Preliminary draft

MaxSAT Evaluation 2020

Solver and Benchmark Descriptions

Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins *(editors)*

University of Helsinki
Department of Computer Science
Series of Publications B

Helsinki 2020

PREFACE

The MaxSAT Evaluations (`https://maxsat-evaluations.github.io`) are a series of events focusing on the evaluation of current state-of-the-art systems for solving optimization problems via the Boolean optimization paradigm of maximum satisfiability (MaxSAT). Organized yearly starting from 2006, the year 2020 brought on the 15th edition of the MaxSAT Evaluations, organized as a satellite event of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT 2020). Some of the central motivations for the MaxSAT Evaluation series are to provide further incentives for further improving the empirical performance of the current state of the art in MaxSAT solving, to promote MaxSAT as a serious alternative approach to solving NP-hard optimization problems from the real world, and to provide the community at large heterogenous benchmark sets for solver development and research purposes. In the spirit of a true evaluation—rather than a competition, unlike e.g. the SAT Competition series—no winners are declared, and *no awards or medals are handed out* to overall best-performing solvers.

The 2020 evaluation consisted of a total of four tracks: three for complete solvers (one for solvers focusing on unweighted and one for solvers focusing on weighted MaxSAT instances, as well as a "Top-k" special track on enumerating top-k solutions, new for 2020) and a special track for incomplete MaxSAT solvers (using two short per-instance time limits, 60 and 300 seconds, differentiating from the per-instance time limit of 1 hour imposed in the main complete tracks). As in 2017-2019, no distinction was made between "industrial" and "crafted" benchmarks, and no track for purely randomly generated MaxSAT instances was organized.

Adhering to the new rules introduced in 2017, solvers were now required to be open-source, and the source codes of all participating solvers were made available online on the evaluation webpages after the evaluation results were presented at the SAT 2020 conference. Furthermore, a 1-2 page solver description was required for each solver submission, to provide some details on the search techniques implemented in the solvers. The solvers descriptions together with descriptions of new benchmarks for 2020 are collected together in this compilation.

Finally, we would like to thank everyone who contributed to MaxSAT Evaluation 2020 by submitting their solvers or new benchmarks. We are also grateful for the computational resources provided by the StarExec initiative which enabled running the 2020 evaluation smoothly.

Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, & Ruben Martins
MaxSAT Evaluation 2020 Organizers

Contents

Benchmark Descriptions

SOLVER DESCRIPTIONS

A short description of the solver EvalMaxSAT

Florent Avellaneda
Computer Research Institute of Montreal
Montreal, Canada
florent.avellaneda@gmail.com

I. Introduction

EvalMaxSAT[1] is a MaxSAT solver written in modern C++ language mainly using the Standard Template Library (STL). The solver is built on top of the SAT solver Glucose [1], but any other SAT solver can easily be used instead. EvalMaxSAT is based on the OLL algorithm [2] originally implemented in the MSCG MaxSAT solver [3], [4] and then reused in the RC2 solver [5].

The OLL algorithm considers all soft variables as hard and attempts to solve the formula. If the formula has no solution, then a conjunction of soft variables that cannot be satisfied (a *core*) is extracted. Each variable constituting this core is then *relaxed* (removed from the list of soft variables or incremented if it is a cardinality) and a new cardinality is added to the list of soft variables encoding the constraint "at most one variable from the core can be false". When the formula is finally satisfied, we obtain a MaxSAT assignment.

In practice, the size of the cores plays an important role in the performance of this algorithm. Indeed, the more variables the cores contain, the more expensive the encoding of cardinalities will be. Thus, once a core is found, a core minimization phase consists of removing unnecessary variables. Although heuristics are generally used to perform this minimization, this phase remains very expensive. EvalMaxSAT performs this minimization several times by calling the solver SAT with a limited number of conflicts. In addition, the algorithm used can easily be adapted to perform the minimization in parallel with the *core* searching.

II. Description

The algorithm used is a modification of the OLL algorithm (see Algorithm 1). The main modification is that when a core is found and minimized, new variables and constraints are not added to the SAT solver immediately. All these new constraints will be added only when the formula becomes satisfiable, or when finding a new solution takes too much time. By doing that, this algorithm tries to reduce the number of implications leading from cardinality to a soft variable.

A second modification made by the EvalMaxSAT solver is in the minimize function. Indeed, this function will perform several minimizations in order to obtain small cores. A first minimization is done by making successive calls to *solver(core)* where solver calls are limited to zero conflicts. Each call to the solver attempts to remove a literal from the core (a literal can be removed if the formulation remains unsatisfiable). After that, we apply the same algorithm with 1000 limited conflicts by considering the variables in differing orders.

[1]See https://github.com/FlorentAvellaneda/EvalMaxSAT

Algorithm 1 (Pseudo-code of the sequential algorithm)

Input: A formula φ

1: $cost \leftarrow extractAM1(\varphi)$
2: **while** $true$ **do**
3: $(st, \varphi_c) \leftarrow SATSolver(\varphi)$
4: **if** $st = true$ **then**
5: $\varphi \leftarrow \varphi \cup \varphi_{tmp}$
6: $(st, \varphi_c) \leftarrow SATSolver(\varphi)$
7: **if** $st = true$ **then**
8: **return** $cost$
9: **end if**
10: **end if**
11: $\varphi_c \leftarrow minimize(\varphi, \varphi_c)$
12: $k \leftarrow exhaust(\varphi, \varphi_c)$
13: $cost \leftarrow cost + k$
14: $\varphi \leftarrow relax(\varphi, \varphi_c)$
15: $\varphi_{tmp} \leftarrow \varphi_{tmp} \cup createSum(\varphi_c, k)$
16: **end while**

III. Implementation Details

Extract AM1 Two algorithms are used to extract AtMost1 constraints from soft variables. The first one uses the mcqd library [6] to find the maximum clique in the incompatibility graph of the soft variables; the second one uses a heuristic.

Cardinality The Totalizer Encoding [7] is used to represent cardinalities. The implementation reuses the code from the PySAT's ITotalizer [8].

Exhaust After the minimization is performed, a *core exhaustion* [5] or *cover optimization* [9] is done.

Timeout Many timeouts are used to stop minimization when they take too much time.

IV. Multicore version

An interesting feature of the algorithm used is that it is very simple to parallelize it. Although the competition does not allow the multi-threaded calculation, this feature has been implemented in the solver **but disabled for the competition**. The architecture of the parallelized algorithm is depicted in Figure 1. The main thread looks for new cores to minimize and when it finds one, it removes all variables

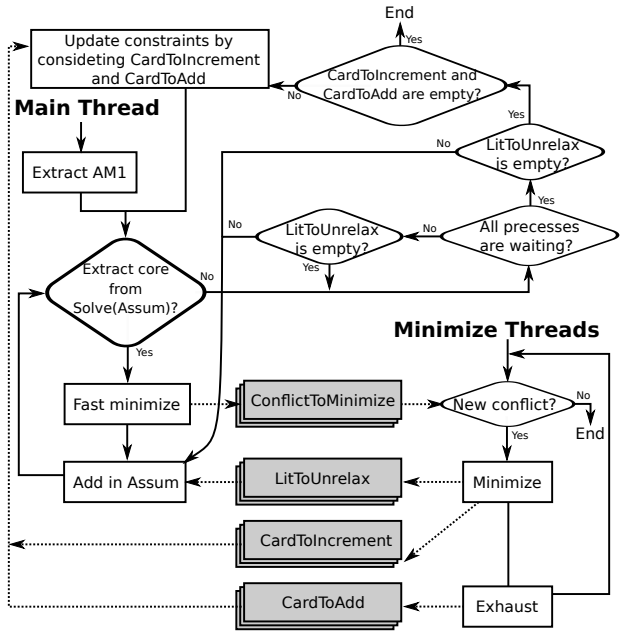


Fig. 1. Algorithm architecture

present in the core from the list of soft variables (Assum). Paralleling this, threads access the cores found by the main thread (ConflictToMinimize), minimize them and share new cardinalities (CardToAdd), unused variables (LitToUnrelax) and cardinalities to be incremented (CardToIncrement). Before searching for new cores, the main thread collects the variables that had previously been removed but were not used in any previous thread.

When the main thread no longer finds a core, all minimization threads have been completed and no variables are to be reconsidered as soft, then the main thread considers the new cardinalities to be added and incremented before restarting the *core* search. If there are no cardinalities to add or increment, then the search is complete and we get a MaxSAT assignment.

REFERENCES

[1] G. Audemard, J. Lagniez, and L. Simon, "Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction," in *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, ser. Lecture Notes in Computer Science, M. Järvisalo and A. V. Gelder, Eds., vol. 7962. Springer, 2013, pp. 309–317. [Online]. Available: https://doi.org/10.1007/978-3-642-39071-5_23

[2] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided MaxSAT with soft cardinality constraints," in *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, ser. Lecture Notes in Computer Science, B. O'Sullivan, Ed., vol. 8656. Springer, 2014, pp. 564–573. [Online]. Available: https://doi.org/10.1007/978-3-319-10428-7_41

[3] A. Morgado, A. Ignatiev, and J. Marques-Silva, "MSCG: robust core-guided maxsat solving," *J. Satisf. Boolean Model. Comput.*, vol. 9, no. 1, pp. 129–134, 2014. [Online]. Available: https://satassociation.org/jsat/index.php/jsat/article/view/127

[4] A. Ignatiev, A. Morgado, V. M. Manquinho, I. Lynce, and J. Marques-Silva, "Progression in maximum satisfiability," in *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, ser. Frontiers in Artificial Intelligence and Applications, T. Schaub, G. Friedrich, and B. O'Sullivan, Eds., vol. 263. IOS Press, 2014, pp. 453–458. [Online]. Available: https://doi.org/10.3233/978-1-61499-419-0-453

[5] A. Ignatiev, A. Morgado, and J. Marques-Silva, "RC2: an efficient maxsat solver," *J. Satisf. Boolean Model. Comput.*, vol. 11, no. 1, pp. 53–64, 2019. [Online]. Available: https://doi.org/10.3233/SAT190116

[6] J. Konc and D. Janezic, "An improved branch and bound algorithm for the maximum clique problem," *proteins*, vol. 4, no. 5, 2007.

