Efficient MUS Extraction with Resolution

Alexander Nadel[1], Vadim Ryvchin[1,2], Ofer Strichman[2]
[1]Intel Corporation, P.O. Box 1659, Haifa 31015 Israel
{alexander.nadel, vadim.ryvchin}@intel.com
[2]Information Systems Engineering, Technion, Israel ofers@ie.technion.ac.il

***Abstract*—We report advances in state-of-the-art algorithms for the problem of Minimal Unsatisfiable Subformula (MUS) extraction. First, we demonstrate how to apply techniques used in the past to speed up resolution-based Group MUS extraction to plain MUS extraction. Second, we show that *model rotation*, presented in the context of *assumption-based* MUS extraction, can also be used with *resolution-based* MUS extraction. Third, we introduce an improvement to rotation, called *eager rotation*. Finally, we propose a new technique for speeding-up resolution-based MUS extraction, called *path strengthening*. We integrated the above techniques into the publicly available resolution-based MUS extractor `HaifaMUC`, which, as a result, now outperforms leading MUS extractors.**

I. Introduction

Given an unsatisfiable formula in Conjunctive Normal Form (CNF), an *Unsatisfiable Subformula* (or *Unsatisfiable Core*; hereafter, *US*) is an unsatisfiable subset of its clauses. A *Minimal Unsatisfiable Subformula (MUS)* is a US such that removal of any of its clauses renders it satisfiable. The problem of finding a MUS is an active area of research [1]–[6].

The basic algorithm used in modern MUS extractors such as `MUSer2` [7] and `HaifaMUC` [3] is as follows. In the initial *approximation stage* the algorithm finds a not-necessarily-minimal US S with one or more invocations of a SAT solver [8], [9]. It then applies the following *deletion-based* iterative process over S's clauses until S becomes a MUS. Each iteration removes a *candidate* clause c from S and invokes a SAT solver. If the resulting formula is satisfiable, c must belong to the MUS, so c is returned to S and marked as *necessary*. Otherwise c is removed from S. In addition, the following two optimizations are commonly applied. First, incremental SAT solving [10], [11] is used across all SAT invocations. Second, when a clause c is found to be not necessary, one can remove from S not only c, but all the clauses (if any) omitted from the new core found by the SAT solver. This latter technique is called *clause set refinement* in [6]. The algorithm we have described up to here was introduced in [12] and improved in [2], while the idea of removing constraints one by one in order to get a minimally infeasible set can be traced back to [13], [14]. See [2] for a more detailed presentation of the algorithm and [1] for an overview of various approaches to MUS extraction.

It was demonstrated in [2] that the approach we have described can be implemented using either a resolution-based or an assumption-based algorithm. The former relies on the resolution proof maintained by the SAT solver for detecting the core at each step, while the latter adds a new *assumption literal* to each clause and detects the core using these assumptions. It was shown in [2] that the resolution-based approach to MUS extraction is faster than the assumption-based approach mainly because of the overhead of maintaining assumption literals.

Various applications require finding a MUS with respect to user-given groups of clauses [2], [15], called *interesting constraints*, while clauses that do not belong to any interesting constraint are called the *remainder*. The resulting problem is called *Group MUS (GMUS)* extraction (or *high-level MUS* extraction). It was shown in [2] that the approach we described for plain MUS extraction can be applied to GMUS extraction as well. Furthermore, it was shown in [3] that the resolution-based approach to GMUS extraction can be improved considerably by directing the search to ignore the interesting constraints and to use the remainder and the necessary clauses instead whenever possible. We call the techniques of [3] *MUS-biased search*.

The first contribution of this paper is in showing that MUS-biased search can be applied to plain MUS extraction. The key observation is that while there are no *remainder* clauses in plain MUS extraction, *necessary* clauses can still be used for MUS-biased search after the approximation stage.

A recent essential enhancement to the plain MUS extraction algorithm we have described is *model rotation* (or, simply, *rotation*) [4], [6], [16]. Rotation was proposed in the context of assumption-based MUS extraction. After implementing rotation, the resulting assumption-based MUS extractor `MUSer2` outperformed the state-of-the-art resolution-based MUS extractor `HaifaMUC`. It is sometimes postulated that rotation gives the assumption-based approach an edge over the resolution-based approach (cf. [5]).

The second contribution of this paper is thus in showing that model rotation can be integrated into the resolution-based approach. The paper's third contribution is an improvement to model rotation, called *eager rotation*, detailed in Sect. II-B.

