Evaluation of Domain Agnostic Approaches for Enumeration of Minimal Unsatisfiable Subsets

Jaroslav Bendík and Ivana Černá

Faculty of Informatics, Masaryk University, Brno, Czech Republic
{xbendik,cerna}@fi.muni.cz

Abstract

In many different applications we are given a set of constraints with the goal to decide whether the set is satisfiable. If the set is determined to be unsatisfiable, one might be interested in analysing this unsatisfiability. Identification of minimal unsatisfiable subsets (MUSes) is a kind of such analysis. The more MUSes are identified, the better insight into the unsatisfiability is obtained. However, the full enumeration of all MUSes is often intractable. Therefore, algorithms that identify MUSes in an *online* fashion, i.e., one by one, are needed. Moreover, since MUSes find applications in various constraint domains, and new applications still arise, there is a desire for domain agnostic MUS enumeration approaches.

In this paper, we present an experimental evaluation of four state-of-the-art domain agnostic MUS enumeration algorithms: MARCO, TOME, ReMUS, and DAA. The evaluation is conducted in the SAT, SMT, and LTL constraint domains. The results evidence that there is no silver-bullet algorithm that would beat all the others in all the domains.

1 Introduction

In various areas of computer science, such as requirements analysis, model checking, or constraint processing, we are given a set of constraints with the goal to determine whether the set of constraints is satisfiable, i.e. whether all the constraints can hold simultaneously. If the set is determined to be unsatisfiable, it is usually desirable to analyse the unsatisfiability. Identification of the minimal unsatisfiable subsets (MUSes) of the given set of constraints is a kind of such analysis. A set of constraints is a minimal unsatisfiable subset (MUS) if it is unsatisfiable, yet all of its proper subsets are satisfiable.

The problem of MUS identification was extensively studied in the past decades [8, 7, 28, 2, 26, 4, 11, 13]. The existing solutions can be divided into two categories: identification of a *single* MUS, and *enumeration* of all MUSes. In our work we focus on the enumeration problem. Solutions of this problem subsume the task of deciding whether a given concrete set of constraints is satisfiable. Depending on the constrain domain, the satisfiability problem can be even NP-hard and thus the test can be very expensive. On top of that, the number of all MUSes can be exponential w.r.t. size of the constraint set and this makes the enumeration problem generally intractable. Therefore we give preference to *online* algorithms which enumerate MUSes one by one (and can be stopped any-time).

Furthermore, we can classify the existing MUS identification approaches either as *domain specific* or *domain agnostic* (i.e. applicable in any constraint domain). In this work, we focus on the domain agnostic approaches for MUS enumeration.

Domain agnostic MUS enumeration algorithms have shown to be very useful mainly due to three reasons. First, there are still arising new applications where MUSes are searched for. New applications may bring new constraint domains. In such a case, any domain agnostic algorithm can be used almost immediately [22]. Second, the domain agnostic algorithms often

form a basis for developing domain specific algorithms. Finally, contemporary domain agnostic MUS enumeration algorithms often employ black-box domain specific subroutines. This makes the domain agnostic algorithms competitive even to fully domain specific ones [2, 3, 29].

The domain agnostic algorithms were extensively studied in the recent years and several algorithms were proposed [4, 25, 26, 30, 12, 11, 13, 10]. However, the relevant papers usually present experimental evaluation of algorithms only in one constraint domain, typically in the domain of Boolean (SAT) constraints. There is no evidence of how do these algorithms perform in other domains.

In this work, we present an experimental evaluation of four state-of-the-art **domain agnostic online MUS enumeration algorithms**: MARCO, ReMUS, TOME, and DAA. The comparison is conducted in three constraint domains: SAT, SMT, and LTL. Our goal is to examine the behaviour of the four algorithms in situation where the enumeration of all MUSes is intractable (would take too much time). The main comparison criterion is thus the number of identified MUSes within a given time limit.

We selected the three constraint domains because several applications of MUSes arised in these domains recently (below, we provide two example use cases). Also, the three domains vary in the complexity of the satisfiability problem going from relatively effectively solvable NP-complete problem in the SAT domain to PSPACE-complete problem in the LTL domain.

