On Efficient Computation of Variable MUSes

Anton Belov[1], Alexander Ivrii[3], Arie Matsliah[3], and Joao Marques-Silva[1,2]

[1] CASL, University College Dublin, Ireland
[2] IST/INESC-ID, Lisbon, Portugal
[3] IBM Research – Haifa, Israel

Abstract. In this paper we address the following problem: given an unsatisfiable CNF formula $\mathcal{F}$, find a minimal subset of variables of $\mathcal{F}$ that constitutes the set of variables in *some* unsatisfiable core of $\mathcal{F}$. This problem, known as *variable MUS (VMUS)* computation problem, captures the need to reduce the number of *variables* that appear in unsatisfiable cores. Previous work on computation of VMUSes proposed a number of algorithms for solving the problem. However, the proposed algorithms lack all of the important optimization techniques that have been recently developed in the context of (clausal) MUS computation. We show that these optimization techniques can be adopted for VMUS computation problem and result in multiple orders magnitude speed-ups on industrial application benchmarks. In addition, we demonstrate that in practice VMUSes can often be computed faster than MUSes, even when state-of-the-art optimizations are used in both contexts.

1 Introduction

Concise descriptions of the sources of inconsistency in unsatisfiable CNF formulas have traditionally been associated with Minimally Unsatisfiable Subformulas (MUSes). An MUS of a CNF formula is an unsatisfiable subset of its clauses that is minimal in the sense that any of its proper subsets is satisfiable. Development of efficient algorithms for computation of MUSes is an active area of research motivated by many applications originating from industry [10, 6, 11, 17, 12]. The most recent generation of MUS extraction algorithms is capable of handling large industrial formulas efficiently.

Additional ways of capturing sources of inconsistency in CNF formulas have been proposed. For example, in [9, 12] the inconsistency is analysed in terms of sets of clauses (the so called *groups* of clauses); efficient algorithms for the computation of *group-MUSes* have been developed in [12, 16]. Sources of inconsistency can also be described in terms of the sets of the *variables* of the formula. One such description, the *variable-MUS (VMUS)*, has been proposed in [4] — a variable-MUS of an unsatisfiable CNF formula $\mathcal{F}$ is a subset V of its variables that constitutes the set of variables of some unsatisfiable subformula of $\mathcal{F}$ and is minimal in the sense that no proper subset of V has this property. While [4] does not develop any VMUS extraction algorithms, in [6] several such algorithms have been proposed, and their applications have been pointed out. However, the proposed algorithms lack all the optimization techniques parallel to

[⋆] This work is partially supported by SFI grant BEACON (09/IN.1/I2618), and by FCT grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/123051/2010).

those that have been recently developed in the context of (clausal) MUS computation (e.g. [10, 2]) and are known to be essential for handling large industrial instances. This observation motivates the development of novel optimization techniques for VMUS computation algorithms.

Beside a pure scientific interest in the development of efficient algorithms for VMUS computation, this line of research is motivated by a number of possible industrially-relevant applications of VMUSes (e.g. [5]) and other related variable-based descriptions of inconsistency in CNF formulas. To this end, in this paper we make the following contributions. We formalize the VMUS computation problem and its extensions, and establish basic theoretical properties. We describe a number of optimization techniques to the basic VMUS computation algorithm presented in [6] and demonstrate empirically the multiple-order of magnitude improvements in the performance of the algorithm on the set of industrially-relevant benchmarks used for the evaluation of MUS extractors in SAT Competition 2011. We develop a relaxation-variable based constructive algorithm for VMUS extraction, based on the ideas proposed in [10]. We also describe a number of indirect approaches whereby the VMUS computation problem is translated to group-MUS computation problem, and evaluate these approaches empirically. Finally, we describe a number of potential industrial applications of VMUSes and its extensions.

2 Preliminaries

We focus on formulas in CNF (*formulas*, from hence on), which we treat as (finite) (multi-)sets of clauses. We assume that clauses do not contain duplicate variables.

Given a formula $\mathcal{F}$ we denote the set of variables that occur in $\mathcal{F}$ by $Var(\mathcal{F})$, and the set of variables that occur in a clause $C \in \mathcal{F}$ by $Var(C)$. An *assignment* τ for $\mathcal{F}$ is a map $\tau : Var(\mathcal{F}) \to \{0, 1\}$. By $\tau|_{\neg x}$ we denote the assignment $(\tau \setminus \{\langle x, \tau(x)\rangle\}) \cup \{\langle x, 1 - \tau(x)\rangle\}$. Assignments are extended to clauses and formulas according to the semantics of classical propositional logic. By $Unsat(\mathcal{F}, \tau)$ we denote the set of clauses of $\mathcal{F}$ falsified by τ. If $\tau(\mathcal{F}) = 1$, then τ is a *model* of $\mathcal{F}$. If a formula $\mathcal{F}$ has (resp. does not have) a model, then $\mathcal{F}$ is *satisfiable* (resp. *unsatisfiable*). By SAT (resp. UNSAT) we denote the set of all satisfiable (resp. unsatisfiable) CNF formulas.

A CNF formula $\mathcal{F}$ is *minimally unsatisfiable* if (i) $\mathcal{F} \in \mathsf{UNSAT}$, and (ii) for any clause $C \in \mathcal{F}$, $\mathcal{F} \setminus \{C\} \in \mathsf{SAT}$. We denote the set of minimally unsatisfiable CNF formulas by MU. A CNF formula $\mathcal{F}'$ is a *minimally unsatisfiable subformula (MUS)* of a formula $\mathcal{F}$ if $\mathcal{F}' \subseteq \mathcal{F}$ and $\mathcal{F}' \in \mathsf{MU}$. The set of MUSes of a CNF formula $\mathcal{F}$ is denoted by $\mathsf{MUS}(\mathcal{F})$. (In general, a given unsatisfiable formula $\mathcal{F}$ may have more than one MUS.)

A clause $C \in \mathcal{F}$ is *necessary* for $\mathcal{F}$ (cf. [8]) if $\mathcal{F} \in \mathsf{UNSAT}$ and $\mathcal{F} \setminus \{C\} \in \mathsf{SAT}$. Necessary clauses are often referred to as *transition* clauses. The set of all necessary clauses of $\mathcal{F}$ is precisely $\bigcap \mathsf{MUS}(\mathcal{F})$. Thus $\mathcal{F} \in \mathsf{MU}$ if and only if every clause of $\mathcal{F}$ is necessary. The problem of deciding whether a given CNF formula is in MU is DP-complete [13].

