean formulas in conjunctive normal form (CNF) are used in different settings. Examples of applications include inconsistency measurement (Hunter and Konieczny 2006; 2010; Xiao and Ma 2012), type error debugging (de la Banda, Stuckey, and Wazny 2003; Bailey and Stuckey 2005), debugging of relational specifications (Torlak, Chang, and Jackson 2008; Torlak, Vaziri, and Dolby 2010), analysis of over-constrained temporal problems (Liffiton et al. 2005), axiom pinpointing in description logics (Schlobach et al. 2007), software and hardware model checking (Andraus, Liffiton, and Sakallah 2008), among many others. Enumeration of MUSes is also tightly related with model-based diagnosis (Reiter 1987). In most applications, it is important to provide different explanations of infeasibility. For Boolean formulas, this means that one needs to enumerate some (or preferably all) MUSes. A number of algorithms for enumeration of MUSes have been proposed in recent years (de la Banda, Stuckey, and Wazny 2003; Bailey and Stuckey 2005; Liffiton and Sakallah 2005; 2008; Grégoire, Mazure, and Piette 2007; Stern et al. 2012), that build on earlier work (Reiter 1987; Han and Lee 1999).

Despite the important improvements observed in MUS enumeration, a number of drawbacks can be identified. First, earlier work can require a very large number of calls to a SAT solver (de la Banda, Stuckey, and Wazny 2003; Bailey and Stuckey 2005). This issue is addressed in more recent work (Liffiton and Sakallah 2005; 2008). However, a drawback of (Liffiton and Sakallah 2008) is the enumeration of Minimal Correction Subformulas (MCSes) before hitting set dualization is applied for enumerating MUSes. The worst-case number of MCSes (and of MUSes) is exponential on the size of the formula. Thus, when the worst-case number of MCSes is exercised, resources are exceeded before enumeration of MUSes takes place. In (Liffiton and Sakallah 2008) a solution to this problem is to use Partial Correction Subsets (PCSes), subsets of MCSes such that some MUSes are preserved. PCSes are smaller in number than MCSes, but enumeration of MUSes using PCSes may not be complete. However, as will be shown in this paper, even the enumeration of PCSes as proposed in (Liffiton and Sakallah 2008) may be unrealistic for some formulas. To our best knowledge, for formulas having a very large number of MCSes or PCSes, there exists no solution that can enumerate a non-negligible number of MUSes. In this paper, the problem of enumerating a non-trivial subset (i.e. with more than 1 element) of the MUSes of an unsatisfiable formula is referred to as *partial MUS enumeration*. Despite focusing on the enumeration of MUSes (and MCSes) of CNF formulas, the work in this paper can be related with model-based diagnosis (e.g. (Reiter 1987)), where MUSes correspond to *minimal conflict sets* and MCSes correspond to *minimal diagnoses*.

This paper proposes a novel algorithm for MUS enumeration, that represents an effective approach for partial MUS enumeration. The algorithm works by generating candidate sets of clauses with a key property: each set either contains an MUS or represents the complement of an MCS. As a result, the partial enumeration of MUSes does not necessarily require the complete enumeration of MCSes or PCSes. As the experimental results show, for formulas for which existing solutions fail to enumerate MUSes, our approach is able to partially enumerate MUSes. (As indicated by the experimental results, the number of MUSes that can be enu-

merated depends on the target formula.) It should be noted that the new approach does not replace the state of the art in MUS enumeration (Liffiton and Sakallah 2008). For formulas where enumeration is viable, existing approaches are the preferred option. However, for formulas where enumeration or partial enumeration is impossible with existing solutions, the approach proposed in this paper is able to compute a number of MUSes that is bounded by available (time and memory) resources.

Preliminaries

This section introduces the basic definitions used throughout the paper. Although the new algorithm for MUS enumeration can applied to any constraint programming problem, the paper addresses the more specific problem of Boolean formula satisfiability. In this paper Boolean formulas are represented in Conjunctive Normal Form (CNF), where a formula is a conjunction of clauses, and each clause is disjunction of literals. A literal is either a Boolean variable or its complement. Formulas can also be viewed as sets of clauses, and clauses as sets of literals. Boolean variables can be assigned a value in set $\{0, 1\}$. An interpretation is a mapping from the set of variables V to $\{0, 1\}$, $m : X \rightarrow \{0, 1\}$. A formula is satisfiable if it admits a model, that is an interpretation that satisfies it. Otherwise, $\mathcal{F}$ is said to be unsatisfiable. Given a CNF formula $\mathcal{F}$, the Boolean Satisfiability (SAT) problem consists in deciding whether $\mathcal{F}$ is satisfiable. This paper studies unsatisfiable formulas $\mathcal{F}$, and the definitions below will be used throughout (e.g. see (Büning and Kullmann 2009; Liffiton and Sakallah 2008)). Minimal Unsatisfiable Subsets (MUSes) are defined as follows:

Definition 1. *A subset $\mathcal{M} \subseteq \mathcal{F}$ is an MUS if $\mathcal{M}$ is unsatisfiable and $\forall \mathcal{C} \subset \mathcal{M}$ with $\mathcal{C} \neq \emptyset$, $\mathcal{M} \setminus \mathcal{C}$ is satisfiable.*

Every unsatisfiable formula contains at least one MUS. A related concept is the one of Minimal Correction Subset (MCS):