CEGAR Use Case (SAT and SMT) In some model checking techniques, such as the *counterexample-guided abstraction refinement* (CEGAR) [16], we are dealing with the following question: is the counterexample that was found in an abstract model feasible also in the concrete model? To answer this question, a SAT or SMT formula $cex \wedge conc$ encoding both the counterexample *cex* and the concrete model *conc* is built and tested for satisfiability. If the formula is unsatisfiable, then the counterexample is *spurious* and the negation of the formula $cex \wedge conc$ is used to refine the abstract model. Since both *cex* and *conc* are often formed as a conjunction of smaller subformulas, the whole formula can be seen as a set of conjuncts (constraints). Andraus et al. [1, 16] found out that instead of using the negation of $cex \wedge conc$ for the refinement, it is better to identify the MUSes of $cex \wedge conc$ and use the negations of the MUSes to refine the abstract model.

Requirements Analysis Use Case (LTL) In the *requirements analysis*, the constraints represent requirements on a system that is being developed. Checking for satisfiability (also called *consistency*) means checking whether all the requirements can be implemented at once. If the set of requirements is unsatisfiable, the extraction of MUSes helps to identify and fix the conflicts among the requirements [5, 9].

2 Evaluated Algorithms

As far as we know, the first domain agnostic MUS enumeration algorithm was presented by Hou [24] and applied in the field of diagnosis. Hou's algorithm is based on an explicit exploration of every subset of the given constraint set, starting with the whole constraint set and exploring individual branches of its power set. The author also presented some pruning rules to avoid traversing irrelevant branches. Further improvements to Hou's algorithm were presented later by Han and Lee [23] and by de la Banda et al. [17]. A similar solution based on step-by-step traversal of the power set was proposed by Bauch et al. [5]. However, the explicit exploration of the power set is the bottleneck for all of these algorithms since the power set is exponentially large w.r.t. the size of the given set of constraints. Consequently, a symbolic representation of the power set was introduced into the MUS enumeration, launching into substantially more

efficient algorithms. In our study we evaluate four contemporary algorithms that are based on symbolic representation of constraint sets: MARCO, TOME, ReMUS, and DAA.

MARCO Liffiton et al. [25] and Silva et al. [30] presented independently two nearly identical algorithms for MUS enumeration called MARCO [25] and eMUS [30], respectively. Both algorithms were later merged into a single algorithm and presented under the name MARCO [26]. To avoid explicit exploration of all subsets of the given set C of constraints, MARCO maintains a Boolean formula *map* to represent the *unexplored* subsets of C, i.e. the subsets whose satisfiability is not known yet. Each model of *map* corresponds to a single unexplored subset. To find individual MUSes, MARCO iteratively picks a *maximal* unexplored subset and checks it for satisfiability. The unsatisfiable subset is then *shrunk* (reduced) to a MUS. The shrinking is performed as a black-box operation and can be implemented using any single MUS extraction algorithm; this allows MARCO to be domain agnostic and yet indirectly exploit domain specific properties of the particular constraint domains.

TOME Bendík et al. [11] proposed an algorithm called TOME which also uses a symbolic representation of unexplored subsets and employs a black-box shrinking procedure. The most expensive operation of the algorithm MARCO is the shrinking procedure. TOME tries to optimise the price of this operation by shrinking small subsets. TOME builds a chain $B \subset \ldots \subset T$ of unexplored subsets that starts with a minimal unexplored subset B and ends with a maximal unexplored subset T. If B is satisfiable and T is unsatisfiable, then the chain has to contain a *local MUS* (local w.r.t. the chain). Using binary search, TOME finds the local MUS and subsequently shrinks it to a global MUS. The motivation behind searching for local MUSes is to find unsatisfiable subsets that are relatively close to global MUSes and thus are easy to be shrunk. Moreover, TOME tries to predict the complexity of performing individual shrinking and only those shrinks that seem to be cheap to perform are actually performed.