The fourth contribution of our paper is called *path strengthening*. It is a generalization of a technique proposed in [17] and later called *redundancy removal* in [6] and implemented in `MUSer2` [7]. Redundancy removal adds the literals of $\neg c$ (where c is the candidate clause) as assumptions when checking the satisfiability of $S \setminus c$, because since S is known to be unsatisfiable, then $S \setminus c$ and $(S \setminus c) \wedge \neg c$ are equisatisfiable. Path strengthening, on the other hand, adds as assumptions the literals of $\neg c, \neg c_1, \ldots, \neg c_m$ for some $m \geq 0$, where the sequence of clauses $c, c_1, \ldots, c_m$ constitutes the longest common prefix of all paths in the resolution proof from c to

the empty clause. Further details about path strengthening are provided in Sect. II-C.

We integrated our algorithms into the resolution-based MUS extractor `HaifaMUC`. We show in Sect. III that, as a result, `HaifaMUC` now outperforms the leading MUS extractors `MUSer2` and `Minisatabb` [18]. `Minisatabb` improves `MUSer2` considerably based on the idea of replacing blocks of assumptions with new variables [18].

II. The Algorithms

A. MUS-Biased Search

We will now describe how we adapted optimizations **A-D** of the GMUS-oriented techniques proposed in [3] to plain MUS extraction (we also tried adapting optimizations **E-G** [3], but their impact on plain MUS extraction was negligible). We denote the set of necessary input clauses by M. We call an input clause c *interesting* if it belongs to $S \setminus M$ (i.e., c can still serve as a candidate). A learned clause is marked as *interesting* if it is derived using at least one interesting clause; otherwise it is marked as *necessary*. If an interesting learned clause participates in the proof, then the core includes its interesting roots; this is undesirable since we are trying to minimize the core. Most of our techniques are therefore targeted at biasing the solver towards learning *necessary* rather than *interesting* clauses. This is the reason that we call them, jointly, *MUS-biased search*. An exception is the first optimization below, which is focused on reducing the amount of memory used to store the proof.

A. *Maintain partial resolution proofs*. There is no need to store in the proof any clauses identified as necessary, since the algorithm does not need to work with these clauses explicitly anymore. Hence, we discard from the proof all the clauses that emanate exclusively from M.

B. *Perform selective clause minimization*. Clause minimization [19] is a technique for shrinking conflict clauses. Specifically, if a conflict clause c contains two literals l_1, l_2 such that $l_1 \rightarrow l_2$ because of the rest of the formula, then l_2 can be removed from c. The disadvantage of this technique in our context is that it may reclassify c from 'necessary' to 'interesting', if the implication $l_1 \rightarrow l_2$ depends on an interesting clause. This in turn may increase the size of the core later on as explained above. Hence our optimization does not apply clause minimization if it leads to such a reclassification. In other words we prefer a longer conflict clause if this enables us to maintain its classification as a necessary clause.

C. *Postpone propagation over interesting clauses*. Perform Boolean Constraint Propagation (BCP) on necessary clauses first, with the aim of learning a necessary clause when possible.

D. *Reclassify interesting clauses*. When an interesting clause c becomes necessary, look for any clauses in the resolution derivation that were derived from c that also become necessary (that is, were derived solely from necessary clauses) and reclassify them.

Note that while these optimizations improve GMUS extraction even during the approximation stage owing to the availability of remainder clauses, their impact on plain MUS extraction begins only during the minimization stage, when there are enough necessary clauses (which, like remainder clauses, must be in the proof). Indeed we demonstrate in Sect. III that optimization **B** is not cost-effective before there is a significant number of necessary clauses, which is the reason that we invoke it starting from the 2nd satisfiable iteration.

B. Eager Model Rotation

Model rotation can improve deletion-based MUS extraction by searching for additional clauses that should be marked as necessary *without* an additional SAT call. Suppose, for example, that for an unsatisfiable set S, $S \setminus c$ is satisfiable. Consequently c is marked as necessary. Let h be the satisfying assignment. Note that $h(c) = false$, because otherwise $h(S)$ would be $true$, which contradicts S's unsatisfiability. Now, suppose that an assignment h' that is different than h in only one literal $l \in c$ satisfies all the clauses in S other than exactly one clause $c' \in S$. Hence $h'(S \setminus c') = true$, which means that like c, c' must also be in any unsatisfiable subset of S, and can therefore be marked as necessary as well. Rotation flips the values of each of c's literals one at a time in search of such clauses. When one is found, rotation is called recursively with c'. This algorithm is summarized in Fig. 1(a). We observe that rotation, proposed in the context of assumption-based MUS extraction, can be integrated into our resolution-based algorithm without any changes.