Definition 2. *A subset $\mathcal{N}$ of $\mathcal{F}$ is an MCS if $\mathcal{F} \setminus \mathcal{N}$ is satisfiable and $\forall \mathcal{G} \subseteq \mathcal{N} \wedge \mathcal{G} \neq \emptyset$, $(\mathcal{F} \setminus \mathcal{N}) \cup \mathcal{G}$ is unsatisfiable.*

An MCS can be defined as the complement of a Maximal Satisfiable Subset (MSS):

Definition 3. *A satisfiable subset $\mathcal{S} \subseteq \mathcal{F}$ is an MSS if $\forall \mathcal{C} \subseteq \mathcal{F} \setminus \mathcal{S} \wedge \mathcal{C} \neq \emptyset$, $\mathcal{S} \cup \mathcal{C}$ is unsatisfiable.*

For partial MUS enumeration (Liffiton and Sakallah 2008) use Partial Correction Subsets (PCSes):

Definition 4. *A subset $P \subseteq \mathcal{F}$ is a PCS if there exists some MCS $\mathcal{N}$ such that $P \subseteq \mathcal{N}$.*

Thus, a PCS is part of a diagnosis. Note that the complement of a PCS that is a proper subset of an MCS is unsatisfiable because it is is a superset of an MSS. Moreover, MUSes and MCSes are related by the concept of *minimal hitting set*.

Definition 5. *Given a collection Γ of sets from a universe U, a hitting set H for Γ is a set such that $\forall S \in \Gamma, H \cap S \neq \emptyset$.*

A hitting set H is *minimal* if none of its subset is a hitting set. The relationship between MUSes and MCSes is well-known (e.g. see (Reiter 1987; Birnbaum and Lozinskii 2003; Liffiton and Sakallah 2008)):

Proposition 1. *Let MUSes($\mathcal{F}$) and MCSes($\mathcal{F}$) be the set of all MUSes and MCSes of $\mathcal{F}$ respectively. Then the following hold:*

1. *A subset $\mathcal{M}$ of $\mathcal{F}$ is an MUS if and only if $\mathcal{M}$ is a minimal hitting set of MCSes($\mathcal{F}$).*
2. *A subset $\mathcal{N}$ of $\mathcal{F}$ is an MCS if and only if $\mathcal{N}$ is a minimal hitting set of MUSes($\mathcal{F}$).*

Enumeration of MUSes

In recent years, a number of different algorithms have been proposed for MUS enumeration (Han and Lee 1999; de la Banda, Stuckey, and Wazny 2003; Bailey and Stuckey 2005; Liffiton and Sakallah 2008; Grégoire, Mazure, and Piette 2007). These approaches can be divided into two main categories:

- MUSes obtained by direct computation;
- MUSes obtained by hitting set dualization.

The direct computation of MUSes (Han and Lee 1999; de la Banda, Stuckey, and Wazny 2003) is based on explicit enumeration of the subsets of a formula. The enumeration of subsets is organized in a tree (the CS-tree) to avoid repetition of subsets. An order of the clauses is used to specify how the subsets are considered. Each node of the CS-tree is labeled with a subset $\mathcal{S}$ and each of its children is labeled with a subset $\mathcal{S}' \subset \mathcal{S}$. The nodes of the tree are visited in a depth first manner and at each node a satisfiability test is performed on that subset. A subset $\mathcal{S}$ of a node n is marked as an MUS if $\mathcal{S}$ is unsatisfiable and all the children of n are labeled with satisfiable subsets. (de la Banda, Stuckey, and Wazny 2003) develop a number of improvements over the original approach (Han and Lee 1999), but the explicit enumeration of subsets coupled with the iterative SAT tests is often a performance bottleneck (Bailey and Stuckey 2005). A different alternative can be obtained from Reiter's algorithm for computing diagnoses (Reiter 1987). Although the original purpose was to enumerate diagnoses, the algorithm can be easily adapted to enumerate all MUSes. The algorithm also makes use of a tree (the HS-tree), whose goal is to enumerate all conflict sets without repetition. Each time a new node is expanded, a new conflict set is computed by means of a theorem prover or a new diagnosis is extracted from the tree. The algorithm can be modified to enumerate MUSes, if in place of generic conflict sets, the theorem prover returns MUSes.

Examples of algorithms based on exploiting the hitting set duality between MUSes and MCSes include DAA (Bailey and Stuckey 2005) and CAMUS (Liffiton and Sakallah 2005; 2008). CAMUS starts by computing *all* MCSes using a dedicated procedure based on Maximum Satisfiability (MAX-SAT). MCSes are computed by increasing size using enumeration of MAX-SAT solutions. For the actual implementation, every clause c_i is augmented with a selector variable y_i, that is used to enable or disable the clause. Augmented clauses are of the form $c'_i = (x_1 \vee x_2 \vee \ldots \vee x_n \vee \neg y_i)$, where $c_i = (x_1 \vee x_2 \vee \ldots \vee x_n)$ is the original clause. Moreover, an AtMost($\{\neg y_1, \neg y_2, \ldots, \neg y_n)\}, k$) constraint