ReMUS Recursive algorithm ReMUS [12] also employs black-box shrinking procedures and uses a symbolic representation of unexplored subsets. Similarly as TOME, ReMUS exploits the observation that the larger set is being shrunk the harder is to shrink it. Thus, ReMUS tends to find relatively small unsatisfiable unexplored subsets (*seeds*) for shrinking. In order to do so, ReMUS recursively searches for seeds in smaller and smaller subsets of the given set of constraints. In particular, the initial seed is found among the maximal unexplored subsets of the original set C of constraints. Once a seed S is found, ReMUS shrinks it to a MUS M. To find another MUS, the algorithm picks some R such that $M \subset R \subset S$, and recursively searches for a seed among the maximal unexplored subsets of R.

DAA Bailey and Stuckey [4] proposed an algorithm called DAA which explores the power set in a symbolic way but does not use any shrinking procedure. Instead, the algorithm is based on the relationship between MUSes and the so-called minimal correction sets (MCSes). The relationship states that every $U \subseteq C$ is a MUS of C if and only if U is a minimal hitting set of the set of all MCSes of C. To obtain individual MUSes, DAA in each iteration computes a minimal hitting set H of already known MCSes, and tests H for satisfiability. If H is unsatisfiable, then it is guaranteed to be a MUS of C. Otherwise, if H is satisfiable, it is *grown* to a maximal satisfiable subset S, whose complement $(C \setminus S)$ is a MCS of C and thus the set of already known MCSes is enlarged. The growing can be performed using any single MCS extraction algorithm which allows DAA to indirectly exploit domain specific properties.

There are several other MUS enumeration algorithms [27, 2, 29, 22, 10, 3, 31, 32]. However, the algorithms are either domain specific or do not enumerate MUSes in an online fashion. Thus we do not include them in our evaluation.

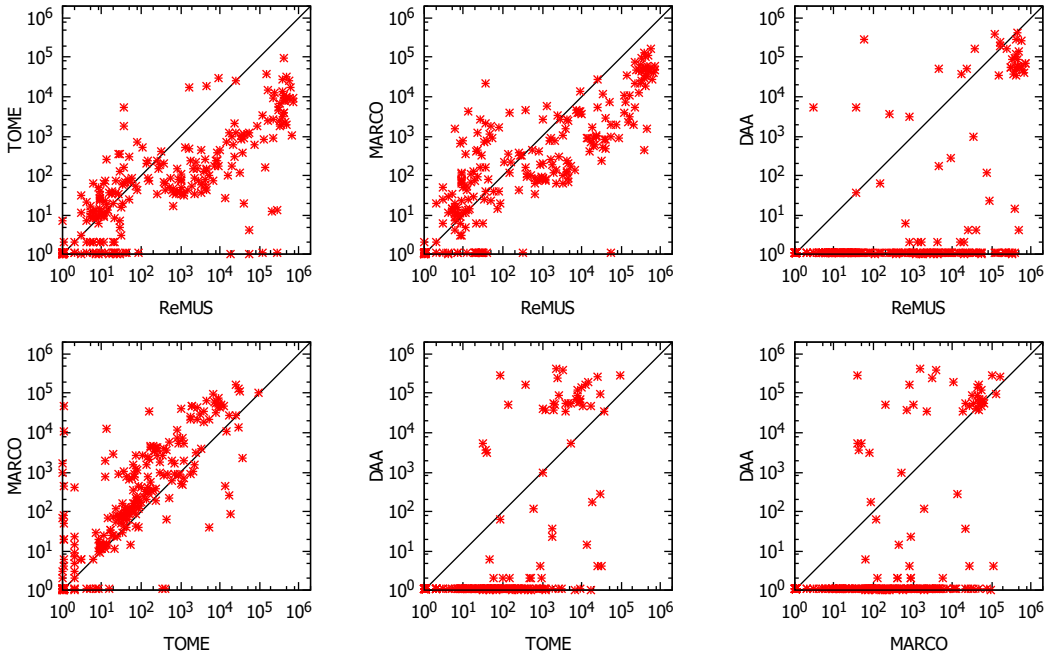

Figure 1: Scatter plots comparing the number of identified MUSes in the SAT domain.