Fig 1(b) shows ERMR (Eager Recursive Model Rotation) – an improvement to rotation that weakens rotation's terminating condition. The reader may benefit from first reading the main algorithm in Fig. 2(a), which calls ERMR. The only difference between ERMR and RMR is that ERMR may call rotation with a clause that is already in M, the reason being that it can lead to additional marked clauses owing to the fact that the call is with a different assignment. Clearly there is a tradeoff between the time saved by detecting more clauses for M and the time dedicated to the search. For example, one may run RMR with more than one satisfying assignment as a starting point, but this will require additional SAT calls to find extra satisfying assignments. ERMR refrains from additional SAT calls. Rather it changes the stopping criterion: instead of stopping when $c \in M$ (line 4 in Fig. 1(a)), it stops when $c \in K$, where K holds the clauses that were discovered in the *current* call from MUS. There are other variations on weakening the terminating condition of rotation in the literature [5], [6]. We leave to future study a detailed comparison of our algorithm to these works.

C. Path Strengthening

Path strengthening relies on the following property, which we call *cut falsifiability* (observed already in [12], [20]). Let S be an unsatisfiable formula, π its resolution proof, and c a candidate clause. Let ρ_c be the subgraph of π containing all the clauses that appear on at least one path from c to the

empty clause $\square$ (including c and $\square$). Then, any model h to $S \setminus \{c\}$ must falsify at least one clause in any vertex cut of ρ_c (since otherwise a satisfiable vertex cut in π would exist). An immediate corollary is that *all* the clauses in *some* path from c to $\square$ must be falsified by any model h to $S \setminus \{c\}$.

We use this property as follows. Let $P = [c_0 = c, c_1, \ldots, c_m]$ be a path in the resolution proof starting from a candidate clause c. P is the *longest unique prefix* if it is the longest path starting at c, such that each $c_i \in P$ has only one child (that is, c participates in the derivation of one clause only). *Path strengthening* is based on the following property, induced by cut falsifiability: all the clauses of P must be falsified in any model h to $S \setminus \{c\}$. Fig. 2(b) shows a variant of the main algorithm in which path strengthening has been applied: each invocation of the SAT solver is carried out under the assumptions $\neg P = \{\neg c_0, \ldots, \neg c_m\}$. Before each iteration our algorithm attempts to increase P length by removing from the resolution proof clauses that are not backward reachable from the empty clause. Note that whenever P contains clauses which do not subsume c, path strengthening will provide more assumptions to the solver than redundancy removal; hence path strengthening is expected to be more efficient than redundancy removal.

Cut falsifiability-based techniques are not immediately compliant with clause set refinement, since clause set refinement requires solving *without assumptions*. `MUSer2` solves this problem for redundancy removal by applying clause set refinement only when the assumptions are not used in the proof; otherwise it skips clause set refinement. Our path strengthening algorithm applies clause set refinement when either the assumptions are not used in the proof or whenever the N latest iterations applied path strengthening and the result was unsatisfiable, N being a user-given threshold.

III. Experimental Results

We checked the impact of our algorithms when applied to the 295 instances used for the MUS track of the SAT 2011 competition. For the experiments we used machines with 32Gb of memory running Intel® Xeon® processors with 3Ghz CPU frequency. The time-out was set to 1800 sec. The implementation was done in `HaifaMUC`. We refer to a configuration of `HaifaMUC` that implements the deletion-based algorithm with incremental SAT and clause set refinement as Base. We compare our tool to the latest version of `MUSer2` [7] and `Minisatabb` [18]. Extended experimental data is available from the second author's home page.

Fig. 3 summarizes the main results. Several observations are in order: 1) rotation is very useful; 2) eager rotation is effective; 3) optimizations **A** and **D** are useful, while optimization **B** is beneficial only if delayed until the second satisfiable iteration (2 being the optimal value, based on experiments); 4) path strengthening (with N=20, 20 being the optimal value experimentally) is more beneficial than redundancy removal, and finally 5) `HaifaMUC`, enhanced by all our algorithms, is 2.18x faster than `MUSer2` and solves 13 more instances, and is 48% faster than `Minisatabb` and solves 4 more instances. `HaifaMUC` is faster than `Minisatabb` on 196 instances, while `Minisatabb` is faster than `HaifaMUC` on 15 instances. Fig. 5 shows a cactus plot comparing Base, `MUSer2`, `Minisatabb` and the new best configuration of `HaifaMUC`, while Fig. 4 compares `HaifaMUC` to `Minisatabb`.