is added to the original formula $\mathcal{F}$, resulting in a new formula $\mathcal{F}'$. The aim of this constraint is to specify a bound on the number of clause selector variables that can be assigned value 0. Let m be a model for the formula $\mathcal{F}'$. Then the procedure guarantees that m contains the minimal number of y_i variables assigned to 0 and consequently the minimal number of disabled clauses. Each new MCS is added as a clause to the formula, to avoid future repetitions. This clause has the form $B = (y_1 \vee y_2 \vee \ldots \vee y_m)$, meaning that at least one of the clauses in the last MCS must be enabled in future solutions. The second phase of CAMUS starts after the entire set of MCSes is computed. In this second phase the MUSes are extracted using a specialized algorithm that computes all minimal hitting sets of the collection of MCSes. A drawback of CAMUS is that the set of MCSes must be computed before enumeration of MUSes can start. For formulas with an exponential number of MCSes, enumeration of MUSes is infeasible. A solution used in CAMUS is to compute PCSes instead of MCSes, thus sacrificing completeness of MUS enumeration. As shown in this paper, the enumeration of PCSes is also often infeasible.

The DAA algorithm (Bailey and Stuckey 2005) is also based on minimal hitting set dualization, but it is able to generate some MUSes before the enumeration of MCSes is completed. At each iteration a new MCS is computed. This is achieved using a *grow* procedure, that takes as input a satisfiable set $\mathcal{S}$ and adds to it clauses that do not make the set unsatisfiable. The resulting final $\mathcal{S}$ is an MSS and its complement is added to the set of MCSes. Each time the set of MCSes is expanded, all of its minimal hitting sets are computed. The computed minimal hitting sets represent the MUS candidates. For each candidate $\mathcal{M}$ a satisfiability check is performed. If one of these candidates is satisfiable, then it is used as the starting point for a new MSS. All $\mathcal{M}$ that are unsatisfiable are recorded as MUSes. As shown in (Liffiton and Sakallah 2008), the main performance bottlenecks of DAA are the number of SAT calls of the procedure, the computation of the minimal hitting sets, and testing whether each minimal hitting set is an MUS.

HYCAM (Grégoire, Mazure, and Piette 2007) proposes the use of local search to improve the performance of CAMUS. Approaches for enumeration of MUSes have also been analyzed in (Junker 2004).

The approach for partial MUS enumeration proposed in this paper (see next section) can be related with a recently algorithm for Maximum Satisfiability (MaxSAT) (Davies and Bacchus 2011), in that the maximal models (or the minimal hitting sets in (Davies and Bacchus 2011)) of a reference CNF formula serve to select a subset of the clauses of the original formula. Another tightly related approach has been concurrently proposed elsewhere (Liffiton and Malik 2013). As shown below, the approach proposed in this paper provides stronger guarantees for the selected subformulas.

New MUS Enumeration Algorithm

This section describes the new MUS enumeration algorithm EMUS. The algorithm addresses the drawbacks of previous approaches for MUS enumeration. First, and similarly to CAMUS, the new algorithm EMUS can compute MCSes.

```
   Input: CNF formula F
 1 begin
 2     I ← {p_i | c_i ∈ F}
 3     Q ← ∅
 4     while true do
 5         (st, P) ← MaximalModel(Q)
 6         if not st then return
 7         F' ← {c_i | p_i ∈ P}
 8         if not SAT(F') then
 9             M ← ComputeMUS(F')
10             ReportMUS(M)
11             b ← {¬p_i | c_i ∈ M}
12         else
               // Can report MCS {c_i | p_i ∈ I \ P}
13             b ← {p_i | p_i ∈ I \ P}
14         Q ← Q ∪ {b}
15 end
```

Algorithm 1: New direct MUS enumeration, EMUS

However, in contrast to CAMUS, complete MCS enumeration before MUS enumeration is not necessary. Second, and similarly to the approaches based on CS-trees (Han and Lee 1999; de la Banda, Stuckey, and Wazny 2003), EMUS iteratively selects subsets of clauses. However, in contrast with earlier work, these sets are not explicitly enumerated, and this provides remarkable performance improvements. Third, and similarly to earlier approaches, EMUS performs iterative SAT tests. However, in contrast with earlier work, these are kept to a minimum.

Algorithm 1 summarizes the main steps of EMUS. A new p_i variable is associated with each clause c_i of target formula $\mathcal{F}$. These p_i variables denote the variables of a second formula $\mathcal{Q}$. The algorithm works with these two formulas, $\mathcal{Q}$ and $\mathcal{F}$. The models of formula $\mathcal{Q}$, represent subformulas of $\mathcal{F}$. Each subformula of $\mathcal{F}$ is identified by the p_i variables assigned value 1. Thus, each model m of $\mathcal{Q}$ induces a subformula $\mathcal{F}'$ of $\mathcal{F}$. For Algorithm 1, each subformula $\mathcal{F}'$ is induced by a maximal model of $\mathcal{Q}$ (line 5). The computation of the maximal models of $\mathcal{Q}$ guarantees that the induced subformulas exhibit key properties, as shown below. Each subformula $\mathcal{F}'$ is checked for satisfiability (line 8). If the subformula $\mathcal{F}'$ is unsatisfiable, then it contains an MUS of $\mathcal{F}$. Since each computed MUS is blocked by adding to $\mathcal{Q}$ a clause of the form $(\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n)$ (line 11), each unsatisfiable subformula $\mathcal{F}'$ contains a *new* MUS of $\mathcal{F}$. If the subformula $\mathcal{F}'$ is satisfiable, then it represents an MSS of $\mathcal{F}$, and its complement represents an MCS of $\mathcal{F}$. Each MCS is blocked by adding a clause of the form $(p_1 \vee p_2 \vee \ldots \vee p_n)$ (line 13) to $\mathcal{Q}$, requiring the next models of $\mathcal{Q}$ to select at least one clause from this MCS. As a result, at each step the algorithm either computes a new MUS or a new MCS of $\mathcal{F}$. The algorithm terminates (line 6) when all MUSes and all MCSes have been enumerated. As shown below, when all MUSes and MCSes have been enumerated, $\mathcal{Q}$ becomes unsatisfiable.

