Smallest MUS Extraction with Minimal Hitting Set Dualization

Alexey Ignatiev[1], Alessandro Previti[2], Mark Liffiton[3], and Joao Marques-Silva[1,2]

[1] IST/INESC-ID, Technical University of Lisbon, Portugal
[2] CASL, University College Dublin, Ireland
[3] Illinois Wesleyan University, Illinois, USA

Abstract. Minimal explanations of infeasibility are of great interest in many domains. In propositional logic, these are referred to as Minimal Unsatisfiable Subsets (MUSes). An unsatisfiable formula can have multiple MUSes, some of which provide more insights than others. Different criteria can be considered in order to identify a good minimal explanation. Among these, the size of an MUS is arguably one of the most intuitive. Moreover, computing the smallest MUS (SMUS) finds several practical applications that include validating the quality of the MUSes computed by MUS extractors and finding equivalent subformulae of smallest size, among others. This paper develops a novel algorithm for computing a smallest MUS, and we show that it outperforms all the previous alternatives pushing the state of the art in SMUS solving. Although described in the context of propositional logic, the presented technique can also be applied to other constraint systems.

1 Introduction

For inconsistent formulae, finding a minimal explanation of their infeasibility is a central task in order to disclose the source of the problem. In general, an inconsistent formula can have multiple explanations. Thus, defining a measure of the quality of an explanation becomes necessary in order to focus on those providing more insights. For the case of a propositional formula, a natural measure of explanation quality is the size of the computed minimal unsatisfiable subsets (MUSes). From a query complexity perspective, extracting an MUS is in $\mathrm{FP}^{\mathrm{NP}}$. In contrast, deciding whether there exists an MUS of size less than or equal to k is Σ_2^{P}-complete [12,8]. As a consequence, extracting a smallest MUS (SMUS) is in $\mathrm{FP}^{\Sigma_2^{\mathrm{P}}}$.

Computing an SMUS is central to a number of practical problems, including validating MUSes computed with modern MUS extractors and finding an equivalent subformula of minimum size.

* This work is partially supported by SFI PI grant BEACON (09/IN.1/I2618), FCT grant POLARIS (PTDC/EIA-CCO/123051/2010) and national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

This paper shows how recent work on iterative enumeration of hitting sets can be adapted to computing the smallest MUS of unsatisfiable formulae, improving upon state-of-the-art algorithms. The approach is implemented and tested on Boolean Satisfiability instances, but the technique applies equally well to computing SMUSes of any constraint system for which the satisfiability of all subsets is well-defined.

The paper is organized as follows. In Section 2 basic definitions are provided. Section 3 introduces an algorithm for the extraction of an SMUS. In Section 4 some optimizations are presented. Finally, Section 5 is dedicated to the presentation of the experimental results, and Section 6 concludes the paper.

2 Preliminaries

This section provides common definitions that will be used throughout the paper. Additional standard definitions are assumed (e.g. see [3]). In what follows, $\mathcal{F}$ denotes a propositional formula expressed in Conjunctive Normal Form (CNF). A formula in CNF is a conjunction of clauses, where each clause is a disjunction of literals. A literal is either a Boolean variable or its complement. Also, we may refer to formulae as sets of clauses and clauses as sets of literals. An assignment to variables that satisfies formula $\mathcal{F}$ is said to be a model of $\mathcal{F}$. A formula is unsatisfiable when no model exists. Unless specified explicitly, $\mathcal{F}$ is assumed to be unsatisfiable. The following definitions also apply.

Definition 1. *A subset $\mathcal{U} \subseteq \mathcal{F}$ is a minimal unsatisfiable subset (MUS) if $\mathcal{U}$ is unsatisfiable and $\forall \mathcal{U}' \subset \mathcal{U}$, $\mathcal{U}'$ is satisfiable. An MUS $\mathcal{U}$ of $\mathcal{F}$ of the smallest size is called a smallest MUS (SMUS).*

Definition 2. *A subset $\mathcal{C}$ of $\mathcal{F}$ is a minimal correction subset (MCS) if $\mathcal{F} \setminus \mathcal{C}$ is satisfiable and $\forall \mathcal{C}' \subseteq \mathcal{C} \wedge \mathcal{C}' \neq \emptyset$, $(\mathcal{F} \setminus \mathcal{C}) \cup \mathcal{C}'$ is unsatisfiable.*