Motivated by several applications, minimal unsatisfiability and related concepts have been extended to CNF formulas where clauses are partitioned into disjoint sets called *groups* [9, 12].

Definition 1 (Group-Oriented MUS). *Given an explicitly partitioned unsatisfiable CNF formula $\mathcal{F} = \mathcal{G}_0 \cup \cdots \cup \mathcal{G}_n$, a* group oriented MUS *(or,* group-MUS*) of $\mathcal{F}$ is a subset $\mathcal{F}' = \mathcal{G}_0 \cup \mathcal{G}_{i_1} \cup \cdots \cup \mathcal{G}_{i_k}$ of $\mathcal{F}$ such that $\mathcal{F}'$ is unsatisfiable and, for every $1 \leq j \leq k$, $\mathcal{F}' \setminus \mathcal{G}_{i_j}$ is satisfiable.*

3 Variable-MUS computation problem, and generalizations

In this section we review the formal definition of VMUS computation problem and some of its basic properties, and generalize it in two ways (motivated by applications): the *interesting variables MUS computation problem (IVMUS)* and the *group-VMUS computation problem (GVMUS)*, the later being related to (clausal) group-MUS.

3.1 VMUS

Variable-MUSes of CNF formula $\mathcal{F}$ are defined in terms of subformulas *induced* by subsets of $Var(\mathcal{F})$[4].

Definition 2 (Induced subformula [4, 6]). *Let $\mathcal{F}$ be a CNF formula, and let V be a subset of $Var(\mathcal{F})$. The subformula of $\mathcal{F}$* induced *by V is the formula $\mathcal{F}|_V = \{C \mid C \in \mathcal{F}$ and $Var(C) \subseteq V\}$.*

Thus, $\mathcal{F}|_V$ is the set of *all* clauses of $\mathcal{F}$ that are defined *only* on variables from V. Alternatively, if we consider the variables in $Var(\mathcal{F}) \setminus V$ as *removed*, then $\mathcal{F}|_V$ is obtained from $\mathcal{F}$ by removing all clauses that contain at least one of the removed variables. Note that in general $Var(\mathcal{F}|_V)$ may be a strict subset of V — consider for example $\mathcal{F}|_{\{p,q\}} = \{(p)\}$ for $\mathcal{F} = \{(p \vee q \vee r), (p)\}$.

Clearly $V_1 \subset V_2 \subseteq Var(\mathcal{F})$ implies $\mathcal{F}|_{V_1} \subseteq \mathcal{F}|_{V_2}$, and so variable minimal unsatisfiability can be well defined as follows.

Definition 3 (Variable minimally unsatisfiable formula, VMU **[4]).** *A CNF formula $\mathcal{F}$ is called* variable minimally unsatisfiable *if $\mathcal{F} \in$ UNSAT and for any $V \subset Var(\mathcal{F})$, $\mathcal{F}|_V \in$ SAT. The set of all such CNF formulas $\mathcal{F}$ is denoted by* VMU.

It is not difficult to see that MU $\subset$ VMU: clearly, every minimally unsatisfiable formula is variable minimally unsatisfiable, while, for example, the formula $\{(p), (\neg p \vee q), (\neg q), (p \vee \neg q)\}$ is in VMU, but not in MU. Nevertheless, as shown in [4], just like MU, the language VMU is DP-complete. The complexity of decision problems associated with various subclasses of VMU is also given in [4].

A *variable-MUS*, or VMUS, of an unsatisfiable CNF formula $\mathcal{F}$, is defined by analogy with (clausal) MUS as follows.

Definition 4 (Variable-MUS, VMUS **[6]).** *Let $\mathcal{F}$ be unsatisfiable CNF formula. Then, a set $V \subseteq Var(\mathcal{F})$ is a* variable-MUS (VMUS) *of $\mathcal{F}$ if $\mathcal{F}|_V \in$ VMU, or, explicitly, $\mathcal{F}|_V \in$ UNSAT, and for any $V' \subset V$, $\mathcal{F}|_{V'} \in$ SAT[5]. The set of all VMUSes of $\mathcal{F}$ is denoted by* VMUS($\mathcal{F}$).

[4] This is the terminology used in [6]; in [4] these subformulas are called *projections* of $\mathcal{F}$.
[5] Note that, in general, for $\mathcal{F}' = \mathcal{F}|_V$ and $V' \subset V$, we have $\mathcal{F}'|_{V'} = \mathcal{F}|_{V'}$.

Thus, a VMUS V of an unsatisfiable formula $\mathcal{F}$ is a subset of $Var(\mathcal{F})$ that has exactly the property we are interested in: it is the set of variables of some unsatisfiable core $\mathcal{F}'$ of $\mathcal{F}$, and for no other unsatisfiable core $\mathcal{F}''$ of $\mathcal{F}$, $Var(\mathcal{F}'')$ is a strict subset of V.

Example 1. Let $\mathcal{F} = \{C_1, \ldots, C_5\}$, where $C_1 = (p)$, $C_2 = (q)$, $C_3 = (\neg p \vee \neg q)$, $C_4 = (\neg p \vee r)$, and $C_5 = (\neg q \vee \neg r)$. $\mathcal{F}$ is unsatisfiable, however is not variable minimally unsatisfiable, because for $V = \{p, q\}$, $\mathcal{F}|_V = \{C_1, C_2, C_3\} \in \mathsf{UNSAT}$. On the other hand, V is a VMUS of $\mathcal{F}$. Note that $\mathcal{F}' = \{C_1, C_2, C_4, C_5\}$ is a (clausal) MUS of F, however $Var(F') = \{p, q, r\}$ is *not* a VMUS of $\mathcal{F}$. (This example contradicts the claim from [6] that variables of any clausal MUS constitute a VMUS.)

We conclude this subsection with a definition of a *necessary variable*, analogous to that of a necessary clause.

Definition 5 (Necessary variable). *Let $\mathcal{F}$ be a CNF formula. A variable $v \in Var(\mathcal{F})$ is* necessary *for $\mathcal{F}$ if $\mathcal{F} \in \mathsf{UNSAT}$, and $\mathcal{F}|_{Var(\mathcal{F}) \setminus \{v\}} \in \mathsf{SAT}$.*

Thus, $\mathcal{F} \in \mathsf{VMU}$ if and only if every variable in $Var(\mathcal{F})$ is necessary for $\mathcal{F}$, and so V is a VMUS of $\mathcal{F}$, if every variable in V is necessary for $\mathcal{F}|_V$. Furthermore, the set of all necessary variables of $\mathcal{F}$ is precisely $\bigcap \mathsf{VMUS}(\mathcal{F})$. The following proposition establishes the property of necessary variables required for ensuring the correctness of VMUS computation algorithms presented in this paper.