Example 1. *Consider the example formula $\mathcal{F}$ shown in Fig-*

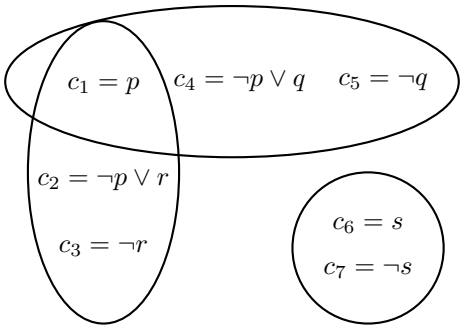

Figure 1: Example formula $\mathcal{F}$

Maximal model $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ sat/unsat	MUS/MCS
1111111	UNSAT	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	UNSAT	$\neg p_6 \vee \neg p_7$
0111101	SAT	$p_1 \vee p_6$
1011101	UNSAT	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	SAT	$p_3 \vee p_5 \vee p_7$
1010110	SAT	$p_2 \vee p_4 \vee p_7$
1100101	SAT	$p_3 \vee p_4 \vee p_6$
0111110	SAT	$p_1 \vee p_7$
1101001	SAT	$p_3 \vee p_5 \vee p_6$
1010101	SAT	$p_2 \vee p_4 \vee p_6$
1011001	SAT	$p_2 \vee p_5 \vee p_6$
1100110	SAT	$p_3 \vee p_4 \vee p_7$
1011010	SAT	$p_2 \vee p_5 \vee p_7$

Table 1: Example MUSes/MCSes

ure 1. The execution of the algorithm is summarized in Table 1. Each maximal model defines which clauses are selected and which are excluded in the next satisfiability test. For example, the model $m = 1011101$ of $\mathcal{Q}$ excludes the clauses c_2 and c_6. Thus, the MUS extracted from $\mathcal{F}' = \mathcal{F} \setminus \{c_2, c_6\} = \{c_1, c_3, c_4, c_5, c_7\}$ is $\{c_1, c_4, c_5\}$. As indicated above, the clauses of $\mathcal{Q}$ containing negated literals represent MUSes, while those containing positive literals represent MCSes. Note that the final formula $\mathcal{Q} = \{(\neg p_1 \vee \neg p_2 \vee \neg p_3), (\neg p_6 \vee \neg p_7), (p_1 \vee p_6), (\neg p_1 \vee \neg p_4 \vee \neg p_5), (p_3 \vee p_5 \vee p_7), (p_2 \vee p_4 \vee p_7), (p_3 \vee p_4 \vee p_6), (p_1 \vee p_7), (p_3 \vee p_5 \vee p_6), (p_2 \vee p_4 \vee p_6), (p_2 \vee p_5 \vee p_6), (p_3 \vee p_4 \vee p_7), (p_2 \vee p_5 \vee p_7)\}$ is unsatisfiable, denoting that all MUSes and MCSes have been enumerated.

To prove the correctness of Algorithm 1, let M denote the set of already computed MUSes of $\mathcal{F}$, and let C denote the set of already computed MCSes of $\mathcal{F}$. Thus, each element of M and C caused a new clause to be added to $\mathcal{Q}$.

Lemma 1. *Let $\mathcal{S}$ denote the clauses induced by a maximal model m of $\mathcal{Q}$ given M and C. Then, $\forall \mathcal{M} \in M$, $\mathcal{M} \not\subseteq \mathcal{S}$ and $\forall \mathcal{C} \in C$, $\mathcal{C} \not\subseteq \mathcal{F} \setminus \mathcal{S} \wedge \mathcal{F} \setminus \mathcal{S} \not\subseteq \mathcal{C}$.*

Proof. (Sketch) For each MUS or MCS, Algorithm 1 *blocks* (lines 11 and 13) subsequent maximal models from inducing subformulas that repeat either MSSes or MUSes. Thus each induced subformula does not contain an already computed MUS and its complement does not include and is not included in any already computed MCS. □

Lemma 2. *The subformula induced by each maximal model of $\mathcal{Q}$ either contains an MUS of $\mathcal{F}$ or is an MSS of $\mathcal{F}$.*