Definition 3. *A satisfiable subset $\mathcal{S} \subseteq \mathcal{F}$ is a maximal satisfiable subset (MSS) if $\forall \mathcal{S}' \subseteq \mathcal{F}$ s.t. $\mathcal{S} \subset \mathcal{S}'$, $\mathcal{S}'$ is unsatisfiable.*

An MSS can also be defined as the complement of an MCS (and vice versa). If $\mathcal{C}$ is an MCS, then $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ represents an MSS. On the other hand, MUSes and MCSes are related by the concept of *minimal hitting set*.

Definition 4. *Given a collection Γ of sets from a universe $\mathbb{U}$, 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 minimal hitting set duality between MUSes and MCSes is well-known (e.g. see [17,15]):

Proposition 1 *Given a CNF formula $\mathcal{F}$, let MUSes($\mathcal{F}$) and MCSes($\mathcal{F}$) be the set of all MUSes and MCSes of $\mathcal{F}$, respectively. Then the following holds:*

1. *A subset $\mathcal{U}$ of $\mathcal{F}$ is an MUS iff $\mathcal{U}$ is a minimal hitting set of MCSes($\mathcal{F}$).*
2. *A subset $\mathcal{C}$ of $\mathcal{F}$ is an MCS iff $\mathcal{C}$ is a minimal hitting set of MUSes($\mathcal{F}$).*

The duality relating MUSes and MCSes is a key aspect of the algorithm presented below. In the next section, we will describe how the smallest MUS can be computed by exploiting this observation.

Algorithm 1: Basic SMUS algorithm

Input: CNF formula $\mathcal{F}$

```
1  begin
2      H ← ∅
3      while true do
4          h ← MinimumHS(H)
5          F' ← {c_i | e_i ∈ h}
6          if not SAT(F') then
7              return SMUS ← F'
8          else
9              C ← grow(F')
10         H ← H ∪ {C}
11 end
```

3 Basic Algorithm

This section describes the new SMUS algorithm. We start by providing the definition of a *minimum size hitting set*:

Definition 5. *Let Γ be a collection of sets and MHS(Γ) the set of all minimal hitting sets on Γ. Then a hitting set $h \in MHS(\Gamma)$ is said to be a* ***minimum*** *hitting set if $\forall\ h' \in MHS(\Gamma)$ we have that $|h| \leq |h'|$.*

The algorithm is based on the following observation:

Proposition 2 *A set $\mathcal{U} \subseteq \mathcal{F}$ is an SMUS of $\mathcal{F}$ if and only if $\mathcal{U}$ is a minimum hitting set of MCSes($\mathcal{F}$).*

By duality [17], we also have that the minimum hitting set of MUSes($\mathcal{F}$) corresponds to a MaxSAT solution for $\mathcal{F}$. This observation has already been exploited in [6]. Algorithm 1 can be seen as the dual of the algorithm presented in [6]. $\mathcal{H}$ represents a collection of sets, where each set corresponds to an MCS on $\mathcal{F}$. Thus, elements of the sets in $\mathcal{H}$ represent clauses of $\mathcal{F}$. Let $elm(\mathcal{H})$ denote the set of elements in $\mathcal{H}$. Each element $e \in elm(\mathcal{H})$ is associated with a clause $c \in \mathcal{F}$. At the beginning of the algorithm $\mathcal{H}$ is empty (line 2). At each step, a minimum hitting set h is computed on $\mathcal{H}$ (see line 4) and the induced formula $\mathcal{F}'$ is tested for satisfiability at line 6. If $\mathcal{F}'$ is satisfiable, it is extended by the *grow* procedure into an MSS containing $\mathcal{F}'$, the complement of which is returned as the MCS $\mathcal{C}$. Then $\mathcal{C}$ is added to the collection $\mathcal{H}$. If instead $\mathcal{F}'$ is unsatisfiable then $\mathcal{F}'$ is guaranteed to be an SMUS of $\mathcal{F}$ as the following lemma states:

Lemma 1. *Let $\mathcal{K} \subseteq$ MCSes($\mathcal{F}$). Then a subset $\mathcal{U}$ of $\mathcal{F}$ is an SMUS if $\mathcal{U}$ is a minimum hitting set on $\mathcal{K}$ and $\mathcal{U}$ is unsatisfiable.*

Proof. Since $\mathcal{U}$ is unsatisfiable it means that it already hits every MCS in MCSes($\mathcal{F}$) (Proposition 1). $\mathcal{U}$ is also a minimum hitting set on MCSes($\mathcal{F}$), since

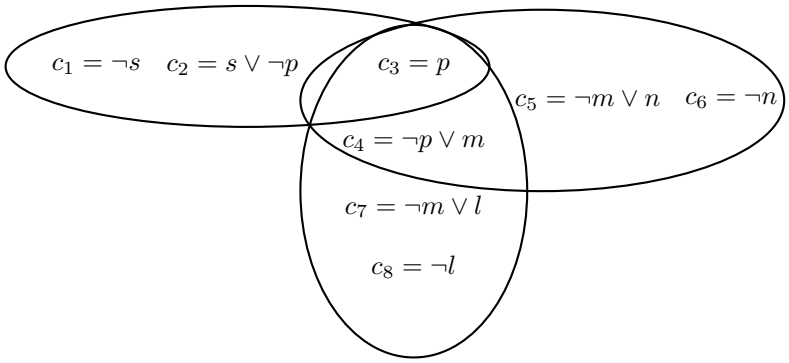


Fig. 1: Formula example

MinimumHS($\mathcal{H}$)	SAT($\mathcal{F}'$)	$\mathcal{F} \setminus grow(\mathcal{F}', \mathcal{F})$	$\mathcal{H} = \mathcal{H} \cup \mathcal{C}$
$\mathcal{F}' \leftarrow \{\emptyset\}$	true	$\mathcal{C} \leftarrow \{c_3\}$	$\{\{c_3\}\}$
$\mathcal{F}' \leftarrow \{c_3\}$	true	$\mathcal{C} \leftarrow \{c_2, c_4\}$	$\{\{c_3\}, \{c_2, c_4\}\}$
$\mathcal{F}' \leftarrow \{c_2, c_3\}$	true	$\mathcal{C} \leftarrow \{c_1, c_4\}$	$\{\{c_3\}, \{c_2, c_4\}, \{c_1, c_4\}\}$
$\mathcal{F}' \leftarrow \{c_3, c_4\}$	true	$\mathcal{C} \leftarrow \{c_1, c_5, c_7\}$	$\{\{c_3\}, \{c_2, c_4\}, \{c_1, c_4\}, \{c_1, c_5, c_7\}\}$
$\mathcal{F}' \leftarrow \{c_1, c_2, c_3\}$	false	$\{c_1, c_2, c_3\}$ is an SMUS of $\mathcal{F}$	

Table 1: Example SMUS computation

it is a minimum hitting set for $\mathcal{K} \subseteq \text{MCSes}(\mathcal{F})$ and no other added MCS can make it grow in size. Moreover, all the other hitting sets can either grow in size or remain the same. Thus, by Proposition 2 $\mathcal{U}$ must be an SMUS. □

Lemma 1 states that it is not necessary to enumerate all MCSes of formula $\mathcal{F}$. Instead, it is enough to compute only those whose minimum hitting set is an MUS. Therefore, Algorithm 1 terminates once the first MUS is computed, which is by construction guaranteed to be of the smallest size.

It is worth noting that nothing in Algorithm 1 is specific to Boolean CNF, and in fact the algorithm can be applied to any type of constraint system for which the satisfiability of constraint subsets can be checked. The algorithm and all following additions to it are constraint agnostic.

An example of a run of Algorithm 1 is shown in Table 1 (Figure 1 illustrates the input formula). The first column contains the formula $\mathcal{F}'$ induced by the minimum hitting set on $\mathcal{H}$. Whenever $\mathcal{F}'$ is satisfiable (second column), the *grow* procedure (see line 9 of Algorithm 1) returns an MCS (third column). The last

column shows the current $\mathcal{H}$, the collected set of MCSes from which a minimum hitting set will be found in the next iteration. In the last row, $\mathcal{F}' = \{c_1, c_2, c_3\}$ and since the call $\texttt{SAT}(\mathcal{F}')$ returns false, set $\{c_1, c_2, c_3\}$ represents an SMUS. Notice that $\texttt{MinimumHS}(\mathcal{H})$ could have returned $\{c_3, c_4, c_5\}$ instead of $\{c_1, c_2, c_3\}$. In this case, further iterations would be necessary in order to find an SMUS.