3 Experimental Evaluation

We compare four domain agnostic MUS enumeration algorithms: MARCO, ReMUS, TOME, and DAA. The comparison is carried out in three constraint domains: the domain of Boolean constraints (SAT domain), the domain of Satisfiability Modulo Theories (SMT domain), and the domain of Linear Temporal Logic (LTL domain). Recall that we focus on comparing algorithms that enumerate MUSes *online*, i.e. one by one, and are suitable for benchmarks where the complete MUS enumeration is within a given time limit *intractable*. We focus in our comparison only on the *intractable benchmarks*. The comparative criterion is the number of found MUSes within a given time limit.

All experiments were run using a time limit of 3600 seconds and computed on an Intel(R) Xeon (R) CPU E5-2630 v2, 2.60GHz, 125 GB memory machine running Arch Linux 4.9.40-l-lts. Complete results are available at

https://www.fi.muni.cz/~xbendik/domain-agnostic-evaluation/

3.1 SAT Domain

Benchmarks As experimental data in the SAT domain, we use a collection of 291 Boolean formulae in conjunctive normal form that comes from the MUS track of the SAT 2011 competition[1]. These benchmarks are used in several recent papers that focus on MUS enumeration [26, 25, 11, 12] as well. The benchmarks range in their size from 70 to 16 million constraints

[1] http://www.cril.univ-artois.fr/SAT11

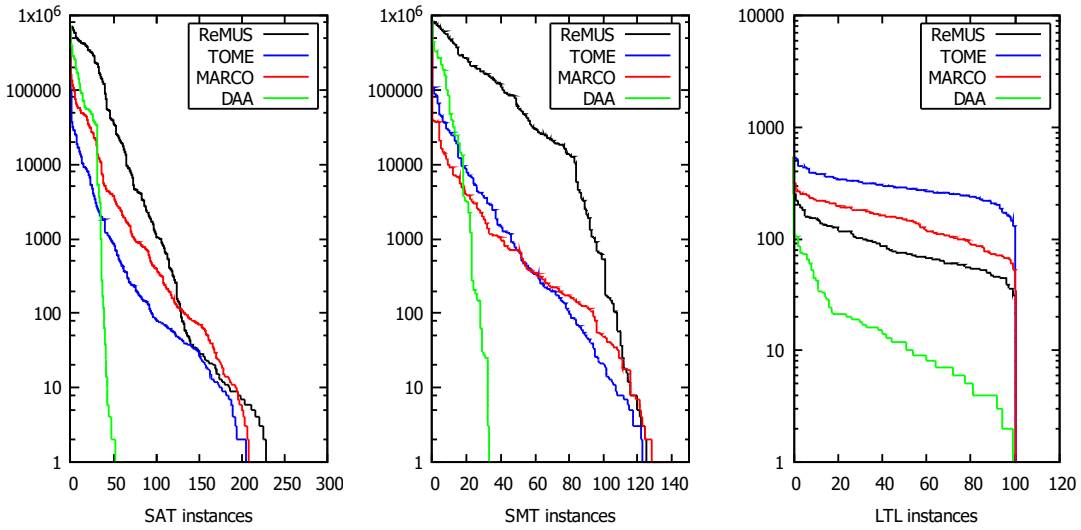

Figure 2: Cumulative plots.

and uses from 26 to 4.4 million variables. Only in case of 23 benchmarks all the evaluated algorithms completed the enumeration. The remaining 268 benchmarks were intractable for at least one of the evaluated algorithms, and thus, these are the benchmarks that we discuss in the evaluation.

Implementation For evaluating MARCO, we used the original implementation[2] by Liffiton and Zhao. Similarly, we use the original implementation[3] of ReMUS and TOME . DAA was originally implemented in a different constraint domain, so we reimplemented it[3]. All the implementations use the same external tools: MUSer2 [8] for shrinking, miniSAT [21] as a SAT solver, and miniSAT [21] for maintaining a symbolic representation of the search space.

Results To measure the efficiency of individual algorithms for the intractable benchmarks we provide scatter plots that pair-wise compare individual algorithms, see Figure 1. Each point in the plot represents the result achieved by the two compared algorithms on one particular benchmark; one algorithm determines the position on the vertical axis and the other one the position on the horizontal axis. Note, that the plots are in a log scale. Since a plot in a log scale cannot show the "0" coordinate, we moved such points to the "1" coordinate, i.e. the points on edges of the plots correspond to benchmarks where an algorithm found either one MUS or none.