IV. Conclusion

We proposed a number of algorithms for speeding up MUS extraction. First, we adapted GMUS-oriented MUS-biased search algorithms to plain MUS extraction. Second, we integrated model rotation into resolution-based MUS extraction. Third, we introduced an enhancement to rotation, called eager rotation. Finally, we introduced a new enhancement, path strengthening, to resolution-based MUS extraction. We implemented the algorithms in the resolution-based MUS extractor `HaifaMUC`, which, as a result, outperformed the leading MUS extractors `MUSer2` and `Minisatabb`.

V. Acknowledgments

The authors would like to thank Daher Kaiss for supporting this work and Paul Inbar for editing the paper.

References

[1] Silva, J.P.M.: Minimal unsatisfiability: Models, algorithms and applications (invited paper). In: ISMVL'10. (2010) 9–14
[2] Nadel, A.: Boosting minimal unsatisfiable core extraction. In: FMCAD'10. (2010) 221–229
[3] Ryvchin, V., Strichman, O.: Faster extraction of high-level minimal unsatisfiable cores. In: SAT'11. (2011) 174–187
[4] Silva, J.P.M., Lynce, I.: On improving MUS extraction algorithms. In: SAT'11. (2011) 159–173
[5] Wieringa, S.: Understanding, improving and parallelizing MUS finding using model rotation. In: CP'12. (2012) 672–687
[6] Belov, A., Lynce, I., Marques-Silva, J.: Towards efficient MUS extraction. AI Commun. **25**(2) (2012) 97–116
[7] Belov, A., Marques-Silva, J.: MUSer2: An efficient MUS extractor. JSAT **8**(1/2) (2012) 123–128
[8] Zhang, L., Malik, S.: Extracting small unsatisfiable cores from unsatisfiable Boolean formula. In: Preliminary Proceedings of SAT'03. (2003)
[9] Goldberg, E., Novikov, Y.: Verification of proofs of unsatisfiability for CNF formulas. In: DATE'03. (2003) 886–891
[10] Strichman, O.: Pruning techniques for the SAT-based bounded model checking problem. In: CHARME'01. (2001) 58–70
[11] Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electr. Notes Theor. Comput. Sci. **89**(4) (2003)
[12] Dershowitz, N., Hanna, Z., Nadel, A.: A scalable algorithm for minimal unsatisfiable core extraction. In: SAT'06. (2006) 36–41
[13] Chinneck, J.W., Dravnieks, E.W.: Locating minimal infeasible constraint sets in linear programs. INFORMS Journal on Computing **3**(2) (1991) 157–168
[14] Bakker, R.R., Dikker, F., Tempelman, F., Wognum, P.M.: Diagnosing and solving over-determined constraint satisfaction problems. In: IJCAI'93. (1993) 276–281
[15] Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. J. Autom. Reasoning **40**(1) (2008) 1–33
[16] Belov, A., Marques-Silva, J.: Accelerating MUS extraction with recursive model rotation. In: FMCAD'11. (2011) 37–40
[17] van Maaren, H., Wieringa, S.: Finding guaranteed MUSes fast. In: SAT'08. (2008) 291–304
[18] Lagniez, J.M., Biere, A.: Factoring out assumptions to speed up MUS extraction. In: SAT'13. (2013) 276–292
[19] Sörensson, N., Biere, A.: Minimizing learned clauses. In: SAT'09. (2009) 237–243
[20] Nadel, A.: Understanding and Improving a Modern SAT Solver. PhD thesis, Tel Aviv University, Tel Aviv, Israel (August 2009)

```
1: function RMR(S, M, c, h)   ▷ recursive model rotation
2:     for all x ∈ Var(S) do
3:         h' = h[x ← ¬x];          ▷ swap assignment of x
4:         if UnsatSet(S, h') = {c'} and c' ∉ M then
5:             M = M ∪ {c'};
6:             RMR (S, M, c', h');
```

(a)

```
1: function ERMR(S, M, K, c, h)   ▷ Initially K = {c}
2:     for all x ∈ Var(S) do
3:         h' = h[x ← ¬x];
4:         if UnsatSet(S, h') = {c'} and c' ∉ K then
5:             K = K ∪ {c'};
6:             if c' ∉ M then M = M ∪ {c'};
7:             ERMR (S, M, K, c', h');
```

(b)

Fig. 1. (a) The recursive model rotation of [16], where $UnsatSet(S, h')$ is the subset of S's clauses that are unsatisfied by the assignment h', and (b) our modified version. K is a set of clauses that is initialized to c before calling ERMR. $K \subseteq M$ is an invariant, and hence ERMR is called at least as many times as RMR in (a).

```
1: function MUS(unsatisfiable formula S)
2:     M = ∅;
3:     while true do
4:         choose c ∈ S \ M. If there is none, break;
5:         if SAT(S \ {c}) then
6:             K = {c};
7:             M = ERMR (S, c, M, K, h)
8:         else
9:             S = core;
```