Proof. (Sketch) By induction on the number of iterations of the while loop in Algorithm 1. Let M and C be as defined above.
Base case: Since the initial value of $\mathcal{Q} = \emptyset$, the initial maximal model must be all 1s. Thus, the maximal model induces $\mathcal{F}$, which is unsatisfiable, and so it contains one MUS.
Inductive hypothesis: All elements in M and C represent, respectively, MUSes and MCSes.
Inductive step: A maximal model of $\mathcal{Q}$ given M and C, induces a subformula $\mathcal{F}'$ that does not contain any previously computed MUSes and MCSes (by Lemma 1). If $\mathcal{F}'$ is unsatisfiable, it contains an MUS of $\mathcal{F}$. Moreover, by Lemma 1, this MUS of $\mathcal{F}$ is not contained in M. Next, we consider the case when $\mathcal{F}'$ satisfiable. We claim that $\mathcal{F}'$ is an MSS of $\mathcal{F}$, and so $\mathcal{F} \setminus \mathcal{F}'$ is an MCS of $\mathcal{F}$. Observe that $\mathcal{F}'$ is induced by a maximal model of $\mathcal{Q}$. This means that no additional p_i variables could be assigned value 1 without unsatisfying some clause of $\mathcal{Q}$. Since no more p_i variables could be assigned value 1, then $\mathcal{F}'$ cannot be extended, and so by Definition 3 it is an MSS of $\mathcal{F}$. □

Theorem 1. *Algorithm 1 finds all the MUSes and terminates iff all MUSes and MCSes have been computed.*

Proof. At each step a new MUS or a new MCS is found (by Lemma 2 and Lemma 1), and so all MUSes are eventually enumerated.
We now want to show that $\mathcal{Q}$ is satisfiable if some MCSes or MUSes are missing. Suppose an MCS $\{c_1, \ldots, c_k\}$ is missing in $\mathcal{Q}$. Then consider an interpretation $\mathcal{I}$ that makes $\{p_1, \ldots, p_k\}$ *false* and all other p_i's *true*. Each MUS contains at least one element from this MCS and this element is *false* under interpretation $\mathcal{I}$. Therefore the interpretation satisfies all the clauses used for blocking MUSes including those that are contained in $\mathcal{Q}$. Now consider the other MCSes. As each MCS is minimal, each other MCS contains at least one element which is different from $\{p_1, \ldots, p_k\}$ and therefore it is satisfied by the interpretation $\mathcal{I}$. Accordingly, the interpretation $\mathcal{I}$ satisfies the blocking clauses used for all other MCSes including those contained in $\mathcal{Q}$. Hence, the interpretation $\mathcal{I}$ is a model of $\mathcal{Q}$ as it satisfies each clause of $\mathcal{Q}$. Now suppose a MUS $\{c_1, \ldots, c_k\}$ is missing in $\mathcal{Q}$. Consider an interpretation $\mathcal{I}$ that makes $\{p_1, \ldots, p_k\}$ *true* and all other p_i's *false*. Each MCS contains at least one element of this MUS and this element is *true* under interpretation $\mathcal{I}$. Therefore, the interpretation satisfies the blocking clauses of all MCSes including those contained in $\mathcal{Q}$. Now consider the other MUSes. As each MUS is minimal, each other MUS contains at least one element which is different to $\{p_1, \ldots, p_k\}$ and therefore satisfied by the interpretation $\mathcal{I}$. Accordingly, the interpretation $\mathcal{I}$ satisfies the blocking clauses used for all other MUSes including those contained in $\mathcal{Q}$. Hence, the interpretation $\mathcal{I}$ is a model of $\mathcal{Q}$ as

it satisfies each clause of $\mathcal{Q}$. We now show that when all the MUSes and MCSes have been found then formula $\mathcal{Q}$ is unsatisfiable. Suppose that all MUSes and MCSes have been computed and that the formula $\mathcal{Q}$ is satisfiable with maximal model m. By Lemma 2 the subformula induced by m either contains an MUS of $\mathcal{F}$ or is an MSS of $\mathcal{F}$. Moreover by Lemma 1 the induced subformula does not contain neither an already computed MUS nor an already computed MSS/MCS, contradicting the fact that all MUSes and MCSes have been computed. □

It is also relevant to analyze the *exact* number of iterations of the while loop in Algorithm 1. Clearly, given the previous results (see Lemmas 1 and 2, and Theorem 1), the number of iterations of the while loop matches the sum of the number of MCSes and MUSes, and so it grows exponentially with the formula size in the worst case.

Experimental Results

The new algorithm EMUS has been implemented in C++. The satisfiability test uses MiniSat (Eén and Sörensson 2003), version 2.2, a state of the art SAT solver. Maximal models are computed with SAT&PREF (Rosa, Giunchiglia, and Maratea 2010; Giunchiglia and Maratea 2012), a modification of MiniSat that allows solving SAT problems with qualitative preferences on literals. A maximal model is obtained by specifying a preference of 1 for the value of each variable. MUSes are computed with MUSer (Belov, Lynce, and Marques-Silva 2012), a state of the art MUS extractor. The experiments were performed on an HPC cluster, where each node is a dual quad-core Xeon E5450 3 GHz with 32 GB of memory. In all the experiments the memory limit was set to 4 GB, and the timeout was set to 3600 seconds.

Complete MUS Enumeration

The first experiment was to compare CAMUS (Liffiton and Sakallah 2008) and EMUS on complete MUS enumeration. Table 2 shows the results on a few selected instances for which complete enumeration is feasible (Liffiton and Sakallah 2008; Sinz, Kaiser, and Küchlin 2003). As can be concluded, CAMUS is in general several orders of magnitude faster than EMUS. This should come as no surprise, since EMUS does not target complete MUS enumeration. Observe that, when complete MUS enumeration is feasible, EMUS has several obvious drawbacks when compared to CAMUS. For each MUS, EMUS must extract that MUS, using an MUS extractor, whereas CAMUS uses an efficient hitting set dualization approach. Moreover, EMUS must compute one maximal model for each MCS, which is then used to decide the subset of the formula checked for satisfiability. This is significantly less efficient than MCS enumeration as used in CAMUS.