A cumulative view on all algorithms provides Figure 2, part SAT. A point with coordinates $[x, y]$ should be read as "for x benchmarks the algorithm computes as least y MUSes". From this we can conlude that ReMUS has the dominant position. Performing a further analysis of the experimental data[3], we can tell that ReMUS found strictly more MUSes than all the other algorithms in 140 intractable benchmarks. MARCO, TOME, and DAA found strictly more MUSes than all the other algorithms in 67, 13, and 12 intractable benchmarks, respectively.

[2] https://sun.iwu.edu/~mliffito/marco/marco_py-2.0.1.tar.gz
[3] https://www.fi.muni.cz/~xbendik/domain-agnostic-evaluation/

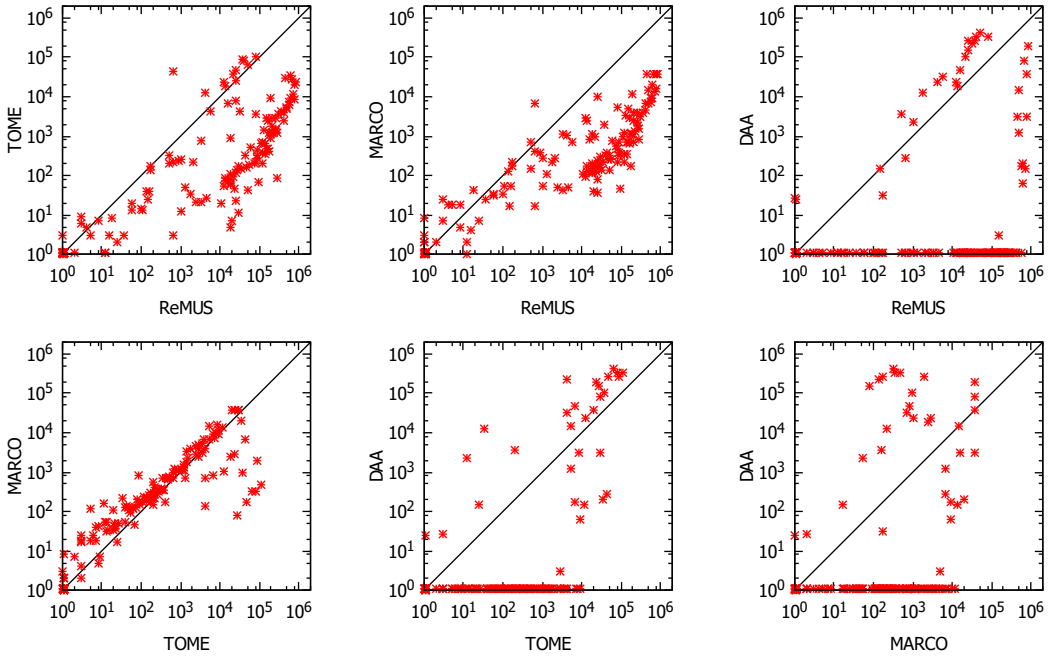

Figure 3: Scatter plots comparing the number of identified MUSes in the SMT domain.

In case of 36 intractable benchmarks, there was no strict winner. Note that although DAA performed the best in several instances, it was often the case that it found no MUS at all.

3.2 SMT Domain

Benchmarks In the SMT domain, we conducted the comparison on a collection of 433 benchmarks from the QF UF, QF IDL, QF RDL, QF LIA and QF LRA divisions of the library SMT-LIB[4]. These benchmarks range in their size from 5 to 145422 constraints and were already used for example in the work by Griggio et al. [15]. Contrary to the SAT domain, a lot of the benchmarks were tractable for complete MUS enumeration. In particular, in case of 238 benchmarks all algorithms completed the enumeration. The remaining 195 benchmarks were intractable for at least one of the algorithms, and thus are objects of our evaluation.