[7] R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce, "Reflections on "incremental cardinality constraints for maxsat"," *CoRR*, vol. abs/1910.04643, 2019. [Online]. Available: http://arxiv.org/abs/1910.04643

[8] A. Ignatiev, A. Morgado, and J. Marques-Silva, "Pysat: A python toolkit for prototyping with SAT oracles," in *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, ser. Lecture Notes in Computer Science, O. Beyersdorff and C. M. Wintersteiger, Eds., vol. 10929. Springer, 2018, pp. 428–437. [Online]. Available: https://doi.org/10.1007/978-3-319-94144-8_26

[9] C. Ansótegui, M. L. Bonet, J. Gabas, and J. Levy, "Improving wpm2 for (weighted) partial maxsat," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2013, pp. 117–132.

Loandra in the 2020 MaxSAT Evaluation

Jeremias Berg*, Emir Demirović†, Peter Stuckey‡
*HIIT, Department of Computer Science, University of Helsinki, Finland
†Delft University of Technology, The Netherlands
‡Monash University, Australia

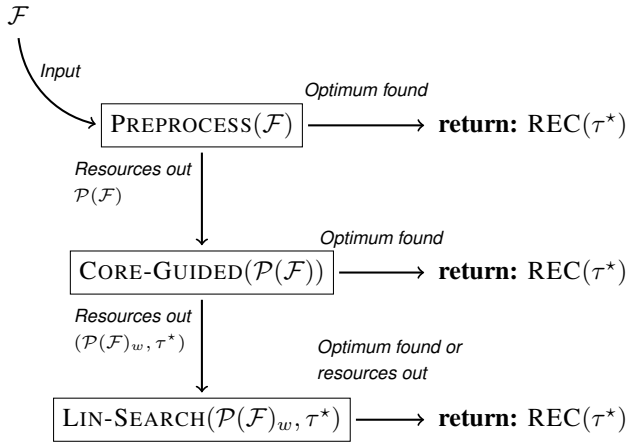


Fig. 1: The structure of Loandra.

I. PRELIMINARIES

We briefly overview the Loandra MaxSAT-solver as it participated in the incomplete track of the 2020 MaxSAT Evaluation, focusing especially on the differences between the 2019 and 2020 versions, more detailed descriptions can be found in [4], [10]. Loandra owes much of its existence to Open-WBO [11], we thank the developers of Open-WBO for their work .

We assume familiarity with conjunctive normal form (CNF) formulas and weighted partial maximum satisfiability (MaxSAT). Treating a CNF formula as a set of clauses a MaxSAT instance $\mathcal{F}$ consists of two CNF formulas, the hard clauses F_h and the soft clauses F_s, as well a weight w_c associated with each $C \in F_s$. A solution to $\mathcal{F}$ is an assignment τ that satisfies F_h. The cost of a solution τ is the sum of weights of the soft clauses falsified by τ. An optimal solution is one with minimum cost over all solutions. An unsatisfiable core κ of $\mathcal{F}$ is a subset of soft clauses s.t. $F_h \wedge \kappa$ is unsatisfiable.

II. STRUCTURE OF LOANDRA

Figure 1 overviews the structure of Loandra. In short, Loandra implements core-boosted linear search [4] augmented with tightly integrated MaxSAT preprocessing [3], [9], [10], [2]. More specifically, Loandra consists of three main components: a) Preprocessing, b) Core-guided search, c) Linear search.

a) Preprocessing: On input $\mathcal{F}$, the execution starts by invoking the MaxPre [9] preprocessor on $\mathcal{F}$ using the standard techniques of MaxPre *except for blocked clause elimination (BCE)*. The reason we are not using BCE is that, as detailed in [10], intermediate solutions to instances preprocessed with BCE are more prone to having their costs miss-interpreted by the core-guided and linear search components of Loandra. If MaxPre can not compute the optimal solution to $\mathcal{F}$, the preprocessed instance $\mathcal{P}(\mathcal{F})$ is handed to the core guided phase (CORE-GUIDED in Figure 1), reusing the assumption variables introduced during preprocessing [3].

b) Core-guided search: The core-guided phase is unchanged from the 2019 version; as the instantiation of the core-guided algorithm, we use a reimplementation of PMRES [12] extended with weight aware core extraction (WCE) [5] and clause hardening. If CORE-GUIDED is able to find an optimal solution τ to $\mathcal{P}(\mathcal{F})$, an optimal solution $\text{REC}(\tau)$ to $\mathcal{F}$ is reconstructed and returned. Otherwise i.e. if the core-guided phase runs out of time, the final working instance $\mathcal{P}(\mathcal{F})_w$ and τ^*, the best found solution to it is handed to the linear search component LIN-SEARCH.

c) Linear search: LIN-SEARCH, the linear search phase of Loandra is an implementation of the SAT/UNSAT linear search algorithm [6], extended with solution guided phase saving and varying resolution in the style of LinSBPS [7]. The component is for the most part the same as in the 2019 version. The main difference is, that in the start of each resolution, the currently best known solution τ^* is minimized in order to alleviate the missinterpretation of costs that might happen due to preprocessing in the context of incomplete solving [10].

More specifically, let $\mathcal{P}(\mathcal{F})_w = \mathcal{P}(\mathcal{F}) \cup \text{CARD}$ be the working instance of LIN-SEARCH where $\mathcal{P}(\mathcal{F})$ is the preprocessed instance computed by MaxPre and CARD are the constraints added in the core-guided phase. At the start of each resolution, LIN-SEARCH computes a set $\mathcal{B}_s$ of blocking variables and an upper bound UB over which the PB constraint is built. The upper bound is computed based on τ^*, the current best known solution to $\mathcal{P}(\mathcal{F})_w$. However, as shown in [10], there can be a significant difference between $\text{COST}(\mathcal{P}(\mathcal{F}), \tau^*)$, the cost of τ^* w.r.t to $\mathcal{P}(\mathcal{F})$, and $\text{COST}(\mathcal{F}, \text{REC}(\tau^*))$, the cost of the solution to $\mathcal{F}$ reconstructed from τ^*. This difference might result UB, and as consequence the whole PB constraint, being much larger than actually required. In order to alleviate this issue LIN-SEARCH uses a simple, procedure that iteratively fixes all variables in $\mathcal{B}_s$ in the following manner. In each iteration, all variables in $\mathcal{P}(\mathcal{F})$

are fixed to the polarities that they are assigned to by τ^*. Additionally an unfixed variable $b \in \mathcal{B}_s$ is fixed to false (i.e. to not incur cost). Then the SAT solver is used to extend these fixings into a satisfying assignment of $\mathcal{P}(\mathcal{F})_w$. If such an assignment τ_b^* can be found, that assignment will have $\text{COST}(\mathcal{P}(\mathcal{F}), \tau_b^*) < \text{COST}(\mathcal{P}(\mathcal{F}), \tau^*)$ while also agreeing with τ^* on all variables in $\mathcal{P}(\mathcal{F})$. The variable b is then fixed to false in subsequent iterations. Otherwise, the variable b is fixed to true in subsequent iterations. Notice that, due to the nature of constraints added by CORE-GUIDED, each individual SAT-solver call is solvable by unit propagation alone (the constraints in CARD are basically cardinality constraints). Even so, preliminary experiments showed that the minimization procedure is too expensive to run for each new solution found, which is why we restrict it to once per resolution.

The linear phase runs until either finding an optimal solution, or running out of time, at which point a reconstruction $\text{REC}(\tau^*)$ of the currently best known solution τ^* to $\mathcal{P}(\mathcal{F})_w$ is returned. Notice that the reconstruction of a solution happens only once, we use the standard, linear time, reconstruction algorithm as implemented by MaxPre.

III. IMPLEMENTATION DETAILS

All algorithms are implemented on top of the publicly available Open-WBO system [11] using Glucose 4.1 [1] as the back-end SAT solver. In order to minimize I/O overhead, we make direct use of the preprocessor interface offered by MaxPre. The linear search algorithm uses the generalized totalizer encoding [8] to convert the PB constraints needed in linear search to CNF. In the evaluation, we set a 30s time limit for the preprocessing phase and a 30 second time limit for the core-guided phase. These limits were chosen based on preliminary experiments. On weighted instances, the core-guided phase is also terminated when the stratification bound would be lowered to 1. On unweighted instances the phase is terminated at the latest after extracting one set of disjoint cores.

IV. COMPILATION AND USAGE

Building and using Loandra resembles building and using Open-WBO. Before building loandra, the maxpre library needs to be built by invoking MAKE LIB in the maxpre subfolder. Afterwards, a statically linked version of Loandra in release mode can be built by running MAKE RS in the base folder.

After building, Loandra can be invoked from the terminal. Except for the formula file, Loandra accepts a number of command line arguments: the flag *-pmreslin-cglim* sets the maximum time that the core-guided phase can run for (in seconds). The rest of the flags resemble the flags accepted by Open-WBO; invoke ./loandra_static –help-verb for more information.

REFERENCES

[1] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *Proc IJCAI*. Morgan Kaufmann Publishers Inc., 2009, pp. 399–404.

[2] A. Belov, A. Morgado, and J. Marques-Silva, "SAT-based preprocessing for MaxSAT," in *Proc. LPAR-19*, ser. Lecture Notes in Computer Science, vol. 8312. Springer, 2013, pp. 96–111.