Nevertheless, the main motivation of EMUS is partial MUS enumeration, for problem instances for which neither CAMUS nor other existing approaches are capable of computing any MUSes. These results are analyzed next.

Partial MUS Enumeration

To evaluate the effectiveness of EMUS, we performed an experimental evaluation on 377 unsatisfiable instances. These are taken from different sources, including (Sinz, Kaiser, and Küchlin 2003; Liffiton and Sakallah 2008; Grégoire, Mazure, and Piette 2007; Safarpour et al. 2007) and simple unsatisfiable instances from the SAT competitions[1].

For many of these instances, it is unrealistic to enumerate *all* MUSes. As a result, instead of enumerating MCSes, we opted to compute PCSes of size 2. This choice was motivated by the fact that PCSes of size 2 are the easiest to compute, and given that computing PCSes of size 1 yields a single MUS. The goal was to select those instances for which CAMUS is unable to enumerate PCSes of size 2 in 3600 seconds. Observe that these are the most relaxed requirements, besides computing a single MUS.

Moreover, we also ran our implementation of DAA (Bailey and Stuckey 2005) on the same set of problem instances, and confirmed that no MUSes could be computed within the given time limit[2]. Observe that, given the experimental comparison between DAA and (de la Banda, Stuckey, and Wazny 2003) in (Bailey and Stuckey 2005), a re-implementation of the approach described in (de la Banda, Stuckey, and Wazny 2003) was deemed unnecessary. Moreover, the work of (de la Banda, Stuckey, and Wazny 2003) is based on explicit manipulation of subsets of the original formula using the CS-tree representation. This approach will not scale when analyzing CNF formulas with hundreds of thousands or millions of clauses. However, this is the case with the majority of the problem instances studied in this paper (see results below).

Out of the target 377 unsatisfiable instances, CAMUS is able to generate all PCSes of size 2 for 234. For the remaining 143, CAMUS times out without terminating the enumeration of PCSes of size 2. This means CAMUS would be unable to compute MUSes without checking each minimal hitting set for unsatisfiability as DAA does. Next, we ran EMUS on the 143 instances for which CAMUS cannot enumerate all PCSes of size 2 in 3600 seconds. For these instances, EMUS exceeds the available resources (either time or memory) on 13 instances, being able to enumerate more than 1 MUS for the remaining 130 instances. (Observe that computing a single MUS can be done with a standard MUS extractor (Belov, Lynce, and Marques-Silva 2012).) Of these 130 instances, the results for a selection of 50 (aiming to be representative of the results for the 130 instances) are shown in Table 3. The first column shows the names of each problem instance. The remaining columns show the number of MUSes and MCSes, respectively after 100s, 200s, 500s, 1000s, 2000s and 3600s (i.e. the timeout). The problem instances selected are fairly diverse. Some instances have less than 2000 thousand clauses (e.g. rocket_ext.b), whereas others have more than 1.7 million clauses (e.g. wb_4m8s4).

[1] http://www.satcompetition.org/2011/.

[2] We used the minimal hitting set algorithm of the second phase of CAMUS. Although efficient, the implementation is not incremental, and so this could represent a possible drawback. However, our results concur with those from (Liffiton and Sakallah 2008).

Instance	CAMUS #MUS/#MCS	time	EMUS #MUS/#MCS	time
C168_FW_UT_851.cnf	102/30	0.21	102/30	6.69
C170_FR_RZ_32.cnf	32768/242	0.31	32768/242	1764.24
C170_FR_SZ_58.cnf	218692/177	5.73	163328/177	3600(TO)
C208_FA_SZ_87.cnf	12884/139	0.63	12884/139	341.49
C208_FA_UT_3254.cnf	17408/155	0.38	17408/155	736.297
C220_FV_RZ_12.cnf	80272/150	1.15	80272/150	1698.02
C220_FV_RZ_13.cnf	6772/76	0.18	6772/76	162.90
C220_FV_SZ_65.cnf	103442/198	2.41	103442/198	2183.52

Table 2: CAMUS vs EMUS for complete MUS enumeration

As can be observed, the number of computed MUSes ranges from 0 (for an instance for which it is also impossible to compute one MCS) to more than 65000. For several instances, a non-negligible number of MUSes can be computed without enumerating any MCS (e.g. c499_gr_rcs_w5.shuffled). For some other instances, after enumerating some MUSes, it then becomes necessary to enumerate a large number of MCSes before being able to identify more MUSes (e.g. cache.inv8.ucl.sat.chaff.4.1).

It should be noted that for most of the problem instances considered, it is infeasible to generate all MUSes. Given the large number of clauses for most of these instances, either the number of MUSes, the number of MCSes (or both) is well beyond of what can conceivably be enumerated. This observation is confirmed by the inability of either CAMUS or (our own implementation of) DAA to produce any results for the 130 instances for which EMUS is able to partially enumerate MUSes.