Implementation As in the case of the SAT domain, we used our implementation[3] of DAA, and the original implementations of MARCO[2], ReMUS[3], and TOME[3]. All implementations use Z3 [18] as the satisfiability solver, miniSAT [21] for maintaining a symbolic representation of the search space, and a custom (but the same) implementation of the shrinking procedure.

Results Figure 3 offers scatter plots that compare individual algorithms on individual benchmarks and Figure 2, part SMT, offers the cumulative view. Same as in the SAT domain, ReMUS is the dominating algorithm. In particular, ReMUS found strictly more MUSes than all the other algorithms for 94 intractable benchmarks. DAA, MARCO, and TOME were strictly better than all the other algorithms for 18, 11, and 3 benchmarks, respectively.

[4] http://www.smt-lib.org/

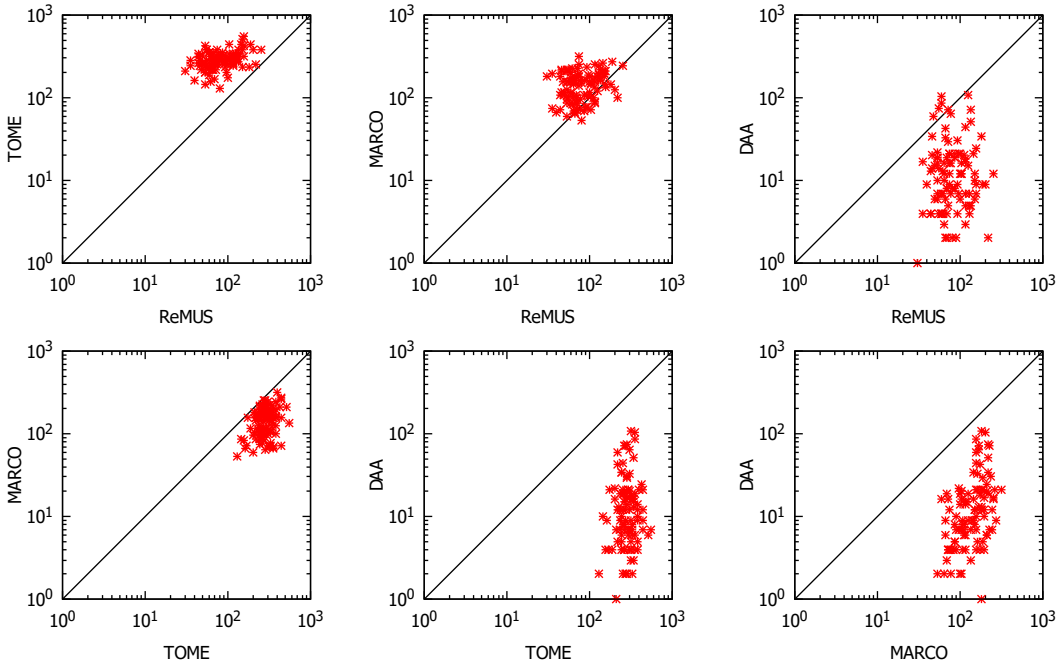

Figure 4: Scatter plots comparing the number of identified MUSes in the LTL domain.

3.3 LTL Domain

Benchmarks Since the applications of MUSes in the LTL domain have begun to be studied quite recently [6, 5, 9], there is no publicly available database of industrial benchmarks (or at least, we are not aware of it). Therefore, to obtain experimental data, we followed the approach of Bauch et al. [5] and generated a collection of random benchmarks using the randltl tool from the SPOT library [19]. According to statistics about the most common industrial LTL formulas [20], the depth of a syntactic tree of an LTL formula is rarely higher than 5. Therefore we generated formulas where the depth is at most 5. In total, we generated 100 benchmarks (sets of formulas) that use up to 15 variables (atomic propositions) and range in their size from 150 to 250 formulas (constraints). Note, that compared to the SAT and SMT domain, the benchmarks are relatively small in their size. This is caused by the complexity of the satisfiability problem which is very high in the LTL domain. It would be intractable to work with larger benchmarks.