Proposition 1 (Monotonicity). *Let $\mathcal{F}$ be an unsatisfiable CNF formula. If $v \in Var(\mathcal{F})$ is necessary for $\mathcal{F}$, then it is also necessary for any unsatisfiable subset $\mathcal{F}'$ of $\mathcal{F}$.*

Finally, from the perspective of clausal MUSes of an unsatisfiable CNF formula $\mathcal{F}$ we have that $v \in Var(\mathcal{F})$ is necessary for $\mathcal{F}$ if and only if every MUS of $\mathcal{F}$ includes some clause containing v.

3.2 Generalizations

The generalizations of VMUS computation problem developed in this section are motivated by some of the applications, which we describe in Section 5. In the first generalization, the set $Var(\mathcal{F})$ is partitioned into the set of *interesting* variables I and the set of *uninteresting* variables U, that is $Var(\mathcal{F}) = I \uplus U$, and the VMUSes are computed in terms of interesting variables only.

Definition 6 (Interesting-VMUS, IVMUS). *Let $\mathcal{F}$ be an unsatisfiable CNF formula, and let I and U be a partition of $Var(\mathcal{F})$ into the sets of* interesting *and* uninteresting *variables, respectively. Then, the* interesting-VMUS (IVMUS) *of $\mathcal{F}$ is a set of variables $V \subseteq I$ such that $\mathcal{F}|_{V \cup U} \in \mathsf{UNSAT}$, and for any $V' \subset V$, $\mathcal{F}|_{V' \cup U} \in SAT$.*

Thus, the uninteresting variables play the role analogous to that of the clauses in group $\mathcal{G}_0$ in group-MUS computation problem, which represent the clauses outside of the interesting constraints [12].

Clearly, for a formula $\mathcal{F}$, VMUS computation problem is a special case of IVMUS computation problem when all variables are interesting, i.e. $I = Var(\mathcal{F})$. Note that, as opposed to VMUSes, IVMUSes can be empty even if $I \neq \emptyset$. Consider, for example, the formula $\mathcal{F} = \{(p), (\neg p), (p \vee q), (\neg p \vee q), (\neg q)\}$. If $I = \{q\}$ and $U = \{p\}$, then IVMUS of $\mathcal{F}$ is empty set because we can remove all clauses with q while preserving unsatisfiability. If, on the other hand, $I = \{p\}$ and $U = \{q\}$, then IVMUS of $\mathcal{F}$ is $\{p\}$. This is also the case when $I = \{p, q\}$ and $U = \emptyset$.

Further generalization of IVMUS (and of VMUS) computation problem can be obtained by partitioning the set of interesting variables into disjoint *groups* of variables, and then analyzing the minimal unsatisfiability of the formula in terms of these groups. This is the parallel of group-MUS computation problem in the clausal context.

Definition 7 (Group-VMUS, GVMUS). *Let $\mathcal{F}$ be an unsatisfiable CNF formula, and let $U, I_1, \ldots, I_n$ be a partition of $Var(\mathcal{F})$ into the set of uninteresting variables U and the sets I_i of interesting variables called* groups. *Then, the* group-VMUS (GVMUS) *of $\mathcal{F}$ is a set of groups $G \subseteq \{I_1, \ldots, I_n\}$, such that for $V = U \cup \bigcup_{I \in G} I$, $\mathcal{F}|_V \in \mathsf{UNSAT}$, and for any $G' \subset G$ and $V' = U \cup \bigcup_{I \in G'}$, the formula $\mathcal{F}|_{V'}$ is in* SAT.

Thus, IVMUS computation problem is a special case of GVMUS computation problem where each group contains exactly one interesting variable.

4 Algorithms for VMUS computation

The algorithms described in this section are based on iterative invocations of a SAT solver. Specifically, the function call `SAT`$(\mathcal{F})$ accepts a CNF formula $\mathcal{F}$ and returns a tuple $\langle \mathsf{st}, \tau, \mathcal{U} \rangle$ with the following semantics: if $\mathcal{F} \in \mathsf{SAT}$, then st is set to *true* and τ is a model of $\mathcal{F}$; otherwise $\mathsf{st} = false$ and $\mathcal{U} \subseteq \mathcal{F}$ is an unsatisfiable core of $\mathcal{F}$[6].

We introduce an additional notation. Given a CNF formula $\mathcal{F}$ and a variable $v \in Var(\mathcal{F})$, by $\mathcal{F}^v = \{C \mid C \in \mathcal{F} \text{ and } v \in Var(C)\}$ we denote the set of clauses of $\mathcal{F}$ that contain v. Note that $\mathcal{F}|_{Var(\mathcal{F}) \setminus \{v\}} = \mathcal{F} \setminus \mathcal{F}^v$.

4.1 Hybrid VMUS Computation

Without the optimizations on lines RR, REF and VMR, Algorithm 1 (`VHYB`) represents a basic destructive algorithm for VMUS computation, similarly to the algorithm `Removal` proposed in [6]. Note that it also closely mimics the organization of a *hybrid* algorithm for MUS computation (cf. [11]).

The algorithm accepts an unsatisfiable CNF formula $\mathcal{F}$ as input, and maintains three datastructures: the set of necessary variables V (initially empty), the *working set of variables* V_w (initialized to $Var(\mathcal{F})$), and the *working formula* $\mathcal{F}_w$ (initialized to $\mathcal{F}$).

On each iteration of the while-loop (line 4) the algorithm removes a variable v from V_w and tests whether v is necessary for $\mathcal{F}_w$ (see Definition 5). This test is performed by invoking a SAT solver on the formula $\mathcal{F}_w \setminus \mathcal{F}_w^v$ (line 7). If the formula is unsatisfiable, v is not necessary, and the clauses of $\mathcal{F}_w^v$ are removed from $\mathcal{F}_w$ (line 9). Otherwise, v is necessary, and it is added to V.

[6] Some of the modern SAT solvers have the capability of producing unsatisfiable cores (although not necessarily minimal); for SAT solvers without this capability we can set $\mathcal{U} = \mathcal{F}$.