[3] J. Berg, P. Saikko, and M. Järvisalo, "Improving the effectiveness of SAT-based preprocessing for MaxSAT," in *Proc. IJCAI*. AAAI Press, 2015, pp. 239–245.

[4] J. Berg, E. Demirovic, and P. J. Stuckey, "Core-boosted linear search for incomplete maxsat," in *CPAIOR*, ser. Lecture Notes in Computer Science, vol. 11494. Springer, 2019, pp. 39–56.

[5] J. Berg and M. Järvisalo, "Weight-aware core extraction in SAT-based MaxSAT solving," in *Proc. CP*, ser. Lecture Notes in Computer Science, 2017, to appear.

[6] D. L. Berre and A. Parrain, "The sat4j library, release 2.2," *J. Satisf. Boolean Model. Comput.*, vol. 7, no. 2-3, pp. 59–6, 2010. [Online]. Available: https://satassociation.org/jsat/index.php/jsat/article/view/82

[7] E. Demirovic and P. J. Stuckey, "Techniques inspired by local search for incomplete maxsat and the linear algorithm: Varying resolution and solution-guided search," in *CP*, ser. Lecture Notes in Computer Science, vol. 11802. Springer, 2019, pp. 177–194.

[8] S. Joshi, R. Martins, and V. M. Manquinho, "Generalized totalizer encoding for pseudo-boolean constraints," in *Proc. CP*, ser. LNCS, vol. 9255, 2015, pp. 200–209.

[9] T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo, "Maxpre: An extended maxsat preprocessor," in *SAT*, ser. Lecture Notes in Computer Science, vol. 10491. Springer, 2017, pp. 449–456.

[10] M. Leivo, J. Berg, and M. Järvisalo, "Preprocessing in incomplete maxsat solving," in *Proc ECAI*, ser. Frontiers in Artificial Intelligence and Applications, vol. ???? IOS Press, 2020, p. (to appear).

[11] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: A modular MaxSAT solver," in *Proc. SAT*, ser. Lecture Notes in Computer Science, vol. 8561. Springer, 2014, pp. 438–445.

[12] N. Narodytska and F. Bacchus, "Maximum satisfiability using core-guided MaxSAT resolution," in *Proc. AAAI*. AAAI Press, 2014, pp. 2717–2723.

Pacose: An Iterative SAT-based MaxSAT Solver

Tobias Paxian, Bernd Becker
Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 051
79110 Freiburg, Germany
{paxiant|becker}@informatik.uni-freiburg.de

I. Overview

Pacose is a SAT-based MaxSAT solver, using two incremental CNF encodings, a binary adder [1] and the Dynamic Polynomial Watchdog (DPW) [2], for Pseudo-Boolean (PB) constraints. It is an extension of QMaxSAT 2017 [3], based on Glucose 4.2.1 [4] SAT solver. It uses a Boolean Multilevel Optimization (BMO) pre- / inprocessing method to simplify the instances. Additionally a trimming method is applied to cut off unsatisfiable soft clauses and find a good initial satisfiable weight to reduce the size of the encoding.

II. Pre- / Inprocessing

The 2019 version of Pacose contains two new pre-/inprocessing methods, Generalized Boolean Multilevel Optimization (GBMO) and a trimming algorithm.

Multi-Objective Combinatorial Optimization (MOCO) [5] problems are addressing multiple optimization problems with possibly conflicting purposes. Boolean Multilevel Optimization (BMO) [6], [7] is the mapping of MOCO to MaxSAT solving. We generalized the plain variant of Boolean Multilevel Optimization thereby making it possible to split additional instances, even in cases where the weight differences of the sum of smaller weights is non-strictly smaller than the next biggest weight.

The trimming algorithm tries to satisfy each soft clause at least once with the additional goal to find a good approximation of the weight. It works in two phases, in the first phase it optimizes the overall weight and in the second phase it satisfies as many soft clauses as possible in the next solver call. After a timeout which is based on the number of soft clauses, it switches from the first phase to the second. An additional timeout for each incrementally solver call is included.

III. Encoding and Algorithm

Our DPW encoding is based on the Polynomial Watchdog (PW) encoding [8], which uses totalizer networks [9]. Essentially the DPW encoding employs multiple totalizer networks to perform a binary addition with carry on the sorted outputs. A special algorithm to solve these instances incremental is presented in [2].

Additionally the adder network [1] is used which has a linear complexity in encoding size in contrast to at least $\mathcal{O}(n^2)$ for the DPW sorting network. With the adder network many complementary instances to the DPW encoding can be solved and therefore it is well suited, to be chosen, together with DPW by a heuristic, as described in the following chapter. The algorithm and encoding are partly adapted and inspired from QMaxSAT.

This work is supported by DFG "Algebraic Fault Attacks" (BE 1176/20-2)

IV. Heuristics

Pacose uses straightforward heuristics based on available MaxSAT benchmarks. All heuristics are based on the number of soft clauses and the overall sum of soft weights.

- *Encoding:* The DPW encoding empirically works best if the average weight for soft clauses is small, or the overall sum of soft weights is huge (bigger than 80 billion). For the other benchmarks the binary adder is chosen.
- *Trimming:* As for instances with only a few soft clauses the trimming preprocessing algorithm is not effective, it is only used if the benchmark contains at least a certain amount of soft clauses.
- *Compression Rate:* For benchmarks with only a few soft clauses, the encoding is smaller and additional clauses can be added. Therefore the binary adder encoding can solve overall more benchmarks if the compression rate is chosen accordingly.

References

[1] J. P. Warners, "A linear-time transformation of linear inequalities into conjunctive normal form," *Information Processing Letters*, vol. 68, no. 2, pp. 63–69, 1998.
[2] T. Paxian, S. Reimer, and B. Becker, "Dynamic polynomial watchdog encoding for solving weighted MaxSAT," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2018, pp. 37–53.
[3] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A partial Max-SAT solver system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 8, pp. 95–100, 2012.
[4] G. Audemard and L. Simon, "On the glucose SAT solver," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 01, p. 1840001, 2018.
[5] E. L. Ulungu and J. Teghem, "Multi-objective combinatorial optimization problems: A survey," *Journal of Multi-Criteria Decision Analysis*, vol. 3, no. 2, pp. 83–104, 1994.
[6] J. Argelich, I. Lynce, and J. Marques-Silva, "On solving boolean multilevel optimization problems," in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.
[7] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3-4, pp. 317–343, 2011.
[8] O. Bailleux, Y. Boufkhad, and O. Roussel, "New encodings of pseudo-boolean constraints into CNF," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2009, pp. 181–194.
[9] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of Boolean cardinality constraints," in *Principles and Practice of Constraint Programming–CP 2003*. Springer, 2003, pp. 108–122.

RC2-2018 @ MaxSAT Evaluation 2020

Alexey Ignatiev
Monash University, Australia
alexey.ignatiev@monash.edu

I. Introduction

RC2 is an open-source MaxSAT solver written in Python and based on the PySAT framework[1] [1]. It is designed to serve as a simple example of how SAT-based problem solving algorithms can be implemented using PySAT while sacrificing just a little in terms of performance. In this sense, RC2 can be seen as a solver prototype and can be made somewhat more efficient if implemented in a low-level language. RC2 is written from scratch and implements the RC2/OLLITI (i.e. *relaxable cardinality constraints*) MaxSAT algorithm [2]–[4] originally implemented in the MSCG MaxSAT solver [3], [5]. The RC2 algorithm proved itself efficient in the previous editions of the MaxSAT Evaluation (MSE): namely in 2014, 2015, and 2016 (see the results of the MSCG solver, which was one of the best complete MaxSAT solvers in the aforementioned competitions) and, more recently, in 2018 and 2019 where it was *ranked first* in the complete track of the MSE.

II. Description

For details on the implementation of RC2, a reader is referred to [4]. Here we give a brief overview of the competition versions of RC2-2018.

RC2 supports *incrementally* a variety of SAT solvers provided by PySAT, and its competition version uses Glucose 3.0 [6] as an underlying SAT oracle. Two variants of the solver were submitted to the MaxSAT Evaluation 2018 including RC2-A and RC2-B. Both of these versions implement the same algorithm [2], [3] and share most of the techniques used [3]. Their major components and differences are briefly described below.

III. Variants of the Solver

The following heuristics are used by both solver variants submitted to the MaxSAT Evaluation 2018: incremental SAT solving [7], Boolean lexicographic optimization [8] and stratification [9] for weighted instances, unsatisfiable core exhaustion (originally referred to as cover optimization) [9].

Additionally, the following heuristic was used in both variants of RC2: given a set S of soft clauses, a number of subsets $S' \subseteq S$ were identified such that at most one soft clause in S' can be satisfied, i.e. $\sum_{c \in S'} c \leq 1$. Every subset S' can be treated as an unsatisfiable core of cost $|S'| - 1$, which can be represented as a single clause.

The only difference between the solver variants is the policy for unsatisfiable core minimization. In contrast to RC2-A, RC2-B applies heuristic unsatisfiable core minimization done with a simple deletion-based *minimal unsatisfiable subset* (*MUS*) extraction algorithm [10]. During the core minimization phase in RC2-B, all SAT calls are dropped after obtaining 1000 conflicts. Note that core minimization in RC2-B is disabled for large *plain* MaxSAT formulas, i.e. those having no hard clauses but more than 100000 soft clauses. The reason is that having this many soft clauses (and, thus, as many assumption literals) and no hard clauses is deemed to make SAT calls too expensive. Although core minimization is disabled in RC2-A, reducing the size of unsatisfiable cores can be still helpful for weighted instances due to the nature of the RC2/OLLITI algorithm, i.e. because of the clause splitting applied to the clauses of an unsatisfiable core depending on their weight. Therefore, when dealing with weighted instances RC2-A *trims* unsatisfiable cores at most 5 times (e.g. see [3] for details) aiming at getting rid of unnecessary clauses. Note that core trimming is disabled in RC2-A for unweighted MaxSAT instances and it is not used in RC2-B at all.