4 Additional Details

This section describes some essential details of the approach being proposed. These include computing a minimum hitting set with at most one SAT call, enumerating disjoint MCSes, and reporting an upper bound within a reasonably short time.

4.1 Reducing the Number of SAT Calls

A number of different approaches can be used for computing minimum size hitting sets [6]. This section describes a different alternative that exploits the use of SAT solvers as well as the problem structure. The proposed approach exploits incremental MaxSAT solving [5].

Let the current minimum hitting set (MHS) h have size k, and let $\mathcal{C}$ be a new set to hit. Call a SAT solver on the resulting formula, requesting an MHS of size k. If the formula is satisfiable, then we have a new MHS of size k that also hits $\mathcal{C}$. However, if the formula is unsatisfiable, this means there is no MHS of size k. Thus, an MHS of size $k+1$ can be constructed by adding to h *any* element of $\mathcal{C}$. Clearly, by construction, h does not hit any element of $\mathcal{C}$. Thus, every MHS can be obtained with a *single* call to a SAT solver.

Additionally, extraction of each MCS $\mathcal{C}$ (see line 9) requires a number of SAT calls. Alternatively, one can avoid the computation of MCS $\mathcal{C}$ and use any correction subset $\mathcal{C}'$ s.t. $\mathcal{C} \subseteq \mathcal{C}'$ instead. Indeed, any MHS of the set of all correction subsets of $\mathcal{F}$ also hits every MCS of $\mathcal{F}$ [17]. Moreover, observe that a proposition similar to Lemma 1 can be proved for a partial set of correction subsets of $\mathcal{F}$. Therefore, the *grow* procedure can be skipped, and the complement to $\mathcal{F}'$ can be directly added to $\mathcal{H}$.

4.2 Disjoint MCS Enumeration

Enumeration of disjoint inconsistencies has been used in various settings in recent years. For example, disjoint unsatisfiable core enumeration in MaxSAT was proposed in [16], and nowadays it is often used in state-of-the-art MaxSAT solvers (e.g. see [7]). Enumeration of disjoint MCSes, which act as unsatisfiable cores over quantified constraints, has also proven its relevance for the SMUS problem in the context of QMaxSAT [9]. Also note that disjoint MCSes are known to have a strong impact on the performance of the branch-and-bound algorithms for SMUS by refining the lower bound on the size of the SMUS [13].

Our approach based on the minimum hitting set duality [17,15] between MCSes and MUSes of an unsatisfiable CNF formula $\mathcal{F}$ can also make use of disjoint MCSes. Indeed, since each MCS must be hit by any MUS of a CNF formula, disjoint MCSes must be hit by the smallest MUS of the formula. Therefore, given a set of disjoint MCSes $\mathcal{D}$, one can initialize set $\mathcal{H}$ with $\mathcal{D}$ instead of the empty set (see line 2 of Algorithm 1) in order to boost the performance of Algorithm 1. Note that this also simplifies the computation of the first minimum hitting set, which for a set of disjoint MCSes of size k is exactly of size k. According to the experimental results described in Section 5, this improvement has a huge impact on the overall performance of the proposed approach.

4.3 Finding Approximate Solutions

Although the approach being proposed performs better than the known alternative approaches to SMUS, this problem is computationally much harder than extracting any MUS of a CNF formula (the decision version of the SMUS problem is known to be Σ_2^{P}-complete, e.g. see [12,8]). One may find this complexity characterization a serious obstacle for using SMUS algorithms in practice. Indeed, a user may prefer to find an MUS close to the smallest size within a reasonable amount of time instead of waiting until the smallest MUS is found.

In order to resolve this issue and following the ideas of [10], Algorithm 1 can be extended for computing a *"good"* upper bound on the exact solution of the SMUS problem. Any MUS can be considered as an upper bound on the smallest MUS. Therefore and since extracting one MUS is a relatively simple task, enumerating MUSes within a given time limit and choosing the smallest one among them can be satisfactory for a user. MUS enumeration can be done in a way similar to the one proposed in [14]. Observe that Algorithm 1 iteratively refines a lower bound of the SMUS, and so can serve to measure the quality of approximate upper bounds.