Algorithm 1: $\texttt{VHYB}(\mathcal{F})$ — Hybrid VMUS Computation

```
Input  : Unsatisfiable CNF Formula $\mathcal{F}$
Output: VMUS $V$ of $\mathcal{F}$
begin
1       $V \leftarrow \emptyset$                                  // VMUS under-approximation
2       $V_w \leftarrow Var(\mathcal{F})$                          // Working set of variables
3       $\mathcal{F}_w \leftarrow \mathcal{F}$                                  // Working formula
4       while $V_w \neq \emptyset$ do  // Inv: $\mathcal{F}_w = \mathcal{F}|_{V \cup V_w}$ and $\forall v \in V$ is nec. for $\mathcal{F}_w$
5           $v \leftarrow$ PickVariable($V_w$)
6           $V_w \leftarrow V_w \setminus \{v\}$
7           $(\mathsf{st}, \tau, \mathcal{U}) =$ SAT($\mathcal{F}_w \setminus \mathcal{F}_w^v$)
RR          $\mathcal{R} \leftarrow$ CNF($\neg\mathcal{F}_w^v$)                    // Redundancy removal
RR          $(\mathsf{st}, \tau, \mathcal{U}) =$ SAT($(\mathcal{F}_w \setminus \mathcal{F}_w^v) \cup \mathcal{R}$)
8           if $\mathsf{st} = \mathsf{false}$ then              // $v$ is not necessary for $\mathcal{F}_w$
REF             if $\mathcal{U} \cap \mathcal{R} = \emptyset$ then $V_w \leftarrow V_w \cap Var(\mathcal{U})$;        // Refinement
9               $\mathcal{F}_w \leftarrow \mathcal{F}_w|_{V \cup V_w}$
10          else                                      // $v$ is necessary for $\mathcal{F}_w$
11              $V \leftarrow V \cup \{v\}$
VMR             VModelRotation($\mathcal{F}_w, V, \tau$)        // $v \in V$ after this call
VMR             $V_w \leftarrow V_w \setminus V$
12      return $V$                       // $V \in \mathsf{VMUS}(\mathcal{F})$ and $\mathcal{F}_w = \mathcal{F}|_V \in \mathsf{VMU}$
end
```

The correctness of the algorithm follows from the the loop invariant presented on line 4. This invariant is trivially satisfied prior to any iteration of the loop, and its inductiveness can be easily established from the pseudocode. Since on every iteration one variable is removed from V_w, the loop eventually terminates, and on termination it holds that $\mathcal{F}_w = \mathcal{F}|_V$, and that every variable in V is necessary for $\mathcal{F}_w$. Hence, $\mathcal{F}|_V \in \mathsf{VMU}$ and $V \in \mathsf{VMUS}(\mathcal{F})$.

It comes as no surprise that the basic algorithm described above is not efficient for large CNF formulas — on every iteration exactly one variable is removed from V_w, and so the algorithm makes exactly $Var(\mathcal{F})$ SAT solver calls. This lack of scalability is demonstrated clearly in our experimental evaluation, presented in Section 6. In the context of MUS extraction a number of crucial optimization techniques have been proposed — these include clause-set refinement [10] to remove multiple unnecessary clauses in a single SAT solver call, recursive model rotation [1] to detect multiple necessary clauses in a single SAT solver call, and redundancy removal [17, 10] to make SAT instances easier to solve. We now describe the way these techniques can be adopted in the setting of VMUS computation.

Redundancy Removal The idea behind the *redundancy removal* technique is to add certain constraints to the formula $\mathcal{F}_w \setminus \mathcal{F}_w^v$ prior to the invocation of a SAT solver on line 7. These additional constraints are taken to be the clauses of the Plaisted-Greenbaum CNF transformation [14] of the propositional formula $\neg\mathcal{F}_w^v$, denoted by $CNF(\neg\mathcal{F}_w^v)$. These clauses are constructed as follows: for each $C \in \mathcal{F}_w^v$, create an auxiliary variable a_C, and the binary clauses $(\neg a_C \vee \neg l)$, for each literal $l \in C$; then add a clause $(\bigvee_{C \in \mathcal{F}_w^v} a_C)$. The correctness of the redundancy removal technique is guaranteed by the following proposition.

Proposition 2. *Let $\mathcal{F}$ be an unsatisfiable CNF formula. Then for any $v \in Var(\mathcal{F})$, $\mathcal{F} \setminus \mathcal{F}^v \in \mathtt{SAT}$ if and only if $(\mathcal{F} \setminus \mathcal{F}^v) \cup CNF(\neg \mathcal{F}^v) \in \mathtt{SAT}$.*

Proof. Let τ be any model of $\mathcal{F} \setminus \mathcal{F}^v$. Since $\mathcal{F} \in \mathsf{UNSAT}$, for any $\tau' \supset \tau$ that extends τ to $Var(\mathcal{F}^v) \setminus Var(\mathcal{F})$, we have $\tau'(\mathcal{F}^v) = 0$. Thus, for some extension τ'' of τ', $\tau''(CNF(\neg \mathcal{F}^v)) = 1$ [7]. Therefore τ'' is a model of $(\mathcal{F} \setminus \mathcal{F}^v) \cup CNF(\neg \mathcal{F}^v)$. The opposite direction holds trivially. □

The technique is integrated into `VHYB` by replacing line 7 with the two lines labeled RR. Even though the additional constraints imposed by $CNF(\neg \mathcal{F}^v)$ are redundant, they can help the SAT solver to prune the search space more efficiently, and in fact our experiments show that in practice this method leads to an improved performance.

Variable-Set Refinement *Variable-set refinement* is a technique for detection of unnecessary variables that takes advantage of the capability of modern SAT solvers to produce unsatisfiable cores. The technique is based on the following observation.

Proposition 3 (Variable-Set Refinement). *Let $\mathcal{U}$ be an unsatisfiable core of $\mathcal{F} \in \mathsf{UNSAT}$. Then, there exists $V \subseteq Var(\mathcal{U})$ such that $V \in \mathsf{VMUS}(\mathcal{F})$.*