IV. MaxSAT Evaluation 2020

The solver submitted to MSE 2020 brings *no changes* to the underlying algorithm and heuristics. The only modifications made include: (1) support of the new v-line format and (2) support of *Top-k* track[2] of the evalution. The latter is done through *maximal satisfiable subset* (*MSS*) enumeration. In the model enumeration mode, soft clause *hardening*, which is normally operating when BLO and stratification are active, is disabled. Also, as soon as the first top solution is computed, BLO and stratification are disabled and the solver proceeds in the standard mode as all soft clauses of the formula get active.

V. Availability

RC2 is distributed as a part of the PySAT framework, which is available under an MIT license at `https://github.com/pysathq/pysat`. It can also be installed as a Python package from PyPI:

```
pip install python-sat
```

The RC2 solver can be used as a standalone executable `rc2.py` and can also integrated into a complex Python-based problem solving tool, e.g. using the standard *import* interface of Python:

```
from pysat.examples import rc2
```

[1] `http://pysathq.github.io`

[2] `https://maxsat-evaluations.github.io/2020/topk.html`

REFERENCES

[1] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: a Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, to appear.

[2] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided MaxSAT with soft cardinality constraints," in *CP*, 2014, pp. 564–573.

[3] A. Morgado, A. Ignatiev, and J. Marques-Silva, "MSCG: Robust core-guided MaxSAT solving," *JSAT*, vol. 9, pp. 129–134, 2015.

[4] A. Ignatiev, A. Morgado, and J. Marques-Silva, "RC2: an efficient MaxSAT solver," *J. Satisf. Boolean Model. Comput.*, vol. 11, no. 1, pp. 53–64, 2019.

[5] A. Ignatiev, A. Morgado, V. M. Manquinho, I. Lynce, and J. Marques-Silva, "Progression in maximum satisfiability," in *ECAI*, 2014, pp. 453–458.

[6] G. Audemard, J. Lagniez, and L. Simon, "Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction," in *SAT*, 2013, pp. 309–317.

[7] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electr. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543–560, 2003.

[8] J. Marques-Silva, J. Argelich, A. Graca, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence (AMAI)*, vol. 62, no. 3-4, pp. 317–343, 2011.

[9] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy, "Improving WPM2 for (weighted) partial maxsat," in *CP*, 2013, pp. 117–132.

[10] J. M. Silva, "Minimal unsatisfiability: Models, algorithms and applications (invited paper)," in *ISMVL*, 2010, pp. 9–14.

SATLike-c(w): Solver Description

1st Zhendong Lei
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
leizd@ios.ac.cn

2nd Shaowei Cai
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
caisw@ios.ac.cn

***Abstract*—In this document, we briefly describe the techniques employed by the *SATLike-c(w)* solver participation in MaxSAT Evaluation 2020.**

I. Introduction

SATLike-c(w) participates in incomplete track. *SATLike-c(w)* has two engines, one is local search solver *SATLike* [1] and the other is SAT-based solver *TT-Open-WBO-inc* [2]. First, a core-guided SAT solver is executed to find a feasible solution. Then *SATLike* is executed with this feasible solution as its initial solution. *SATLike* keeps working until it fails to improve the current solution in a given time limit. After that, *TT-Open-WBO-inc* is executed to continue to improve the current solution.

II. Acknowledgement

Thank Zhihan Chen for his contribution to this work.

References

[1] Zhendong Lei and Shaowei Cai. "Solving(weighted) partial maxsat by dynamic local search for SAT." InProceedings of the Twenty-Seventh International Joint Confer-ence on Artificial Intelligence, IJCAI 2018, July 13-19, 2018,Stockholm, Sweden. pages 1346–1352, 2018.

[2] Alexander Nadel. "Anytime weighted maxsat withimproved polarity selection and bit-vector optimization." InClark W. Barrett and Jin Yang, editors,2019 Formal Methodsin Computer Aided Design, FMCAD 2019, San Jose, CA, USA,October 22-25, 2019, pages 193–202. IEEE, 2019.

QMaxSAT in MaxSAT Evaluation 2018

Aolong Zha
Faculty of Information Science and Electrical Engineering
Kyushu University
744 Motooka, Nishi-ku, Fukuoka, Japan
cyouryuuryuu@gmail.com

QMaxSAT is a satisfiability-based solver, which uses CNF encoding of pseudo-Boolean (PB) constraints [1]. The efficiency of MaxSAT solvers depends on critically on which SAT solver we use and how we encode the PB constraints. The QMaxSAT is obtained by adapting a CDCL based SAT solver Glucose 3.0 [2], [3]. In addition, we introduce a new encoding method, called n-level modulo totalizer encoding in to our solver. This encoding is a hybrid between Modulo Totalizer (MTO) [4] and Weighted Totalizer (WTO) [5], incorporating the idea of mixed radix base [6].

Let $\phi = \{(C_1, w_1), \ldots, (C_m, w_m), C_{m+1}, \ldots, C_{m+m'}\}$ be a MaxSAT [7] instance where C_i is a soft clause with weight w_i $(i = 1, \ldots, m)$ and C_{m+j} is a hard clause $(j = 1, \ldots, m')$. We added a new blocking variable, b_i, to each soft clause $C_i (i = 1, \ldots, m)$. Solving the MaxSAT problem for ϕ is reduced to finding a SAT model of $\phi' = \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$, which minimizes $\sum_{i=1}^{m} w_i b_i$.

Such SAT models are obtained using a SAT solver as follows: Run the SAT solver to get an initial model and calculate $k = \sum_i w_i b_i$ in it, add PB constraint $\sum_i w_i b_i < k$, and run the solver again. If ϕ' is unsatisfiable, then ϕ is also unsatisfiable as the MaxSAT problem. Otherwise, the process is repeated with the new smaller solution. The latest model is a MaxSAT solution of ϕ. QMaxSAT leaves the manipulation of the PB constraints to Glucose by encoding them into SAT.

We introduce a hybrid encoding [8] which inherits modular arithmetic from MTO and distinct combinations of weights from WTO. The latter is essentially the same as Generalized Totalizer, which only generate auxiliary variables for each unique combination of weights. We also enhanced the encoding by multi-level modulo arithmetic based on a mixed radix numeral system [9]. This encoding method always produces a polynomial-size CNF in the number of input variables.

It is important to find a suitable mixed radix base with low time-consumption that reduces the number of auxiliary variables for our new encoding. We select the integer whose rate of divisibility is the highest for all weights[1] as the suitable modulus for each digit. Furthermore, we also add other heuristics tailored in our implementation, such as evaluating and voting for the candidates of modulus, dynamically adjusting the lower limit of the required rate of divisibility, etc.

[1]Before selecting the next modulus, we update all the weights to their quotients of dividing the previous selected modulus.

References

[1] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A Partial Max-SAT Solver," *JSAT*, vol. 8, no. 1/2, pp. 95–100, 2012.

[2] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, C. Boutilier, Ed., 2009, pp. 399–404.

[3] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518.

[4] T. Ogawa, Y. Liu, R. Hasegawa, M. Koshimura, and H. Fujita, "Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT Solvers," in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*. IEEE Computer Society, 2013, pp. 9–17.

[5] S. Hayata and R. Hasegawa, "Improvement in CNF Encoding of Cardinality Constraints for Weighted Partial MaxSAT," *SIG-FPAI, in Japanese*, vol. 4, no. 04, pp. 85–90, 2015.

[6] M. Codish, Y. Fekete, C. Fuhs, and P. Schneider-Kamp, "Optimal Base Encodings for Pseudo-Boolean Constraints," in *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, ser. Lecture Notes in Computer Science, P. A. Abdulla and K. R. M. Leino, Eds., vol. 6605. Springer, 2011, pp. 189–204.

[7] C. M. Li and F. Manyà, "MaxSAT, Hard and Soft Constraints," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 185, pp. 613–631.

[8] A. Zha, M. Koshimura, and H. Fujita, "A Hybrid Encoding of Pseudo-Boolean Constraints into CNF," in *Conference on Technologies and Applications of Artificial Intelligence, TAAI 2017, Taipei, Taiwan, December 1-3, 2017*. IEEE, 2017, pp. 9–12.

[9] A. Zha, N. Uemura, M. Koshimura, and H. Fujita, "Mixed Radix Weight Totalizer Encoding for Pseudo-Boolean Constraints," in *2017 IEEE 29th International Conference on Tools with Artificial Intelligence, Boston, MA, USA, November 6-8, 2017*. IEEE Computer Society, 2017, pp. 868–875.

Stable Resolving

Julian Reisch
Synoptics GmbH Dresden, Germany
julian.reisch@synoptics.de

Peter Großmann
Synoptics GmbH Dresden, Germany
peter.grossmann@synoptics.de

***Abstract*—We describe Stable Resolving (SR) [6], a solver competing in the incomplete track of the 2020 MaxSAT evaluation. SR is a randomized local search heuristic for both weighted and unweighted instances. The algorithm consists of three steps that are executed repeatedly. In a perturbation, the search space is explored. Then, local improvements are performed by flipping the signs of variables in over-satisfied clauses. Finally, a simulated annealing solution checking allows for leaving local optima.**

***Index Terms*—MaxSAT, heuristic, local search, incomplete solving**

I. Overall Procedure

The solver starts with a SAT-based preprocessing and a call of the SAT-solver Glucose [2] for an initial solution before the three steps of perturbation, improvements and solution checking are executed repeatedly until the global timeout is reached. The procedure is shown in Algorithm 1.

Algorithm 1 StableResolving()

```
Preprocess()
CalculateInitialSolution()
while timeout has not been reached do
    Perturbation()
    Improvements()
    SolutionChecking()
end
```

This algorithm structure has been proposed by [1] for the maximum independent set problem and applied in an adapted form to transformed MaxSAT instances by [7].