This pragmatic policy of computing an upper bound before computing the exact solution provides a user with a temporary approximate answer within a short period of time and continues with computing the exact solution, if needed (i.e. if the user is not satisfied with the quality of the upper bound). Otherwise, the upper bound is enough and the user can stop the process.

5 Experimental Results

This section evaluates the approach to the smallest MUS problem proposed in this paper. The experiments were performed in Ubuntu Linux on an Intel Xeon E5-2630 2.60GHz processor with 64GByte of memory. The time limit was set to 800s and the memory limit to 10GByte. The approach proposed above was implemented in a prototype called FORQES (*FORmula QuintESsence extractor*). The underlying SAT solver of FORQES is Glucose 3.0[1] [1]. A weakened version

[1] Available from http://www.labri.fr/perso/lsimon/glucose/.

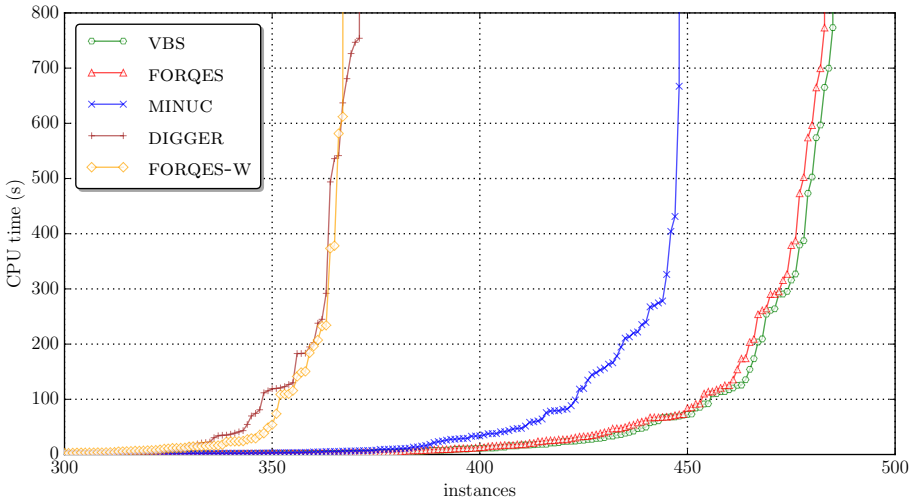


Fig. 2: Cactus plot showing the performance of FORQES, MINUC, and DIGGER

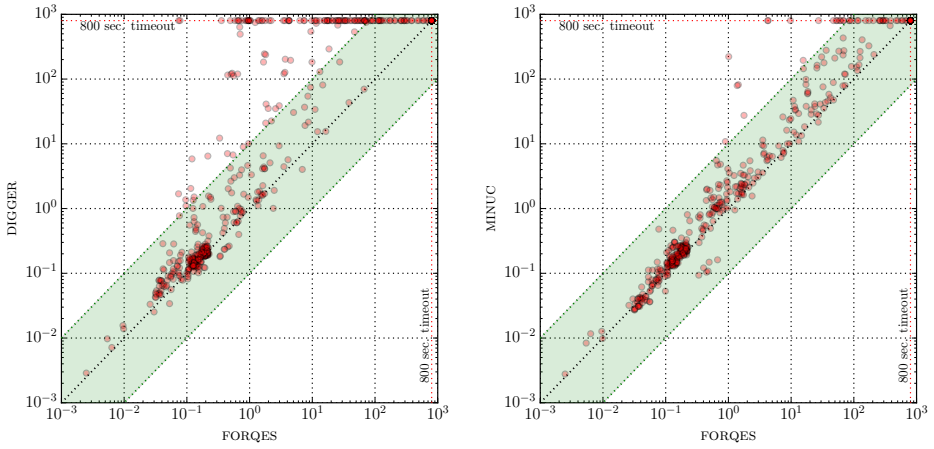
of FORQES, which does not use disjoint MCS enumeration and implements Algorithm 1, is referred to as FORQES-W.