Proof. Take V to be any VMUS of the formula $\mathcal{F}|_{Var(\mathcal{U})}$. □

When the outcome of SAT solver call is UNSAT (line 8), the unsatisfiable core $\mathcal{U}$ of the formula $(\mathcal{F}_w \setminus \mathcal{F}_w^v) \cup \mathcal{R}$, where $\mathcal{R} = CNF(\neg \mathcal{F}_w^v)$, can be used in the following way: if $\mathcal{U}$ does not include any of the clauses of $\mathcal{R}$, then any variable of the working set V_w outside of the set $Var(\mathcal{U})$ is not necessary for $\mathcal{F}_w$, and thus can be removed from V_w (line REF). This is because the condition $\mathcal{U} \cap \mathcal{R} = \emptyset$ guarantees that $\mathcal{U}$ is an unsatisfiable core of the formula $(\mathcal{F}_w \setminus \mathcal{F}_w^v)$, and so Proposition 3 applies. Note that since the set V contains variables that are necessary for $\mathcal{F}_w$, it must be that $V \subseteq Var(\mathcal{U})$. If the unsatisfiable core $\mathcal{U}$ does include some of the clauses of $\mathcal{R}$, the formula $\mathcal{U} \setminus \mathcal{R}$ could be satisfiable, and so some of the variables necessary for $\mathcal{F}_w$ might be outside $Var(\mathcal{U})$. As an example, consider $\mathcal{F} = \{(p \vee q), (p \vee \neg q), (\neg p \vee r), (\neg p \vee \neg r), (\neg p \vee s), (\neg p \vee \neg s)\}$. Suppose we try to remove the variable r first. $\mathcal{F}^r = \{(\neg p \vee r), (\neg p \vee \neg r)\}$, and so $\mathcal{F} \setminus \mathcal{F}^r \in \mathsf{UNSAT}$. As the clauses of $\mathcal{R} = CNF(\neg \mathcal{F}^r)$ imply the unit clause (p), the returned unsatisfiable core might consist only of the clauses $\{(\neg p \vee s), (\neg p \vee \neg s)\}$ together with the clauses of $\mathcal{R}$. Note that the variable q is necessary for $\mathcal{F}$ but not included in this core.

It should be noted that, just as in the case of MUS extraction, variable-set refinement is a *crucial* optimization technique that leads to dramatic (multiple orders of magnitude) speed-ups in VMUS computation — see Section 6 for the empirical data.

[7] τ'' extends τ' by assigning values to the auxiliary variables introduced by Plaisted-Greenbaum transform.

Algorithm 2: `VModelRotation`$(\mathcal{F}, V, \tau)$ — Variable-based Model Rotation

```
Input : $\mathcal{F}$ — an unsatisfiable CNF formula
      : $V$ — a set of necessary variables $\mathcal{F}$
      : $\tau$ — a model of $\mathcal{F} \setminus \mathcal{F}^v$ for some $v \in Var(\mathcal{F})$
Effect: $V$ contains $v$ and possibly additional necessary variables of $\mathcal{F}$
1 begin
2     $V' \leftarrow \bigcap_{C \in Unsat(\mathcal{F},\tau)} Var(C)$        // all common variables; $v \in V'$
3     foreach $v \in V'$ do
4         if $v \notin V$ then
5             $V \leftarrow V \cup \{v\}$            // $v$ is a new necessary variable
6             $\tau' \leftarrow \tau|_{\neg v}$
7             VModelRotation$(\mathcal{F}, V, \tau')$
8 end
```

Variable-based Model Rotation (VMR) *Variable-based model rotation (VMR)* is a technique for detection of multiple necessary variables in a single SAT call. The technique makes use of the satisfying assignment τ returned by the SAT solver on line RR of Algorithm 1 (or line 7 if redundancy removal is not used). The technique uses the following property.

Proposition 4. *Let $\mathcal{F}$ be an unsatisfiable CNF formula, let τ be an assignment to $Var(\mathcal{F})$, and let $V = \bigcap_{C \in Unsat(\mathcal{F},\tau)} Var(C)$. Then any variable in V is necessary for $\mathcal{F}$.*

Proof. Take any $v \in V$. Clearly $Unsat(\mathcal{F}, \tau) \subseteq \mathcal{F}^v$, and so $\mathcal{F}|_{Var(\mathcal{F})\setminus\{v\}} = \mathcal{F} \setminus \mathcal{F}^v \subseteq \mathcal{F} \setminus Unsat(\mathcal{F}, \tau)$. Since $\mathcal{F} \setminus Unsat(\mathcal{F}, \tau) \in \texttt{SAT}$, we have $\mathcal{F}|_{Var(\mathcal{F})\setminus\{v\}} \in \texttt{SAT}$ □

An assignment τ is called a *witness for necessity of a variable v in $\mathcal{F}$* if satisfies the condition $v \in \bigcap_{C \in Unsat(\mathcal{F},\tau)} Var(C)$. Note that in the context of Algorithm 1 the assignment τ returned by the SAT solver in the case the formula $(\mathcal{F}_w \setminus \mathcal{F}_w^v) \cup \mathcal{R}$ is satisfiable is exactly a witness of necessity of v in $\mathcal{F}_w$ — it must be that $Unsat(\mathcal{F}_w, \tau) \subseteq \mathcal{F}_w^v$ and so v appears in all clauses of $Unsat(\mathcal{F}_w, \tau)$ (and, in fact, in the same polarity). Furthermore, it is possible that there are other variables shared among the clauses of $Unsat(\mathcal{F}_w, \tau)$. These variables can be immediately declared as necessary for $\mathcal{F}_w$, *without additional SAT calls*. A special case of particular practical importance is when $Unsat(\mathcal{F}_w, \tau)$ consists of a single clause — in this case all of the other variables of this clause are necessary as well.

Next, suppose that τ is a witness for (the necessity of) v. Note that the assignment $\tau' = \tau|_{\neg v}$, obtained by flipping the value of v in τ, is *also* a witness for v — indeed all the clauses in $Unsat(\mathcal{F}_w, \tau)$ are satisfied by τ', and all the clauses falsified by τ' must share the variable v (in the opposite polarity). Thus τ' can be analyzed in the same manner as τ — if there are any variables beside v that are shared among the clauses of $Unsat(\mathcal{F}_w, \tau')$, then these variables are necessary for $\mathcal{F}_w$ and τ' is the witness for each of these variables. This leads to the recursive process of detection of necessary variables and construction of witnesses, which we refer to as *variable-based model rotation (VMR)*.

The algorithm for VMR is presented in Algorithm 2. VMR is integrated into Algorithm 1 by replacing line 11 with the lines labelled VMR. Note that the necessary variable v detected by the SAT call in Algorithm 1 is always added to the set V as a result of VMR. The purpose of the if-statement on line 4 of Algorithm 2 is to prevent the algorithm from re-detecting variables that are already known to be necessary. In turn, this bounds the number of recursive calls to VMR during the execution of VMUS computation algorithm – this bound is twice the number of variables in the computed VMUS. Thus VMR is a light-weight technique for detection of necessary variables. In practice, however, the technique is extremely effective — as demonstrated in Section 6, VMR results in very significant performance improvements (up to the factor of 20).