Implementation The original implementation[2] of MARCO supports only the SAT and SMT domains, thus we reimplemented it[3]. We compared our reimplementation of MARCO to the original implementation in the SAT and SMT domain and both implementations performed the same, thus the use of our implementation in the LTL domain does not handicap MARCO in comparison with other algorithms. We used the original implementations of ReMUS[3] and TOME[3], and our reimplementation of DAA[3]. All implementations use nuXmv [14] as the satisfiability solver, miniSAT [21] for maintaining a symbolic representation of the search space, and a custom (but the same) implementation of the shrinking procedure.

Results In all of the benchmarks each algorithm found at least one MUS and all the benchmarks were intractable for the complete MUS enumeration. The scatter plots in Figure 4

together with the cumulative view in Figure 2, part LTL, indicate that TOME performed conclusively the best. In fact, it found strictly more MUSes than its competitors in all the benchmarks. MARCO was better than ReMUS finding on average two times more MUSes. DAA was again conclusively the worst, yet it performed much better than in the other two domains where it often found no MUS at all.

4 Recommendations

We have evaluated four different domain agnostic algorithms in three different constraint domains and the results show that there is no silver bullet algorithm that would beat all the others in all the domains. Here, we point out characteristics of the constraint domains and benchmarks that affect the performance of the evaluated algorithms.

The MARCO [26] algorithm is based on shrinking, i.e. using a black-box domain specific single MUS extraction algorithm as a subroutine. Therefore, it is very efficient in constraint domains where efficient single MUS extractors exist. This is in particular the case of the SAT domain, where the conjunctive normal form of formulas allows to significantly reduce the number of performed satisfiability checks during the shrinking [8, 28]. However, MARCO shrinks *maximal unexplored subsets* which are relatively large and very expensive to be shrunk if no efficient shrinking algorithm is available.

ReMUS [12] is also based on shrinking, however it tends to shrink relatively small unsatisfiable subsets (seeds) by recursively reducing the search space in which are the seeds for the shrinking searched for. The reduction is based on previously found MUSes and in order to perform deep recursive calls, the input instance has to contain many similar MUSes [12]. Moreover, the larger the number of constraints in the input constraint set is, the more significant reduction is possible. The SAT and SMT benchmarks in our evaluation contained thousands or millions of constraints and also contained thousands of MUSes; thus the recursion was able to fully manifest. On the other hand, the LTL benchmarks were relatively small and contained much fewer MUSes.

TOME [11] also employs shrinking procedure and, similarly as ReMUS, tends to find relatively small unsatisfiable subsets (seeds) for shrinking. However, instead of recursively reducing the dimension of the search space, it uses binary search to find small seeds. The binary search does not require the input constraint set to contain many MUSes and also does not require the constraint set to be large. TOME shall be very efficient in constraint domains for which no efficient shrinking procedure exists. On the other hand, in domains where efficient shrinking procedures exist, the effort needed to find local MUSes might outweigh the effort needed to perform the shrinking.

DAA [4] is the oldest of the evaluated algorithms and has been already shown before [26] to be inefficient in the SAT domain. Our evaluation has shown that it is also very inefficient in the other domains. Yet, in case of some benchmarks, DAA outperformed all its competitors which means that it is at the end suitable for some kind of benchmarks. DAA exploits the duality [4] between MUSes and the minimal correction sets (MCSes) that allows to extract MUSes by first computing the MCSes. The more MCSes is already computed, the more MUSes can be extracted. We conclude that DAA is efficient in the case of benchmarks that contain relatively small number of MCSes which allows fast extraction of MUSes. However, it might not be easy to predict how many MCSes are contained in a benchmark. Moreover, each benchmark might contain up to exponentially many MCSes which is the reason why DAA in our experiments often found no MUS at all.

References

[1] Zaher S Andraus, Mark H Liffiton, and Karem A Sakallah. Cegar-based formal hardware verification: A case study. *Ann Arbor*, 1001:48109–2122, 2007.

[2] Fahiem Bacchus and George Katsirelos. Using minimal correction sets to more efficiently compute minimal unsatisfiable sets. In *CAV (2)*, volume 9207 of *Lecture Notes in Computer Science*, pages 70–86. Springer, 2015.