The performance of FORQES was compared to the state-of-the-art approach to the SMUS problem that uses the quantified MaxSAT formulation of SMUS [9]. The most efficient version of the tool described in [9] performs core-guided QMaxSAT solving; in the following it is referred to as MINUC. Additionally, a well-known branch-and-bound SMUS extractor called DIGGER (see [13]) was also considered. Note that the versions of MINUC and DIGGER participating in the evaluation make use of disjoint MCS enumeration.

Several sets of benchmarks were used to assess the efficiency of the new algorithm, all used in the evaluation in [9] as well. This includes a collection of instances from automotive product configuration benchmarks [18] and two sets of circuit diagnosis instances. Additionally, we selected instances from the complete set of the MUS competitions benchmarks[2] as follows. Because extracting a smallest MUS of a CNF formula is computationally much harder than extracting any MUS of the formula, the instances that are difficult for a state-of-the-art MUS extractor were excluded. Instead, we considered only formulae solvable by the known MUS extractor MUSER-2 (e.g. see [2]) within 10 seconds. The total number of instances considered in the evaluation is 682.

Figure 2 shows a cactus plot illustrating the performance of the tested solvers on the total set of instances. FORQES exhibits the best performance, being able to solve 483 instances. MINUC comes second with 448 instances solved. Thus, FORQES solves 7.8% more instances than MINUC. DIGGER and FORQES-W have almost the same performance solving 371 and 367 instances, respectively.

[2] See http://www.satcompetition.org/2011/.



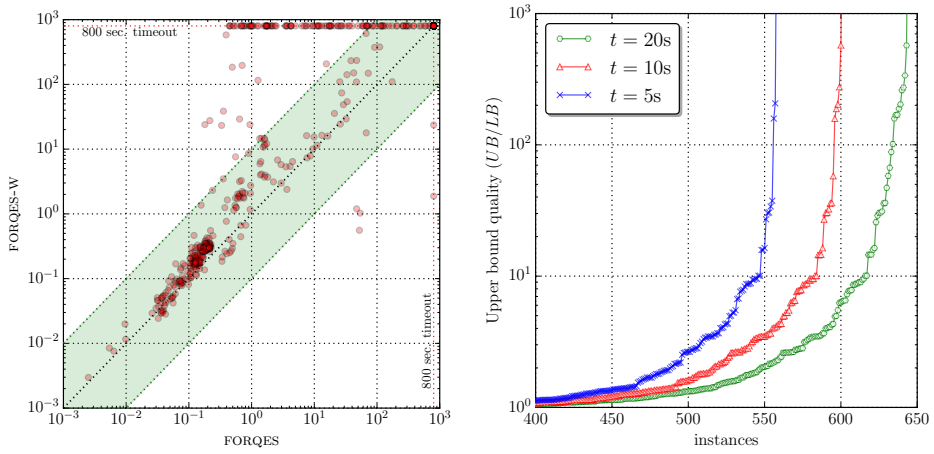
(a) FORQES vs DIGGER (b) FORQES vs MINUC

Fig. 3: Performance of FORQES, MINUC, and DIGGER

More details on the solvers' performance can be found in Figure 3a, Figure 3b, and Figure 4a. Also note that the *virtual best solver* (VBS) among all the considered solvers is able to solve 485 instances, which is only 2 more than what FORQES can solve on its own. Interestingly, neither MINUC nor DIGGER contribute to the VBS. Although the weakened version of FORQES (FORQES-W) performs quite poorly, it is the only solver (besides the normal version of FORQES) that contributes to the VBS. This can be also seen in Figure 4a.

As it was described in Section 4.3, the approach being proposed can report an upper bound, which can be used as an approximation of the SMUS if finding the exact solution is not efficient and requires too much time. The following evaluates the quality of the upper bound reported by this pragmatic solving strategy. Given an upper bound UB computed within some time limit and the optimal value $opt \leq UB$, the closer value $\frac{UB}{opt}$ is to 1 the better the quality of UB is. Since it is often hard to find the exact optimal value, one can consider a lower bound on the exact solution instead of the optimal value in order to estimate the quality of the upper bound. Indeed, any set of disjoint MCSes found within a given timeout can be seen as a lower bound on the exact solution.