The basic VMR algorithm described above allows for many various modifications. For example, one can allow the algorithm to re-visit variables already known to be necessary — it is very likely that a different initial model will lead to discovering a different set of new necessary variables. However, it is important to bound the total running time of the algorithm (and in particular to avoid loops). One specific modification of this type, which we refer to as *extended* VMR (EVMR), takes advantage of the diversity of the initial models passed to Algorithm 2 over the life-time of VMUS computation. In this variant the set V in Algorithm 2 should be thought of as the set of variables found necessary in the current invocation of VMR from Algorithm 1 (in other words, Algorithm 1 always calls Algorithm 2 with $V = \emptyset$, and at the end updates its own set V of all necessary variables to include the newly discovered variables). Although in the worst case the number of recursive calls to VMR can grow quadratically, in our experiments such growth was not observed. On the other hand, the number of necessary variables detected by EVMR was on average 5% higher than that of VMR alone (recall that every additional variable saves a potentially expensive SAT call). Overall, the EVMR modification has a positive impact on the performance of Algorithm 1.

Extensions Algorithm 1 can be extended to handle the generalizations to IVMUS or to GVMUS introduced in section 3. These extensions, with the exception of VMR in GVMUS setting, are rather straightforward, and we do not describe them explicitly.

4.2 Relaxation-variables approach

A constructive MUS extraction algorithm based on relaxing clauses and AtMost1 constraint has been proposed in [10]. The idea is to augment each clause C_i of the unsatisfiable input formula $\mathcal{F}$ with a fresh *relaxation* variable a_i, thus replacing C_i with $(a_i \vee C_i)$ in $\mathcal{F}$. Furthermore, the CNF representation of the constraint $\sum a_i \leq 1$ is added to the modified formula. The resulting formula $\mathcal{F}'$ is then checked for satisfiability. If $\mathcal{F}'$ is satisfiable, then since the original formula $\mathcal{F}$ is unsatisfiable, exactly one of the variables a_i must be set to true — the associated clause C_i is then necessary for $\mathcal{F}$. The algorithm is quite special in that it essentially offloads the task of *searching* for a necessary clause to the SAT solver.

This algorithm can be extended to VMUS in the following way. The first step consists of relaxing variables instead of clauses. Let v be a variable of $\mathcal{F}$. The positive literals of v are replaced with a new variable v_p, and the negative literals of v are replaced with a new variable v_n. If both v_n and v_p are assigned value 1, then variable v

is said to be *relaxed*. To control the relaxation of v, another variable v_r is used, and the constraint $(\neg v_p \vee \neg v_n \vee v_r)$ is added to the formula. Note that when v_r is set to 1, v_p and v_n can be assigned by the SAT solver to 1 freely, effectively removing the clauses of $\mathcal{F}^v$. Otherwise, the constraint represents the relationship between v_p and v_n. The new variables are represented by sets V_P, V_N and V_R, where V denotes the initial set of variables. Now, a variable is necessary for $\mathcal{F}$ if by relaxing (or removing) it the formula becomes satisfiable — similar to the clausal version of the algorithm, this condition achieved with the constraint $\sum_{v_r \in V_R} v_r = 1$.

4.3 Reduction to group-MUS computation

An alternative approach to handle the VMUS computation problem is to provide a reduction to a MUS (or to a group-MUS) computation — especially in the light of the extensive amount of tools capable solving the latter problem efficiently. We sketch here two possible reductions, one suited for dense formulas (with #vars $\ll$ #clauses), and the other for sparse formulas (with #vars $\approx$ #clauses).

Reduction from VMUS to group-MUS (for dense formulas) Let $\mathcal{F}$ be a CNF formula. We translate it to $\mathcal{F}'$ as follows. For each variable $v_i \in Vars(\mathcal{F})$ introduce two new variables p_i and n_i in $\mathcal{F}'$, and translate all clauses $C \in \mathcal{F}$ to clauses $C' \in \mathcal{F}'$ over variables $\{p_i, n_i\}$ by replacing every positive occurrence of v_i with p_i, and every negative occurrence v_i with n_i. As an example, the clause $(v_1 \vee \neg v_3 \vee v_4)$ gets translated to the clause $(p_1 \vee n_3 \vee p_4)$. Additionally, add $|Vars(\mathcal{F})|$ pairs of clauses $\{(p_i, n_i), (\neg p_i, \neg n_i)\}$ to $\mathcal{F}'$ (forcing the positive and negative representatives to have opposite values).

Define $\mathcal{G}_0 = \{C'\}$ to be the group consisting of all of the translated clauses, and for each i define the group $\mathcal{G}_i = \{(p_i, n_i), (\neg p_i, \neg n_i)\}$.

Proposition 5. *Any group-MUS S of $\mathcal{F}'$ (w.r.t. groups $\mathcal{G}_i$) corresponds to some VMUS V of the original formula $\mathcal{F}$, where $v_i \in V$ iff $\mathcal{G}_i \in S$.*

(Proof follows by construction.) It is also straightforward to extend this reduction to be applicable for IVMUS and GVMUS problems.

Reduction from VMUS to group-MUS (for sparse formulas) Let $\mathcal{F}$ be a CNF formula. We translate it to $\mathcal{F}'$ as follows. For each variable $v_i \in Vars(\mathcal{F})$ introduce an activation variable a_i, and translate all clauses $C \in \mathcal{F}$ to clauses $C' \in \mathcal{F}'$ that contain the same set of literals plus their corresponding activation literals (namely C' is double in size). As an example, the clause $(v_1 \vee \neg v_3 \vee v_4)$ gets translated to the clause $(v_1 \vee a_1 \vee \neg v3 \vee a_3 \vee v_4 \vee a_4)$. Additionally, add $|Vars(\mathcal{F})|$ unit clauses $(\neg a_i)$ to $\mathcal{F}'$.

Define $\mathcal{G}_0 = \{C'\}$ to be the group consisting of all of the translated clauses, and for each i define the group $\mathcal{G}_i = \{(\neg a_i)\}$.

Proposition 6. *Any group-MUS S of $\mathcal{F}'$ (w.r.t. groups $\mathcal{G}_i$) corresponds to some VMUS V of the original formula $\mathcal{F}$, where $v_i \in V$ iff $\mathcal{G}_i \in S$.*

Here too, proof follows by construction, and the reduction is easily extended to IVMUS and GVMUS problems.