(a)

```
1: function MUS(unsatisfiable formula S)
2:     M = ∅;
3:     while true do
4:         choose c ∈ S \ M. If there is none, break;
5:         let P be the longest unique prefix
6:         discard clauses not backward reachable from □
7:         if SAT(S \ {c}, {¬c_i | c_i ∈ P}) then
8:             K = {c}; M = ERMR (S, c, M, K, h)
9:         else
10:            if ¬P not used in proof then S = core;
11:            else
12:                S = S \ {c}
13:                if condition then   ▷ Heuristic. See text
14:                    SAT(S);         ▷ guaranteed unsat
15:                    S = core;
```

(b)

Fig. 2. (a) Deletion-based MUS extraction enhanced by eager rotation and clause set refinement, where h is the satisfying assignment, and $core$ is the unsatisfiable core (b) an improvement based on path strengthening. In line 7 the literals defined by $\{\neg c_i \mid c_i \in P\}$ are assumptions.

	Base	rot	erot	erot_AD	erot_ABD	erot_AB2D	erot_AB2CD	**erot_AB2CD_rr**	**erot_AB2CD_ps20**	MUSer2	Minisatabb
Time	93931	48018	44335	36295	37798	32968	32918	30800	**27263**	59502	40485
Unsolved	30	12	10	8	13	8	8	6	**4**	17	8

Fig. 3. Total run-time in sec. and number of unsolved instances for various solvers, when applied to the 295 instances from the 2011 MUS competition, excluding 12 instances which were not solved by any of the solvers (the time-out value of 1800 sec. was added to the run-time when a memory-out occured). Base is defined in Sect. III, rot = Base+rotation, erot = Base+eager rotation. **A**, **B**, **C**, and **D** correspond to the optimizations defined in Sect. II-A. '2' in AB2CD means that the optimization was invoked after the 2nd satisfiable result. 'rr' refers to redundancy removal combined with clause set refinement using `MUSer2`'s scheme, described in Sect. II-C. 'ps20' means that path strengthening with $N = 20$ was applied as described in Sect. II-C.

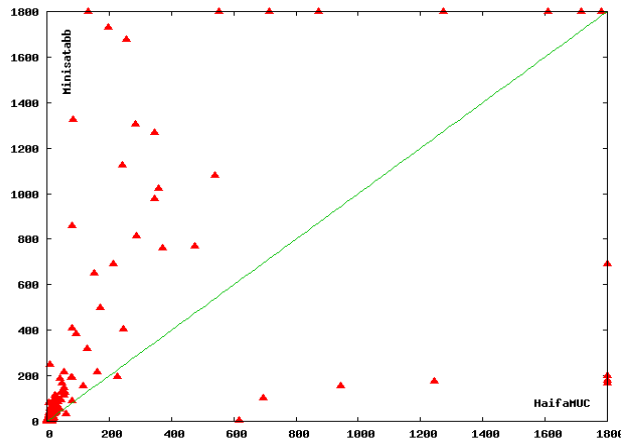

Fig. 4. Direct comparison of the new best configuration of `HaifaMUC` erot_AB2CD_ps20 (X-Axis) and `Minisatabb` (Y-Axis).

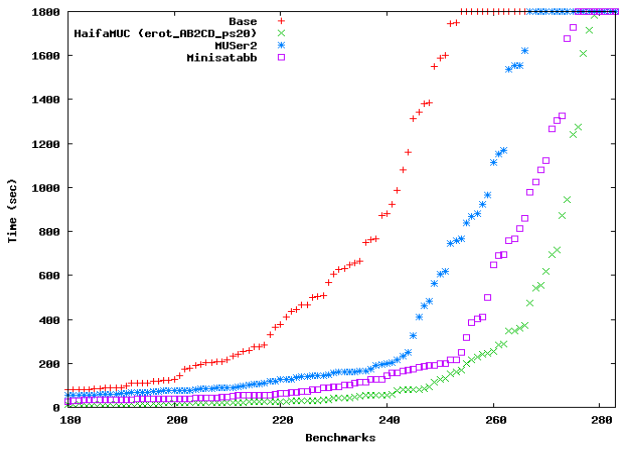

Fig. 5. Comparison of Base, `MUSer2`, `Minisatabb`, and the new best configuration of `HaifaMUC` erot_AB2CD_ps20. The graph shows the number of solved instances (X-Axis) per time-out in seconds (Y-Axis) for each solver.