Conclusions

This paper describes a novel algorithm for partial enumeration of MUSes, EMUS. The new algorithm excels on formulas for which either complete or partial enumeration of MUSes with existing approaches is infeasible. A key aspect of the proposed approach is the iterative selection of subformulas such that each one either contains an MUS or is an MSS of the target formula. Experimental results, obtained on a selection of well-known unsatisfiable formulas, demonstrate that the new algorithm complements current state of the art MUS enumeration approaches, being able to partially enumerate MUSes for problem instances for which other approaches are unable to. Moreover, several of these instances have in excess of a million clauses. These results are significant and advance the state of the art in the enumeration of MUSes.

Future research will pursue a number of lines of research. One line of research is to adapt the algorithm described in this paper for computing representative explanations (O'Sullivan et al. 2007). The use of SAT&PREF for computing maximal models can be adapted to generating candidate MUSes that are representative of the reasons of infeasibility. Another line of research is to investigate the integration of EMUS with existing approaches, e.g. CAMUS and DAA. Additional research directions include the integration of different MUS extraction algorithms (e.g. (Marques-Silva, Janota, and Belov 2013)), replacing SAT&PREF with more efficient MCS extraction algorithms (e.g. (Marques-Silva et al. 2013)), and extensions to more expressive constraints.

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

References

Andraus, Z. S.; Liffiton, M. H.; and Sakallah, K. A. 2008. Reveal: A formal verification tool for verilog designs. In *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, 343–352.

Bailey, J., and Stuckey, P. J. 2005. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *International Symposium on Practical Aspects of Declarative Languages*, 174–186.

Belov, A.; Lynce, I.; and Marques-Silva, J. 2012. Towards efficient MUS extraction. *AI Commun.* 25(2):97–116.

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

Büning, H. K., and Kullmann, O. 2009. Minimal unsatisfiability and autarkies. In *Handbook of Satisfiability*. IOS Press. 339–401.

Davies, J., and Bacchus, F. 2011. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Principles and Practice of Constraint Programming*, 225–239.

de la Banda, M. J. G.; Stuckey, P. J.; and Wazny, J. 2003. Finding all minimal unsatisfiable subsets. In *Conference on Principles and Practice of Declarative Programming*, 32–43.

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

	#MUS/#MCS					
Instance	100s	200s	500s	1000s	2000s	3600s
25s.smv.sat.chaff.4.1	12/19	24/19	61/19	122/19	240/19	427/20
9symml_gr_rcs_w5.shuffled	163/0	320/0	777/0	1511/0	2908/0	5017/0
SM_AS_TOP_buggy1	5/0	10/0	26/0	52/0	101/0	177/0
alu2_gr_rcs_w7.shuffled	0/0	0/0	1/0	2/0	4/0	5/0
apex7_gr_2pin_w4.shuffled	482/0	997/0	2229/0	4163/1	8909/1	16299/1
b14_opt_bug2_vec1-gate-0	6/0	13/0	33/0	65/0	127/0	227/0
b15-bug-fourvec-gate-0	1/0	2/0	5/0	12/0	25/0	47/0
bf0432-103	1167/355	2371/442	5721/467	10471/467	22683/1016	43007/1434
bf2670-241	965/54	1992/69	4915/103	9611/134	18897/154	33052/252
bf2670-492	16/6158	19/12192	44/25223	51/42975	121/74265	184/101911
bw_large.a	3557/0	7022/0	17132/0	32803/0	62749/0	109273/0
bw_large.d	5/0	10/0	25/0	49/0	98/0	169/0
c3_DD_s3_f1_e1_v1-bug-...	33/1	66/1	167/1	337/1	676/2	1227/2
c499_gr_rcs_w5.shuffled	219/0	442/0	1110/0	2154/0	4202/0	7320/0
c4_DD_s3_f1_e2_v1-bug-...	1/0	3/0	7/0	15/0	31/0	57/0
c5_DD_s3_f1_e1_v2-bug-...	4/0	8/0	20/0	40/0	80/0	144/1
c6288-bug-gate-0	568/1	1026/204	1125/3682	1164/9748	1278/21627	1310/39316
c6_DD_s3_f1_e2_v1-bug-...	4/0	8/0	22/0	43/2	88/2	160/2
c7552-bug-gate-0	740/611	1201/1750	2277/5740	3438/13496	6969/26906	12311/47443
c880_gr_rcs_w5.shuffled	56/0	115/0	285/0	552/0	1014/0	1770/0
c880_gr_rcs_w6.shuffled	0/0	0/0	0/0	0/0	0/0	0/0
cache.inv14.ucl.sat.chaff.4.1	0/0	0/0	0/0	0/0	1/0	3/1
cache.inv8.ucl.sat.chaff.4.1	633/45	1292/47	3188/209	6173/858	8835/9486	9521/30198
divider-problem.dimacs_1	3/0	6/0	16/0	32/0	63/0	113/0
dividers2	19/1	40/1	102/1	203/3	407/3	733/3
dividers6_hack	22/0	45/0	114/0	226/0	447/0	808/0
dividers_multivec1	6/0	12/0	31/0	62/0	119/0	214/0
dlx1c.rwmem.ucl.sat.chaff.4.1	8/1	17/4	44/6	89/6	178/6	327/6
elf.rf9.ucl.sat.chaff.4.1	4/15	10/15	28/19	59/26	119/34	218/36
example2_gr_rcs_w5.shuffled	80/0	114/0	207/0	353/0	644/0	1111/0
fpu3-problem.dimacs_18	8/0	17/0	22/39	44/77	131/79	269/79
fpu7_hack-problem.dimacs_23	4/1	9/2	26/3	65/3	151/3	291/3
i2c-problem.dimacs_25	1/0	2/0	5/0	11/0	24/0	40/0
i2c_master2	11/0	24/0	65/0	133/0	268/0	480/0
mrisc_mem2...dimacs_29	1/0	2/0	5/0	11/0	21/0	39/0
ooo.rf10.ucl.sat.chaff.4.1	0/0	0/0	0/0	1/2	3/4	6/5
ooo.rf8.ucl.sat.chaff.4.1	5/12	10/25	27/29	54/36	117/36	279/36
ooo.tag12.ucl.sat.chaff.4.1	4/7	8/28	22/29	49/29	108/30	207/30
rocket_ext.b	1603/7	3409/10	8783/18	17619/44	36053/101	66913/141
rsdecoder-problem.dimacs_34	4/0	8/0	20/0	40/0	76/0	137/0
rsdecoder3	2/0	4/0	12/0	24/0	45/0	81/0
rsdecoder_multivec1	1/0	3/0	8/0	15/0	29/0	52/0
s15850-bug-fourvec-gate-0	18/0	37/0	91/5	183/10	276/107	277/544
spi-problem.dimacs_42	0/0	1/0	2/0	4/0	9/0	16/0
too_large_gr_rcs_w5.shuffled	106/0	249/0	667/0	1193/0	2118/0	3382/0
vda_gr_rcs_w7.shuffled	0/0	0/0	0/0	0/0	0/0	0/0
wb-problem.dimacs_46	2/0	4/0	12/0	26/0	53/0	97/0
wb1	26/0	50/0	125/0	239/0	458/0	808/0
wb_4m8s4	1/0	2/0	5/0	10/0	21/0	38/0
wb_conmax1	1/0	3/0	9/0	18/0	42/0	83/0