5 Applications of VMUSes

A number of applications of VMUSes have been proposed in the previous work. In [6] the authors use VMUSes to search for vertex critical subgraphs in the context of graph coloring problem. There the variables of the CNF representation of a graph correspond to its vertices, and thus a removal of a variable represents the operation of the removal of a vertex from the graph. The authors of [5] propose to use VMUSes for computing abstractions in the context of the abstraction-refinement approach to model checking. We describe a possible application of GVMUSes in the similar context below. We also describe a possible application of IVMUSes to minimization of satisfying assignments.

Minimizing satisfying assignments Many formal verification techniques require the ability to efficiently generalize bad or interesting assignments to inputs of a circuit — the assignments which by propagation induce a value of 1 on one of the circuit's outputs. Similarly to [15], the problem is formalized as follows. Let F be a CNF with variables $Var(F) = J \uplus W \uplus \{o\}$ separated into a set of input variables J, a set of auxiliary variables W, and the property output o. We require that F satisfies the following property: for any satisfying assignment A to $F \wedge o$, the formula $A_J \wedge F \wedge \neg o$ is unsatisfiable, where A_J denotes the restriction of A to J. In this setting the set of "important" variables I is a subset of J and the set of of "unimportant" variables $U = (J \setminus I) \cup W \cup \{o\}$ consists of the remaining variables. Given such a formula F and an assignment A to $F \wedge o$, the goal is to find a minimal subset V of I which is still sufficient to force the value of 1 on the output, i.e. that the formula $A_V \wedge A_{J \setminus I} \wedge F \wedge \neg o$ remains unsatisfiable. This is precisely the IVMUS problem defined in Section 3.

Proof-based abstraction A popular technique to reduce the size of models in model checking is to create a quality over-approximation of the design under verification. In the abstraction refinement approach one usually unrolls the design for a certain number k of time-frames and runs a SAT solver with the property that there is no erroneous execution within this bound. If the SAT solver returns UNSAT, the unsatisfiable core of the problem is analyzed. In the latch-based abstraction (cf [7]) any latch that does not appear in the core is abstracted away, and it is usually beneficial to remove as many latches as computationally feasible. Though MUS algorithms have been developed for this problem (e.g. [12]), a translation to GVMUS problem is also possible: for every latch L in the design, create a group $\mathcal{G}_L$ of CNF variables corresponding to L in every time-frame of the unrolled design; all the remaining CNF variables are "non-interesting" and put into $\mathcal{G}_0$. The minimization of the set of "interesting" variable groups while preserving the unsatisfiability is precisely the GVMUS problem (cf. Section 3).

Remark 1. An extra care should be taken when reducing problems to VMUS (or generalizations) in applications where CNF formulas arise from encoding of circuits. Certain variables should be split into two copies — a variable x is split into x_i and x_o if it satisfies the following conditions: (i) x represents an output of a gate/latch in the circuit, and (ii) the fan-out of the corresponding gate/latch is greater than 1. For every such x, the clauses that encode $x_i = x_o$ are added to the formula encoding the circuit, and the representative variable for the gate/latch is taken to be x_i.

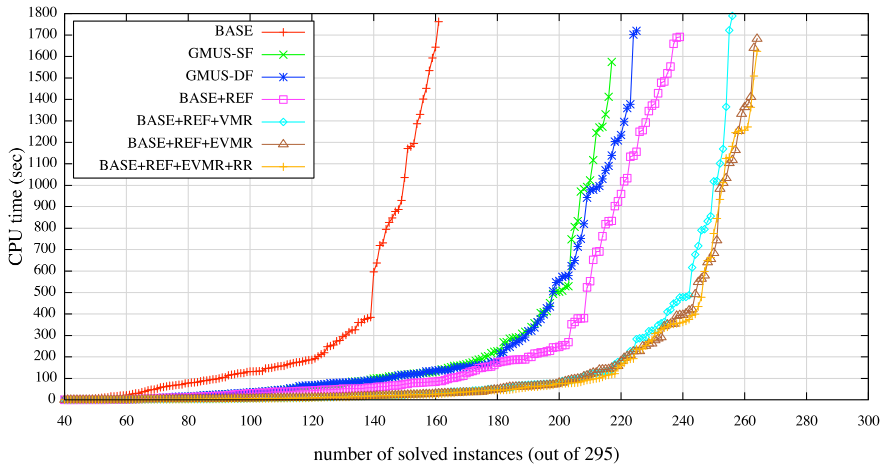

Fig. 6.1. Cactus plot: BASE with various optimizations; translations to GMUS.

6 Experimental Study

We implemented the VMUS extraction algorithm `VHYB` (Algorithm 1) and all of the optimization techniques described in Section 4 on top of the MUS extraction framework of our state-of-the-art (group-)MUS extractor `MUSer2`[8]. In addition, we implemented the translations to GMUS described in Section 4.3. We did not implement the relaxation-variable based algorithm from Section 4.2, since, in the context of (clausal) MUS extraction, this approach is notably less effective than the hybrid approach (cf. [10]).

To evaluate the effectiveness of the proposed techniques we performed a comprehensive experimental study on 295 benchmarks used in the MUS track of SAT Competition 2011 [9]. These benchmark instances originate from various industrial applications of SAT, including hardware bounded model checking, FPGA routing, hardware and software verification, equivalence checking, abstraction refinement, design debugging, functional decomposition and bioinformatics. The experiments were performed on an HPC cluster, where each node is dual quad-core Intel Xeon E5450 3 GHz with 32 GB of memory. The timeout was set to 1800 seconds, and memory was limited to 4 GB. In our experiments, `MUSer2`, and its VMUS-oriented version, was configured to use `picosat-935` [3] in incremental mode as a SAT solver.

The cactus plot in Fig. 6.1 provides an overview of the results of our experimental study[10]. The legend in this, and subsequent, plots is as follows: BASE represents the implementation of `VHYB` without any of the optimizations (i.e. this is the equivalent of the `Removal` algorithm from [6]); +REF represents the addition of variable-set refinement; +VMR (resp. +EVMR) represents the addition of VMR (resp. EVMR); +RR

[8] `http://logos.ucd.ie/wiki/doku.php?id=muser`

[9] `http://www.satcompetition.org/`. Note that the website lists 300 benchmarks in MUS category — this discrepancy is due to duplicate instances.