II. Preprocess

The preprocessing consists of repeated unit clause and pure literal propagation and bounded variable elimination (cf. e.g. [4]) until no more pure literals or unit clauses are can be propagated or a sixth of the global timeout is exceeded. Since these operations are only sound for hard clauses, we label the soft clauses, consider them as hard clauses and add an extra soft clause for each label (cf. [3] for details). Then, for an initial solution the SAT solver glucose [2] is called.

III. Perturbation

In the perturbation, we aim at altering the solution in order to explore the search space even though the solution might worsen. More precisely, we flip the sign of a variable picked uniformly at random in unsatisfied clauses, selected uniformly at random in the set of unsatisfied clauses. We do not pick clauses or variables twice in this procedure. In addition, in every n-th perturbation call, the selected unsatisfied clauses are considered hard clauses and given to the SAT-solver glucose that forces a solution where these clauses are satisfied. Glucose has a time limit of 5 seconds for this. A subset of unsatisfied clauses is found by first sampling a random number k from the geometric distribution with parameter p_1 and then selecting k clauses from the set of unsatisfied clauses. As most of the formula's clauses remain satisfied during many iterations, we keep and update a superset of the unsatisfied clauses during the whole algorithm to speed up the sampling from this superset instead collecting all unsatisfied clauses in each perturbation step. Clauses that become unsatisfied in the perturbation step are added to a set of unsatisfied *candidate* clauses. Finally, the perturbation has a plateau search where, again, a random number k is sampled from the geometric distribution with parameter p_2. Then, k variables are picked uniformly at random and their signs are flipped if no clause becomes unsatisfied by the flip.

IV. Improvements

The improvement part is the core of SR and works in the following way. Starting from a random (unsatisfied) candidate clause, SR flips the sign of a randomly chosen variable from a set of currently unsatisfied clauses. If this flip causes other clauses to become unsatisfied, they are added to this set. If it leads to an over-satisfaction of a clause, the algorithm attempts to flip yet another of this over-satisfied clause's variables' signs if no further clauses become unsatisfied by that second flip. We keep a vector containing the number of true literals for each clause, denoted the clause's *stability*, for this step and perform it whenever the stability of a clause grows from 1 to 2. Moreover, we do not flip a variable's sign twice during a single improvement step. After no further clauses in the set of currently unsatisfied clauses can be satisfied, or an iteration limit of l is reached, the improvement for the clause ends and we carry on with the next candidate clause. After all candidate clauses have been tried to improve, the improvement part ends.

V. Solution Checking

If we see the best solution found so far, we save it and continue. However, it is possible that the improvements could not compensate the solution's worsening of the perturbation step. Nevertheless, with a probability that decreases exponentially in the time elapsed, we accept that worse solution because it might help to leave local optima. This technique is known as

Flag	Description	Type
-z	global timeout	int
-o	file path the solution output	string
-innermaxit	l	int
-maxstepsworse	m	int
-geom	p_1	int
-plateau	p_2	int
-maxstepsperturbtosat	n	int

simulated annealing [5]. However, after m iterations without an update of the best solution, we restore the best solution.

VI. Implementation and Usage

The implementation of SR is in C++. The various parameters within the algorithm are set at the beginning and according to selected features of the instance at hand, such as the number of soft clauses or their average weight. If desired, the parameters can be set manually when calling the solver with the flag *-autoparam*. This disables the automatic setting of parameters and all parameters listed in Table VI can be set.

References

[1] D. V. Andrade, M. G. C. Resende, R. F. F. Werneck, "Fast local search for the maximum independent set problem", in J. Heuristics, 18(4), 2012

[2] G. Audemard, L. Simon, "Predicting learnt clauses quality in modern sat solvers", in Proceedings of the 21st International Jont Conference on Artifical Intelligence (IJCAI'09), San Francisco, CA, USA, 2009

[3] A. Belov, A. Morgado, J. Marques-Silva, "Sat-based preprocessing for maxsat", in McMillan, K., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning. pp. 96–111, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013

[4] M. Davis, H. Putnam, "A computing procedure for quantification theory", in J. ACM7(3), pp. 201–215, 1960

[5] M. Pincus, "A Monte-Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems", in Operation Research, 18(6), 1970

[6] J. Reisch, P. Großmann, N. Kliewer, "Stable Resolving - A Randomized Local Search Heuristic for MaxSAT", unpublished paper under review, 2020

[7] J. Reisch, P. Großmann, N. Kliewer, "Conflict Resolving - A Maximum Independent Set Heuristics for Solving MaxSAT", in Proceedings of the 22nd International Multiconference Information Society, 1, pp. 67-71, 2019

MaxHS in the 2020 MaxSat Evaluation

Fahiem Bacchus
Department of Computer Science
University of Toronto
Ontario, Canada
Email: fbacchus@cs.toronto.edu

1. MaxHS

MaxHS is a MaxSat solver originally developed in [6]. It was the first MaxSat solver to utilize the Implicit Hitting Set (IHS) approach, and its core components are described in [6], [4], [5], [7]. Additional useful insights into IHS are provided in [8], [9]. IHS solvers utilize both an integer programming (IP) solver and a SAT solver in a hybrid approach to MaxSat solving. This separation also allows additional techniques from IP solving to be exploited in the solver [1].

MaxHS utilizes MiniSat v2.2 as its SAT solver and IBM's CPLEX v12.8 as its IP solver. Interestingly experiments with more sophisticated SAT solvers like Glucose http://www.labri.fr/perso/lsimon/glucose/ and Lingeling http://fmv.jku.at/lingeling/ yielded inferior performance. This indicates that the SAT problems being solved are quite simple, too simple for the more sophisticated techniques used in these SAT solvers to pay off. In fact, simpler SAT problems are one of the original motivations behind MaxHS [4].

Nevertheless, some changes have been made to the underlying MiniSat solver. In particular, this year the MiniSat code base was updated to maintain LBD clause scores and to perform clause deletion using LBD scores in a manner identical to the Glucose solver. However, this is the only part of the Glucose that was added.

The 2020 version of MaxHS saw a large change to implement abstract cores [2]. The notion of abstract cores is described in the cited paper; their benefit is that the solver can generate a single abstract core that represents an exponential number of non-abstract cores. With abstract cores the IHS approach avoids having to generate an exponential number of cores in some cases [6].

Also a version for the Top-K track was submitted.

A summary of the main features of MaxHS that extend the basic IHS approach are listed below. Most of these features are unchanged from 2019.

1.0.1. Abstract Cores. In MaxHS cores are sets of soft clauses that jointly cannot be satisfied in any feasible solution (i.e., a solution satisfying the hard clauses). A cores is represented in the solver by a clause containing the soft clause's blocking variables. The IHS approach is to use a SAT solver to collect cores and an IP solver to compute a minimum cost set of soft clauses that can satisfy all of the core constraints (i.e., each core specifies the constraint that at least one soft clause in each core must be falsified). MaxHS keep track of the computed cores and uses the Louvain clustering algorithm [3] to find clusters of soft clauses that frequently appear together in cores. These clusters of soft clauses are then grouped together as inputs to a totalizer. Now instead of generating cores by assuming individual soft clauses to be true, totalizer outputs are assumed that limit the number of soft clauses in a cluster that can be falsified. Thus the specific identity of the soft clauses to be falsified is ignored—all that matters is how many are falsified. MaxHS only clusters the soft clauses when it is failing to make progress. Furthermore, it continues to generate ordinary non-abstract cores as well as abstract cores.

1.0.2. Top-K. In the Top-K track the solver is to return a sequence of non-decreasing cost optimal solutions each of which falsifies a different set of soft clauses. To achieve this the existing code base was modified—later these changes can be added as an option. The following changes were required

- A preprocessing time pure literal detection was removed. Literals that appear in only one polarity in both the hard *and the soft* clauses of the input formula can be set to true without affecting the optimal solution. However, subsequent top-k solutions might require negating a pure literal. So pure literal reduction was turned off in the top-k version.
- Boolean Lexicographic Optimization (BLO) was turned off. This also is a preprocessing step. Let S be the set of soft clauses and H be the set of hard clauses of the instance. In BLO if a set $B \subseteq S$ of soft clauses can be found such that (a) $B \cup H$ is satisfiable and (b) the sum of weights of the remain soft clauses (i.e., those in $S - B$) is less than the minimum weight soft clause in B, then all clauses in B can be made hard. That is, BLO detects the case where all of the clauses in B can be satisfied and it is better to falsify all clauses not in B rather than falsify a single clause in B. MaxHS performs BLO hardening of soft clauses as a preprocessing step, but for top-k this was turned off.

- LP reduced cost hardening [1] used in MaxHS was turned off. This technique relies on an upper bound on the solution cost to harden soft clauses, however with top-k when looking for the next solution the upper bound can increase making such hardening invalid.

Other than these changes the implementation of top-k is simple: simply add a blocking clause computed from the previous found solution and then reoptimize with this added constraint. This blocking clause specifies that at least one of the previously satisfied soft clauses must now be falsified, forcing the solver to find an new optimal solution falsifying a different set of soft clauses from before as required. (The new solution is an optimal solution of the now more constrained formula, it is not usually an optimal solution of the original formula.)

1.0.3. Termination based on Bounding. MaxHS maintains an upper bound (and best model found so far) and a lower bound on the cost of an optimal solution (the IP solver computes valid lower bounds). MaxHS terminates when the gap between the lower bound and upper bound is low enough (with integer weights when this gap is less than 1, the upper bound model is optimal). This means that MaxHS no longer needs to wait until the IP solver returns an hitting set whose removal from the set of soft clauses yields SAT; it can return when the IP solver's best lower bound is close enough to show that the best model is optimal.