[3] Fahiem Bacchus and George Katsirelos. Finding a collection of muses incrementally. In *CPAIOR*, volume 9676 of *Lecture Notes in Computer Science*, pages 35–44. Springer, 2016.

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

[5] Jiří Barnat, Petr Bauch, Nikola Beneš, Luboš Brim, Jan Beran, and Tomáš Kratochvíla. Analysing sanity of requirements for avionics systems. *Formal Aspects of Computing*, 2016.

[6] Jiri Barnat, Petr Bauch, and Lubos Brim. Checking sanity of software requirements. In *SEFM 2012 Proceedings*, volume 7504 of *LNCS*, pages 48–62. Springer, 2012.

[7] Anton Belov, Marijn Heule, and João Marques-Silva. MUS extraction using clausal proofs. In *SAT*, volume 8561 of *Lecture Notes in Computer Science*, pages 48–57. Springer, 2014.

[8] Anton Belov and Joao Marques-Silva. Muser2: An efficient mus extractor. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:123–128, 2012.

[9] Jaroslav Bendík. Consistency checking in requirements analysis. In *ISSTA*, pages 408–411. ACM, 2017.

[10] Jaroslav Bendík, Nikola Beneš, Jiří Barnat, and Ivana Černá. Finding boundary elements in ordered sets with application to safety and requirements analysis. In *SEFM*, volume 9763 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 2016.

[11] Jaroslav Bendík, Nikola Benes, Ivana Cerná, and Jiri Barnat. Tunable online MUS/MSS enumeration. In *FSTTCS*, volume 65 of *LIPIcs*, pages 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[12] Jaroslav Bendík and Ivana Černá. International symposium on automated technology for verification and analysis. 2018. To appear.

[13] Jaroslav Bendík, Elaheh Ghassabani, Michael W. Whalen, and Ivana Cerná. Online enumeration of all minimal inductive validity cores. In *SEFM*, volume 10886 of *Lecture Notes in Computer Science*, pages 189–204. Springer, 2018.

[14] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 334–342. Springer, 2014.

[15] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Computing small unsatisfiable cores in satisfiability modulo theories. *J. Artif. Intell. Res.*, 40:701–728, 2011.

[16] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[17] Maria Garcia de la Banda, Peter J. Stuckey, and Jeremy Wazny. Finding all minimal unsatisfiable subsets. In *Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming*, pages 32–43. ACM, 2003.

[18] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[19] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - A framework for LTL and \omega -automata manipulation. In *ATVA*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129, 2016.

[20] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *FMSP*, pages 7–15. ACM, 1998.

[21] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[22] Elaheh Ghassabani, Michael W. Whalen, and Andrew Gacek. Efficient generation of all minimal inductive validity cores. In *FMCAD*, pages 31–38. IEEE, 2017.

[23] Benjamin Han and Shie-Jue Lee. Deriving minimal conflict sets by cs-trees with mark set in diagnosis from first principles. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 29(2):281–286, 1999.

[24] Aimin Hou. A theory of measurement in diagnosis from first principles. *Artif. Intell.*, 65(2):281–328, 1994.

[25] Mark H. Liffiton and Ammar Malik. Enumerating infeasibility: Finding multiple muses quickly. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings*, volume 7874 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2013.

[26] Mark H. Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva. Fast, flexible MUS enumeration. *Constraints*, pages 1–28, 2015.

[27] Mark H. Liffiton and Karem A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1):1–33, 2008.

[28] Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Accelerated deletion-based extraction of minimal unsatisfiable cores. *JSAT*, 9:27–51, 2014.

[29] Nina Narodytska, Nikolaj Bjørner, Maria-Cristina Marinescu, and Mooly Sagiv. Core-guided minimal correction set and core enumeration. In *IJCAI*, pages 1353–1361. ijcai.org, 2018.

[30] Alessandro Previti and João Marques-Silva. Partial MUS enumeration. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013.

[31] Alessandro Previti and João Marques-Silva. Partial MUS enumeration. In *AAAI*. AAAI Press, 2013.

[32] Christian Zielke and Michael Kaufmann. A new approach to partial MUS enumeration. In *SAT*, volume 9340 of *Lecture Notes in Computer Science*, pages 387–404. Springer, 2015.