[10] Data is available at `http://logos.ucd.ie/paper-results/sat12-vmus`

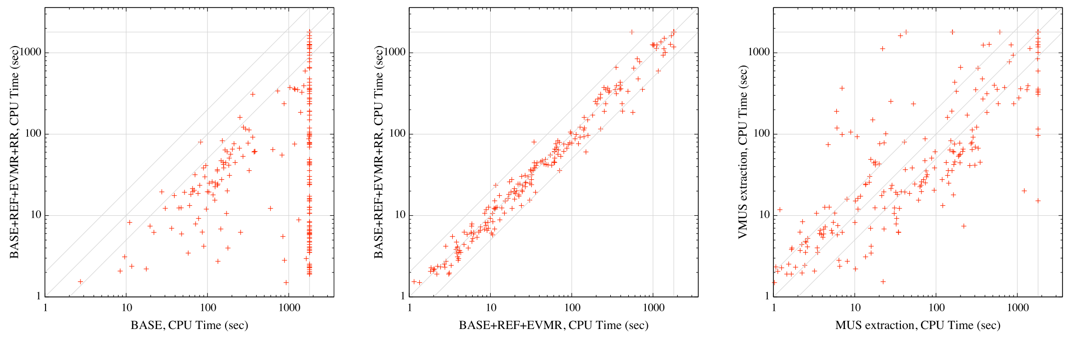

Fig. 6.2. Selected scatter plots (timeout 1800 sec.)

represents the addition of redundancy removal; GMUS-SF (resp. GMUS-DF) represents the results of the group-MUS version of `MUSer2` on the GMUS translation for sparse (resp. dense) formulas. A number of observations can be made. We note that the addition of variable-set refinement has a dramatic effect on the performance of `VHYB` allowing to solve 80 more instances than BASE within the timeout. The addition of VMR to the variable-set refinement results in another huge leap in the performance of `VHYB`, demonstrating the positive impact of the extended version of VMR (EVMR). We also conclude that the proposed translations of VMUS computation problem to GMUS computation does not allow to extract VMUSes efficiently, despite the fact that the group-MUS extractor `MUSer2` used in these experiments implements the group-MUS analog of the all crucial optimizations described in this paper — the exceptions are the *extended* model rotation and redundancy removal.

The scatter plots in Fig. 6.2 present additional views on the results of our experimental evaluation. The plot on the left-hand side demonstrates the combined effect of all the optimization techniques described in this paper to the VMUS extraction algorithm of [6]. Note the positive impact on the effectiveness of the algorithm, resulting in up to 3 orders of magnitude improvement in runtime of VMUS computation. The plot in the center of Fig. 6.2 allows to take a closer look at the effects of redundancy removal. While the results are mixed, a careful examination of the plot reveals an overall positive impact (close to 15%) of the technique on the runtime of `HYB`. The plot on the right-hand side of Fig. 6.2 deserves special attention. The plot compares the runtimes of VMUS extraction with the runtime of MUS extraction on the same set of instances. MUS extraction was performed with `MUSer2`, and we employed all the relevant optimizations, including the extended version of recursive model rotation [1]. We observe that in many cases it is cheaper to compute a VMUS of an instance, rather than the (clausal) MUS. Note that in the presence of the current MUS optimization techniques, such as clause set refinement and recursive model rotation, this observation is not trivial, and suggests that some of the current applications of MUS extraction algorithms could be reconsidered with VMUSes in mind.

Conclusion In this paper we re-visit the VMUS computation problem and propose a number of its extensions. We develop a state-of-the-art VMUS extraction algorithm

VHYB by introducing a number of optimization techniques to the basic VMUS extraction algorithm proposed in [6]. In addition, we present a relaxation-variable based constructive algorithm for VMUS extraction, and a number of translations of VMUS computation problem to group-MUS computation problem. We demonstrate empirically that the optimization techniques employed in VHYB lead to significant improvements in the runtime of VMUS computation on a set of industrially-relevant benchmarks. We also show that the indirect approach via group-MUS computation can be significantly less efficient than VHYB. Finally, we demonstrate that in many cases computation of VMUSes is cheaper than the computation of MUSes. This observation motivates re-evaluation of the current applications of MUSes, further investigation of the applications of VMUSes and its extensions, and the development of relevant algorithms.

Acknowledgements We thank the anonymous referees for insightful comments.

References

1. Anton Belov and Joao Marques-Silva. Accelerating MUS extraction with recursive model rotation. In *Formal Methods in Computer-Aided Design*, 2011.
2. Anton Belov and Joao Marques-Silva. Minimally unsatisfiable Boolean circuits. In *Theory and Applications of Satisfiability Testing*, pages 145–158, 2011.
3. A. Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.
4. Zhen-Yu Chen and De-Cheng Ding. Variable minimal unsatisfiability. In *Proc. of TAMC'06*, pages 262–273. Springer-Verlag, 2006.
5. Zhen-Yu Chen, Zhi-Hong Tao, Hans Kleine Büning, and Li-Fu Wang. Applying variable minimal unsatisfiability in model checking. *Journal of Software*, 19(1):39–47, 2008.
6. C. Desrosiers, P. Galinier, A. Hertz, and S. Paroz. Using heuristics to find minimal unsatisfiable subformulas in satisfiability problems. *J. Comb. Optim.*, 18(2):124–150, 2009.
7. Aarti Gupta, Malay K. Ganai, Zijiang Yang, and Pranav Ashar. Iterative abstraction using sat-based bmc with proof analysis. In *ICCAD*, pages 416–423, 2003.
8. Oliver Kullmann, Inês Lynce, and Joao Marques-Silva. Categorisation of clauses in conjunctive normal forms: Minimally unsatisfiable sub-clause-sets and the lean kernel. In *Theory and Applications of Satisfiability Testing*, pages 22–35, 2006.
9. Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008.
10. J. Marques-Silva and I. Lynce. On improving MUS extraction algorithms. In *Proc. of SAT 2011*, 2011.
11. Joao Marques-Silva. Minimal unsatisfiability: Models, algorithms and applications. In *International Symposium on Multiple-Valued Logic*, pages 9–14, 2010.
12. Alexander Nadel. Boosting minimal unsatisfiable core extraction. In *Formal Methods in Computer-Aided Design*, October 2010.
13. Christos H. Papadimitriou and David Wolfe. The complexity of facets resolved. *J. Comput. Syst. Sci.*, 37(1):2–13, 1988.
14. D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.
15. Kavita Ravi and Fabio Somenzi. Minimal assignments for bounded model checking. In *TACAS*, pages 31–45, 2004.
16. Vadim Ryvchin and Ofer Strichman. Faster extraction of high-level minimal unsatisfiable cores. In *Theory and Applications of Satisfiability Testing*, pages 174–187, 2011.
17. Hans van Maaren and Siert Wieringa. Finding guaranteed MUSes fast. In *Theory and Applications of Satisfiability Testing*, pages 291–304, 2008.