1.0.4. Early Termination of Cplex. In early versions of MaxHS, the IP solver was run to completion forcing it to find an optimal solution every time it is called. However, with bounding, optimal solutions are not always needed. In particular, if the IP solver finds a feasible solution whose cost is better than the current best model it can return that: either the IP solution is feasible for the MaxSat problem, in which case we can lower the upper bound, or it is infeasible in which case we can obtain additional cores to augment the IP model (and thus increase the lower bound). Terminating the IP solver before optimization is complete can yield significant time savings.

1.0.5. Reduced Cost fixing via the LP-Relaxation. Using an LP relaxation and the reduced costs associated with the optimal LP solution, some soft clauses can be hardened or immediately falsified. See [1] for more details.

1.0.6. Mutually Exclusive Soft Clauses. Sets of soft clauses of which at most one can be falsified or at most one can be satisfied are detected. Sets where at most one soft clause can be satisfied are re-encoded to a new MaxSat problem with one soft clause to represent condition that one of these previous soft clauses is satisfied. With abstract cores it was found that sets where at most one soft clause can be falsified were no longer helpful, and these are not used in this version.

1.0.7. Other clauses to the IP Solver. Problems with a small number of variables are given entirely to the IP solver, so that it directly solves the MaxSat problem. In this case the SAT solver is used to first compute some additional clauses and cores, and to find a better initial model for the IP solver. This additional information from the SAT solver often makes the IP solver much faster than just running the IP solver and represents an alternate way of hybridizing SAT and IP solvers.

1.0.8. Other techniques for finding Cores. MaxHS iteratively calls the IP solver to obtain a hitting set of the cores computed so far. If that hitting set does not yield an optimal MaxSat solution then more cores must be added to the IP solver. In some of these iterations very few cores can be found causing only a slight improvement to the IP solver's model. This results in a large number of time consuming calls to the IP solver. Two method were developed to aid this situation (a) we ask the IP solver for more solutions and generate cores from these as hitting sets as well and (b) if we have a new upper bound model we try to improve this model by converting it to a minimal correction set (MCS). In converting the upper bound model to an MCS we either find a better model (lowering the upper bound) or we compute additional conflicts that can be added to the IP solver.

References

[1] Bacchus, F., Hyttinen, A., Järvisalo, M., Saikko, P.: Reduced cost fixing in maxsat. In: Beck, J.C. (ed.) Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10416, pp. 641–651. Springer (2017), https://doi.org/10.1007/978-3-319-66158-2_41

[2] Berg, J., Bacchus, F., Poole, A.: Abstract Cores in Implicit Hitting Set MaxSat solving. In: Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, SAT 2019 Proceedings. Lecture Notes in Computer Science, Springer (2020), in press

[3] Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10), P10008 (Oct 2008), https://doi.org/10.1088%2F1742-5468%2F2008%2F10%2Fp10008

[4] Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Proc. CP. Lecture Notes in Computer Science, vol. 6876, pp. 225–239. Springer (2011)

[5] Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MaxSAT. In: Proc. SAT. Lecture Notes in Computer Science, vol. 7962, pp. 166–181. Springer (2013)

[6] Davies, J.: Solving MAXSAT by Decoupling Optimization and Satisfaction. Ph.D. thesis, University of Toronto (2013), http://www.cs.toronto.edu/~jdavies/Davies_Jessica_E_201311_PhD_thesis.pdf

[7] Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: Proc. CP. Lecture Notes in Computer Science, vol. 8124, pp. 247–262. Springer (2013)

[8] Saikko, P., Berg, J., Järvisalo, M.: LMHS: A SAT-IP hybrid MaxSAT solver. In: Proc. SAT. Lecture Notes in Computer Science, vol. 9710, pp. 539–546. Springer (2016)

[9] Saikko, P.: Re-implementing and Extending a Hybrid SAT-IP Approach to Maximum Satisfiability. Master's thesis, University of Helsinki (2015), http://hdl.handle.net/10138/159186

Maxino

Mario Alviano
Department of Mathematics and Computer Science
University of Calabria
87036 Rende (CS), Italy
ORCID: 0000-0002-2052-2063

***Abstract*—Maxino is based on the k-ProcessCore algorithm, a parametric algorithm generalizing OLL, ONE and PMRES. Parameter k is dynamically determined for each processed unsatisfiable core by a function taking into account the size of the core. Roughly, k is in $O(\log n)$, where n is the size of the core. Satisfiability of propositional theories is checked by means of a pseudo-boolean solver extending Glucose 4.1 (single thread). Top-k is implemented by disabling hardening and using blocking clauses involving the soft literals associated with soft clauses in the input formula.**

***Index Terms*—component, formatting, style, styling, insert**

A Very Short Description of the Solver

The solver MAXINO is build on top of the SAT solver GLUCOSE [7] (version 4.1). MaxSAT instances are normalized by replacing non-unary soft clauses with fresh variables, a process known as *relaxation*. Specifically, the relaxation of a soft clause ϕ is the clause $\phi \vee \neg x$, where x is a variable not occurring elsewhere; moreover, the weight associated with clause ϕ is associated with the soft literal x. Hence, the normalized input processed by MAXINO comprises hard clauses and soft literals, so that the computational problem amounts to maximize a linear function, which is defined by the soft literals, subject to a set of constraints, which is the set of hard clauses.

The algorithm implemented by MAXINO to address such a computational problem is based on unsatisfiable core analysis, and in particular takes advantage of the following *invariant*: A model of the constraints that satisfies all soft literals is an optimum model. The algorithm then starts by searching such a model. On the other hand, if an inconsistency arises, the unsatisfiable core returned by the SAT solver is analyzed. The analysis of an unsatisfiable core results into new constraints and new soft literals, which replace the soft literals involved in the unsatisfiable core. The new constraints are essentially such that models satisfying all new soft literals actually satisfy all but one of the replaced soft literals. Since there is no model that satisfies all replaced soft literals, it turns out that the invariant is preserved, and the process can be iterated.

Specifically, the algorithm implemented by MAXINO is K, based on the k-ProcessCore procedure introduced by Alviano et al. [2]. It is a parametric algorithm generalizing OLL [3], ONE [2] and PMRES [8]. Intuitively, for an unsatisfiable core $\{x_0, x_1, x_2, x_3\}$, ONE introduces the following constraint:

$$x_0 + x_1 + x_2 + x_3 + \neg y_1 + \neg y_2 + \neg y_3 \geq 3$$
$$y_1 \rightarrow y_2 \quad y_2 \rightarrow y_3$$

where y_1, y_2, y_3 are fresh variables (the new soft literals that replace x_0, x_1, x_2, x_3). OLL introduces the following constraints (the first immediately, the second if a core containing y_1 is subsequently found, and the third if a core containing y_2 is subsequently found):

$$x_0 + x_1 + x_2 + x_3 + \neg y_1 \geq 3$$
$$x_0 + x_1 + x_2 + x_3 + \neg y_2 \geq 2$$
$$x_0 + x_1 + x_2 + x_3 + \neg y_3 \geq 1$$

Concerning PMRES, it introduces the following constraints:

$$x_0 \vee x_1 \vee \neg y_1 \quad z_1 \leftrightarrow x_0 \wedge x_1$$
$$z_1 \vee x_2 \vee \neg y_2 \quad z_2 \leftrightarrow z_1 \wedge x_2$$
$$z_2 \vee x_3 \vee \neg y_3$$

which are essentially equivalent to the following constraints:

$$x_0 + x_1 + \neg z_1 + \neg y_1 \geq 2 \quad z_1 \rightarrow y_1$$
$$z_1 + x_2 + \neg z_2 + \neg y_2 \geq 2 \quad z_2 \rightarrow y_2$$
$$z_2 + x_3 \quad + \neg y_3 \geq 1$$

where y_1, y_2, y_3 are fresh variables (the new soft literals that replace x_0, x_1, x_2, x_3), and z_1, z_2 are fresh auxiliary variables.

Algorithm K, instead, introduces a set of constraints of bounded size, where the bound is given by the chosen parameter k, and is specifically $2 \cdot (k+1)$. ONE, which is essentially a smart encoding of OLL, is the special case for $k = \infty$, and PMRES is the special case for $k = 1$. For the example unsatisfiable core, another possibility is $k = 2$, which would results in the following constraints:

$$x_0 + x_1 + x_2 + \neg z_1 + \neg y_1 + \neg y_2 \geq 3 \quad z_1 \rightarrow y_1 \quad y_1 \rightarrow y_2$$
$$z_1 + x_3 \quad + \neg y_3 \quad \geq 1$$

In this version of MAXINO, the parameter k is dynamically determined based on the size of the analyzed unsatisfiable core: $k \in O(\log n)$, where n is the size of the core.

The analysis of unsatisfiable core is preceded by a *shrink* procedure [1]. Specifically, a reiterated progression search is performed on the unsatisfiable core returned by the SAT solver. Such a procedure significantly reduce the size of the unsatisfiable core, even if it does not necessarily returns an unsatisfiable core of minimal size. Since minimality of the unsatisfiable cores is not a requirement for the Additionally, satisfiability checks performed during the shrinking process are subject to a budget on the number of conflicts, so that the overhead due to hard checks is limited. Specifically, the budget is set to the number of conflicts arose in the satisfiability

check that lead to detecting the unsatisfiable core; if such a number is less than 1000 (one thousand), the budget is raised to 1000. The budget is divided by 2 every time the progression is reiterated.

Weighted instances are handled by *stratification* and introducing *remainders* [4]–[6]. Specifically, soft literals are partitioned in strata depending on the associated weight. Initially, only soft literals of greatest weight are considered, and soft literals in the next stratum are added only after a model satisfying all considered soft literals is found. When an unsatisfiable core is found, the weight of all soft literals in the core is decreased by the weight associated with last added stratum. Soft literals whose weight become zero are not considered soft literals anymore.