Given a CNF formula, an upper bound on its smallest MUS reported by FORQES within a time limit is computed with an external call to a known MUS enumerator called MARCO [14]. Several timeout values were tested, namely 5, 10, and 20 seconds. Figure 4b shows a cactus plot illustrating the quality of the upper bounds computed this way. It is not surprising that generally the more time is given to the solver, the better upper bound is computed. For about 400 instances, value $\frac{UB}{LB}$ is extremely close to 1 meaning that the upper bound is usually almost equal to the lower bound. As one can see, the upper bound is less than an order of magnitude larger than the lower bound for about 550, 585,



(a) FORQES vs FORQES-W (b) UB quality in 5, 10, and 20 sec

Fig. 4: Performance of FORQES-W and UB quality

and 620 instances if computed within 5, 10, and 20 seconds, respectively. Also note that both lower and upper bounds are relatively easy to compute, e.g. both of them can be computed almost for 650 instances (out of 682). Moreover, they give an idea of how large the interval is between them. Given this information, a user can decide how critical it is to compute the exact solution.

In summary, the experimental results indicate that the proposed approach pushes the state of the art in SMUS solving, outperforming all the previous approaches in terms of the number of solved instances. Moreover, the scatter plots indicate that in most of the cases the proposed approach is also the fastest in comparison to others. Moreover, a pragmatic policy to report an upper bound within a short period of time can be helpful when it is hard to compute the exact solution and provide a user with an approximate solution of the SMUS problem of reasonable size.

6 Conclusion

This paper adapts recent algorithms for implicit set problems [11,4,6,19] to the case of computing the smallest MUS. A number of enhancements are developed and added to the new algorithm. Experimental results, obtained on representative problems instances, show clear gains over what currently represents the state of the art. A natural line of research is to apply the novel SMUS algorithm in concrete practical applications of SMUSes, e.g. finding smallest equivalent subformulae of smallest size.

References

1. G. Audemard, J. Lagniez, and L. Simon. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *SAT*, pages 309–317, 2013.
2. A. Belov, I. Lynce, and J. Marques-Silva. Towards efficient MUS extraction. *AI Commun.*, 25(2):97–116, 2012.
3. 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.
4. K. Chandrasekaran, R. M. Karp, E. Moreno-Centeno, and S. Vempala. Algorithms for implicit hitting set problems. In *SODA*, pages 614–629, 2011.
5. A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. A modular approach to MaxSAT modulo theories. In *SAT*, pages 150–165, 2013.
6. J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *CP*, pages 225–239, 2011.
7. J. Davies and F. Bacchus. Exploiting the power of MIP solvers in MAXSAT. In *SAT*, pages 166–181, 2013.
8. A. Gupta. *Learning Abstractions for Model Checking*. PhD thesis, Carnegie Mellon University, June 2006.
9. A. Ignatiev, M. Janota, and J. Marques-Silva. Quantified Maximum Satisfiability: - A core-guided approach. In *SAT*, pages 250–266, 2013.
10. A. Ignatiev, M. Janota, and J. Marques-Silva. Towards efficient optimization in package management systems. In *ICSE*, pages 745–755, 2014.
11. R. M. Karp. Implicit hitting set problems and multi-genome alignment. In *CPM*, page 151, 2010.
12. P. Liberatore. Redundancy in logic I: CNF propositional formulae. *Artif. Intell.*, 163(2):203–232, 2005.
13. M. H. Liffiton, M. N. Mneimneh, I. Lynce, Z. S. Andraus, J. Marques-Silva, and K. A. Sakallah. A branch and bound algorithm for extracting smallest minimal unsatisfiable subformulas. *Constraints*, 14(4):415–442, 2009.
14. M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva. Fast, flexible MUS enumeration. *Constraints*, 2015. http://dx.doi.org/10.1007/s10601-015-9183-0.
15. M. H. Liffiton and K. A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008.
16. J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *DATE*, pages 408–413, 2008.
17. R. Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.
18. C. Sinz, A. Kaiser, and W. Küchlin. Formal methods for the validation of automotive product configuration data. *AI EDAM*, 17(1):75–97, 2003.
19. R. T. Stern, M. Kalech, A. Feldman, and G. M. Provan. Exploring the duality in conflict-directed model-based diagnosis. In *AAAI*, pages 828–834, 2012.