Table 3: Partial enumeration results for EMUS

Giunchiglia, E., and Maratea, M. 2012. Algorithms for solving satisfiability problems with qualitative preferences. In *Correct Reasoning*, 327–344. Springer.

Grégoire, É.; Mazure, B.; and Piette, C. 2007. Boosting a complete technique to find MSS and MUS thanks to a local search oracle. In *International Joint Conference on Artificial Intelligence*, 2300–2305.

Han, B., and Lee, S.-J. 1999. Deriving minimal conflict sets by CS-trees with mark set in diagnosis from first principles. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 29(2):281–286.

Hunter, A., and Konieczny, S. 2006. Shapley inconsistency values. In *Conference on Principles of Knowledge Representation and Reasoning*, 249–259.

Hunter, A., and Konieczny, S. 2010. On the measure of conflicts: Shapley inconsistency values. *Artif. Intell.* 174(14):1007–1026.

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

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

Liffiton, M. H., and Sakallah, K. A. 2005. On finding all minimally unsatisfiable subformulas. In *Theory and Applications of Satisfiability Testing*, 173–186.

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

Liffiton, M. H.; Moffitt, M. D.; Pollack, M. E.; and Sakallah, K. A. 2005. Identifying conflicts in overconstrained temporal problems. In *International Joint Conference on Artificial Intelligence*, 205–211.

Marques-Silva, J.; Heras, F.; Janota, M.; Previti, A.; and Belov, A. 2013. On computing minimal correction subsets. In *International Joint Conference on Artificial Intelligence*.

Marques-Silva, J.; Janota, M.; and Belov, A. 2013. Minimal sets over monotone predicates in Boolean formulae. In *Computer Aided Verification*.

O'Sullivan, B.; Papadopoulos, A.; Faltings, B.; and Pu, P. 2007. Representative explanations for over-constrained problems. In *AAAI Conference on Artificial Intelligence*, 323–328.

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

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

Safarpour, S.; Mangassarian, H.; Veneris, A. G.; Liffiton, M. H.; and Sakallah, K. A. 2007. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer-Aided Design*, 13–19.

Schlobach, S.; Huang, Z.; Cornet, R.; and van Harmelen, F. 2007. Debugging incoherent terminologies. *J. Autom. Reasoning* 39(3):317–349.

Sinz, C.; Kaiser, A.; and Küchlin, W. 2003. Formal methods for the validation of automotive product configuration data. *AI EDAM* 17(1):75–97.

Stern, R. T.; Kalech, M.; Feldman, A.; and Provan, G. M. 2012. Exploring the duality in conflict-directed model-based diagnosis. In *AAAI Conference on Artificial Intelligence*.

Torlak, E.; Chang, F. S.-H.; and Jackson, D. 2008. Finding minimal unsatisfiable cores of declarative specifications. In *International Symposium on Formal Methods*, 326–341.

Torlak, E.; Vaziri, M.; and Dolby, J. 2010. MemSAT: checking axiomatic specifications of memory models. In *Conference on Programming Language Design and Implementation*, 341–350.

Xiao, G., and Ma, Y. 2012. Inconsistency measurement based on variables in minimal unsatisfiable subsets. In *European Conference on Artificial Intelligence*, 864–869.