Finally, a preprocessing step is performed on unweighted instances, which essentially iterates on all hard clauses of the input theory, sorted by length, and checks whether they already witness some unsatisfiable core. Specifically, an hard clause witnesses an unsatisfiable core if all literals in the clause are the complement of a soft literal. If this is the case, the unsatisfiable core is analyzed immediately. The rationale for such a preprocessing step is that hard clauses in the input theory are often small, and the smaller the better for the unsatisfiable core based algorithms.

REFERENCES

[1] Mario Alviano and Carmine Dodaro. Anytime answer set optimization via unsatisfiable core shrinking. *TPLP*, 16(5-6):533–551, 2016.

[2] Mario Alviano, Carmine Dodaro, and Francesco Ricca. A maxsat algorithm using cardinality constraints of bounded size. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2677–2683. AAAI Press, 2015.

[3] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *28th International Conference on Logic Programming*, pages 211–221, Budapest, Hungary, September 2012.

[4] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *SAT 2009*, pages 427–440, Swansea, UK, June 2009. Springer.

[5] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196(0):77–105, March 2013.

[6] Josep Argelich, Inês Lynce, and João P. Marques Silva. On solving boolean multilevel optimization problems. In *21st International Joint Conference on Artificial Intelligence*, pages 393–398, Pasadena, California, July 2009. IJCAI Organization.

[7] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *21st International Joint Conference on Artificial Intelligence*, pages 399–404, Pasadena, California, July 2009. IJCAI Organization.

[8] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2717–2723, Québec City, Canada, July 2014. AAAI Press.

SATLike-c: Solver Description

1st Zhendong Lei
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
leizd@ios.ac.cn

2nd Shaowei Cai
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
caisw@ios.ac.cn

***Abstract*—In this document, we briefly describe the techniques employed by the *SATLike-c* solver participation in MaxSAT Evaluation 2020.**

I. Introduction

SATLike-c participates in incomplete track. *SATLike-c* has two engines, one is local search solver *SATLike* [1] and the other is SAT-based solver *Loandra* [2]. First, a core-guided SAT solver is executed to find a feasible solution. Then *SATLike* is executed with this feasible solution as its initial solution. *SATLike* keeps working until it fails to improve the current solution in a given time limit. After that, *Loandra* is executed to continue to improve the current solution.

References

[1] Zhendong Lei and Shaowei Cai. "Solving(weighted) partial maxsat by dynamic local search for SAT." InProceedings of the Twenty-Seventh International Joint Confer-ence on Artificial Intelligence, IJCAI 2018, July 13-19, 2018,Stockholm, Sweden. pages 1346–1352, 2018.

[2] Jeremias Berg, Emir Demirovic and Peter J. Stuckey, "Core-Boosted Linear Search for Incomplete MaxSAT" Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings. vol-ume 11494 of Lecture Notes in Computer Science, pages 39–56. Springer, 2019.

Open-WBO @ MaxSAT Evaluation 2020

Ruben Martins
rubenm@cs.cmu.edu
CMU, USA

Norbert Manthey
nmanthey@conp-solutions.com
Dresden, Germany

Miguel Terra-Neves, Vasco Manquinho, Inês Lynce
{neves,vmm,ines}@inesc-id.pt
INESC-ID/IST, Portugal

I. Introduction

Open-WBO [1] is an open source MaxSAT solver that supports several MaxSAT algorithms [2], [3], [4], [5], [6], [7], [8] and SAT solvers [9], [10], [11]. Open-WBO is particularly efficient for unweighted MaxSAT and has been one of the best solvers in the MaxSAT Evaluations from 2014 to 2017. Five versions of Open-WBO were submitted to different tracks at MaxSAT Evaluation 2020: open-wbo-res-mergesat-v1, open-wbo-res-mergesat-v2, open-wbo-res-glucose-v1, open-wbo-res-glucose-v2, and open-wbo-topk. The remainder of this document describes the differences between these versions.

II. SAT solvers

Open-WBO is based on the data structures of MiniSat 2.2 [9], [12]. Therefore, solvers based on MiniSat 2.2 can be used as a potential back-end solver. For the MaxSAT Evaluation 2019, we use Glucose 4.1 [10], [13], [14] as the back-end SAT solver of the versions that end in glucose and MergeSAT [11] as the back-end SAT solver of the versions that end in mergesat.

MergeSAT is a new CDCL solver developed by Norbert Manthey and it is based on the SAT competition winner of 2018, MapleLCMDistChronoBT [15], and adds several known techniques. For restarts, only partial backtracking is used, learned clause minimization is implemented more efficiently, and also applies simplification again in case the first swipe resulted in a simplification. Finally, the time-based decision heuristic switch is made deterministic by using solving steps. Furthermore, subsumption and self-subsuming resolution is used as inprocessing. To support being used inside MaxSAT solvers, the incremental search feature had to been enabled again.

III. MaxSAT Algorithms

In this section, we briefly describe the algorithms used for the complete and top-k tracks at the MSE2020.

A. Complete Unweighted Track

Four versions were submitted to the complete unweighted track: open-wbo-res-mergesat-v1, open-wbo-res-mergesat-v2, open-wbo-res-glucose-v1, open-wbo-res-glucose-v2.

All versions use a variant of the unsatisfiability-based algorithm MSU3 [3] and the OLL algorithm [7]. This algorithm works by iteratively refining a lower bound λ on the number of unsatisfied soft clauses until an optimum solution is found. We use an incremental version of this algorithm by taking advantage of the incremental version of the Totalizer encoding [4]. We also extended the incremental MSU3 algorithm [4] with resolution-based partitioning techniques [8]. We represent a MaxSAT formula using a resolution-based graph representation and iteratively join partitions by using a proximity measure extracted from the graph representation of the formula. The algorithm ends when only one partition remains and the optimal solution is found. Since the partitioning of some MaxSAT formulas may be unfeasible or not significant, we heuristically choose to run either MSU3 with partitions or without partitions. In particular, we do not use partition-based techniques when one of the following criteria is met: (i) the formula is too large ($>$ 1,000,000 clauses), (ii) the ratio between the number of partitions and soft clauses is too high ($>$ 0.8), (iii) the sparsity of the graph is too small ($<$ 0.04), or (iv) there exist some at-most-one relations between soft clauses ($>$ 10), i.e. if one soft clause is satisfied it implies that some other soft clauses will be unsatisfied.

The difference between versions `v1` and `v2` is that version `v1` uses the MSU3 algorithm when the partitioning algorithm is not applicable, whereas `v2` uses the OLL algorithm.

B. Top-k Track

For the top-k track, we use a linear search SAT-UNSAT to find the optimal solution. Once the optimal solution is found, we change to the (W)MSU3 algorithm and enumerate solutions until we exhaust the search space or we find k solutions. Each algorithm is incremental but the swap is made in a non-incremental fashion. For the unweighted track, we use the incremental Totalizer encoding for MSU3 [4] and the incremental SWC encoding for WMSU3 [16].

These algorithms are not optimized for the top-k track and should be consider a baseline for future improvements.

C. Preprocessing

We perform identification of unit cores and at-most-one relations between soft clauses by using unit propagation. A similar technique is done in RC2 [17], the winner of the MaxSAT Evaluation 2018.

D. Other

Open-WBO now supports printing the certificate in a compact mode using 0's and 1's.

IV. Availability

The latest release of Open-WBO is available under a MIT license in GitHub at *https://github.com/sat-group/open-wbo*.

Acknowledgments

We would like to thank Laurent Simon and Gilles Audemard for allowing us to use Glucose 4.1 in the MaxSAT Evaluation. We would also like to thank Niklas Eén and Niklas Sörensson for the development of MiniSat 2.2. Additionally, we would like to thank all the collaborators on previous versions of Open-WBO, namely Saurabh Joshi and Mikoláš Janota. Finally, we would like to thank David Chen for his study on the impact of disjoint cores, unit cores, and at-most-one relations between soft clauses that were done in the scope of Independent Studies at CMU.

References

[1] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a Modular MaxSAT Solver," in *SAT*, ser. LNCS, vol. 8561. Springer, 2014, pp. 438–445.

[2] V. Manquinho, J. Marques-Silva, and J. Planes, "Algorithms for Weighted Boolean Optimization," in *SAT*. Springer, 2009, pp. 495–508.

[3] J. Marques-Silva and J. Planes, "On Using Unsatisfiability for Solving Maximum Satisfiability," *CoRR*, 2007.

[4] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental Cardinality Constraints for MaxSAT," in *CP*. Springer, 2014, pp. 531–548.

[5] R. Martins, V. Manquinho, and I. Lynce, "On Partitioning for Maximum Satisfiability," in *ECAI*. IOS Press, 2012, pp. 913–914.

[6] R. Martins, V. M. Manquinho, and I. Lynce, "Community-based partitioning for maxsat solving," in *SAT*. Springer, 2013, pp. 182–191.

[7] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-Guided MaxSAT with Soft Cardinality Constraints," in *CP*. Springer, 2014, pp. 564–573.

[8] M. Neves, R. Martins, M. Janota, I. Lynce, and V. M. Manquinho, "Exploiting Resolution-Based Representations for MaxSAT Solving," in *SAT*. Springer, 2015, pp. 272–286.

[9] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*. Springer, 2003, pp. 502–518.

[10] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI*, 2009, pp. 399–404.

[11] N. Manthey, "Mergesat," in *Proceedings of SAT Competition 2019: Solver and Benchmark Descriptions*, 2019.

[12] N. Sörensson, N. Een, and N. Manthey. (2018, May) GitHub repository for MiniSat. https://github.com/conp-solutions/minisat.

[13] G. Audemard, J.-M. Lagniez, and L. Simon, "Improving glucose for incremental sat solving with assumptions: Application to mus extraction," in *SAT*. Springer, 2013.

[14] G. Audemard and L. Simon. (2018, May) Glucose's home page. http://www.labri.fr/perso/lsimon/glucose.

[15] A. Nadel and V. Ryvchin, "Chronological backtracking," in *SAT*. Springer, 2018, pp. 111–121.

[16] R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce, "On using incremental encodings in unsatisfiability-based maxsat solving," *JSAT*, vol. 9, no. 1, pp. 59–81, 2014.

[17] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python Toolkit for Prototyping with SAT Oracles," in *Proc. SAT*, ser. Lecture Notes in Computer Science, O. Beyersdorff and C. M. Wintersteiger, Eds., vol. 10929. Springer, 2018, pp. 428–437.

Open-WBO-Inc in MaxSAT Evaluation 2020

Saurabh Joshi, Prateek Kumar
{sbjoshi,cs15btech11031}@iith.ac.in
Indian Institute of Technology Hyderabad, India

Sukrut Rao
sukrut.rao@mpi-inf.mpg.de
Max Planck Institute for Informatics, Germany

Ruben Martins
rubenm@cs.cmu.edu
CMU, USA

I. Introduction

Open-WBO-Inc [1], [2] is developed on top of Open-WBO [3], [4], [5] and placed first and second on the weighted incomplete tracks for 60 and 300 seconds respectively in the MaxSAT Evaluation 2018, and third on both these tracks in the MaxSAT Evaluation 2019. For many applications that can be encoded into MaxSAT, it is important to quickly find solutions even though these may not be optimal. Open-WBO-Inc is designed to find a good solution[1] in a short amount of time. Since Open-WBO-Inc is based on Open-WBO, it can use any MiniSAT-like solver [6]. For this evaluation, we use Glucose 4.1 [7] as our back-end SAT solver.

II. Algorithms

For the MaxSAT Evaluation 2020, we restrict Open-WBO-Inc to the weighted category where it uses the novel approximation algorithms that have been recently proposed [1], [2]. In particular, we submitted three versions of Open-WBO-Inc: inc-bmo-complete, inc-bmo-satlike, and inc-bmo-satlike19.

All versions are based on bounded multilevel optimization [8] using a variant of linear search algorithm SAT-UNSAT [9]. The algorithms used in these versions consider n objective functions where n is the number of different weights in the MaxSAT instance. This is done by performing a sequence of calls to a SAT solver and refining an upper bound μ on the number of unsatisfied soft clauses. To restrict μ at each iteration, we need to encode cardinality constraints into CNF, for which incremental Totalizer encoding [4] has been used. Once the upper bound μ for a given objective function cannot be improved, it is frozen, and the next objective function in the order is optimized.

An optimal solution, if found when using this algorithm, is not necessarily an optimal solution for the input formula. inc-bmo-complete and inc-bmo-satlike versions differ between them when this occurs. inc-bmo-complete keeps the best-known solution and resumes the search using the LSU algorithm which can potentially find better solutions and prove optimality. In contrast, inc-bmo-satlike changes the search algorithm to SATLike [10], a MaxSAT stochastic algorithm. The best model found by the first phase is passed to SATLike as its initial starting model.

[1]By "good solution" we mean that it can be potentially suboptimal but is not far from the optimal solution.

The inc-bmo-satlike19 version corresponds to the best performing version of Open-WBO-Inc in the MaxSAT Evaluation 2019. For the versions of this year, we added a conflict limit of 10^7 on each SAT call when performing the multilevel optimization phase. This prevents the solver from being stuck in some optimization level and never entering the final phase. We have also included the Target-Optimum-Rest-Conservative (TORC) and Target-Score-Bum (TSB) heuristics [11]. The TORC heuristic changes the default polarity of the SAT solver to take into consideration the MaxSAT formula. Relaxation variables that may appear in the cardinality constraints of the multilevel optimization algorithm are always set to polarity *false*. For the remaining variables, the polarity is set according to the best model found during search. The TSB heuristic bumps the score of all relaxation variables to make them more likely to be picked at the beginning of the search. Additionally, we also now support printing a compact certificate using 0's and 1's instead of variable ids.

III. Availability

We submit the source of Open-WBO-Inc as part of our submissions to the MaxSAT Evaluations 2020. The inc-bmo-complete version and the full Open-WBO-Inc framework is available under a MIT license in GitHub at *https://github.com/sbjoshi/Open-WBO-Inc*.

Acknowledgments

We would like to thank Laurent Simon and Gilles Audemard for allowing us to use Glucose in the MaxSAT Evaluation. We would also like to thank Vasco Manquinho, Inês Lynce, Mikoláš Janota, Miguel Terra-Neves and Norbert Manthey for their contributions to Open-WBO on which Open-WBO-Inc is based.

References

[1] S. Joshi, P. Kumar, R. Martins, and S. Rao, "Approximation Strategies for Incomplete MaxSAT," in *CP*. Springer, 2018.
[2] S. Joshi, P. Kumar, S. Rao, and R. Martins, "Open-WBO-Inc: Approximation Strategies for Incomplete Weighted MaxSAT," in *Journal on Satisfiability, Boolean Modeling and Computation*. IOS Press, 2019.
[3] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a Modular MaxSAT Solver," in *SAT*, ser. LNCS, vol. 8561. Springer, 2014, pp. 438–445.
[4] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental Cardinality Constraints for MaxSAT," in *CP*. Springer, 2014, pp. 531–548.
[5] M. Neves, R. Martins, M. Janota, I. Lynce, and V. Manquinho, "Exploiting Resolution-Based Representations for MaxSAT Solving," in *SAT*. Springer, 2015, pp. 272–286.
[6] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*. Springer, 2003, pp. 502–518.

[7] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI*, 2009, pp. 399–404.
[8] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3-4, pp. 317–343, 2011.
[9] D. Le Berre and A. Parrain, "The Sat4j library, release 2.2," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, no. 2-3, pp. 59–6, 2010.
[10] Z. Lei and S. Cai, "Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT," in *IJCAI*. ijcai.org, 2018, pp. 1346–1352.
[11] A. Nadel, "Anytime weighted maxsat with improved polarity selection and bit-vector optimization," in *Proc. Formal Methods in Computer Aided Design*, C. W. Barrett and J. Yang, Eds. IEEE, 2019, pp. 193–202.

`sls-mcs` and `sls-lsu`: Description

Andreia P. Guerreiro[1], Miguel Terra-Neves[1,2], Inês Lynce[1], José Rui Figueira[3], and Vasco Manquinho[1]

[1]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
{andreia, ines, vmm}@sat.inesc-id.pt
[2]OutSystems, Portugal
miguel.neves@outsystems.com
[3]CEG-IST, Instituto Superior Técnico, Universidade de Lisboa, Portugal
figueira@tecnico.ulisboa.pt

1 Introduction

We developed improved versions of `sls-mcs` and `sls-lsu`, two solvers that integrate SAT-based techniques in a Stochastic Local Search (SLS) solver for MaxSAT. In these solvers, the control of the solving process changes from SAT-based procedures to stochastic procedures and vice-versa. At each step, each procedure tries to build upon the information received from the other, instead of being independent procedures. The idea is to use the SLS solver as the main procedure and, occasionally, use an unsatisfiability-based algorithm to correct the SLS current (unsatisfiable) assignment into a satisfiable one, and use a procedure based on Minimal Correction Subset (MCS) enumeration or on the Linear Sat-Unsat (LSU) algorithm to improve the current solution. We submitted new versions of `sls-mcs` and `sls-lsu` for the unweighted incomplete MaxSAT track, and two new versions of `sls-mcs` for the weighted incomplete MaxSAT track.

2 Using SAT Techniques in Local Search

One of the shortcomings of SLS algorithms is that these solvers have difficulties in dealing with highly constrained formulas. Therefore, it might be the case that the SLS algorithm is unable to satisfy the set of hard clauses or gets stuck in some local minima. In these cases, using SAT-based techniques to find a satisfiable assignment would be beneficial.

2.1 Assignment Correction

Consider the case when the SLS algorithm is unable to change from an unsatisfiable assignment ν into a better assignment. Our solver performs a correction to ν in order to guide the SLS algorithm to the feasible region of the search space. First, we start by building a set of assumption literals corresponding to the assignment ν. Next, a SAT call on the set of hard clauses, ϕ_h, is made. Clearly, if ν is not feasible, then this call returns `UNSAT` and returns an unsatisfiable core. The assumption literals that occur in such an unsatisfiable core are removed from the set of assumptions, and a new SAT call is made. The same procedure is repeated until a satisfiable assignment is found.

A conflict limit is defined for the correction procedure. If the conflict budget is not enough to find a satisfiable assignment, then our algorithm applies a similar procedure with a more aggressive strategy where at each iteration 50% of the literals in the set of assumptions are removed. Since the correction procedure only depends on the hard clauses, there is no guarantee regarding its quality. As a result, we also apply a SAT-based improvement procedure.

2.2 Assignment Improvement

Given a MaxSAT instance ϕ, a set of assumptions $\mathcal{A}$, a satisfiable assignment ν, and conflict budget, the goal of this assignment improvement algorithm is to find a better quality solution for ϕ through an MCS enumeration procedure.

The algorithm starts by building a working formula from the set of hard clauses ϕ_h and the set of assumptions $\mathcal{A}$. Next, the algorithm iterates over all MCSes of ϕ, constrained to the set of assumptions $\mathcal{A}$ and returns the best assignment found. Each time a new MCS is found, a blocking clause is added to prevent the enumeration of the same MCS later on. The algorithm returns the best solution found before the conflict budget runs out. Note that the set of literals $\mathcal{A}$ restricts the MCS enumeration procedure. This results in a localized MCS enumeration.

Many different improvement procedures can be devised, including the usage of complete methods. For example, an alternative is to replace the MCS enumeration algorithm by a call to a Linear Sat-

Unsat algorithm (LSU). The call to the LSU algorithm is also limited to a number of conflicts, and all literals in $\mathcal{A}$ are forced to be satisfied. Hence, the LSU call is also restricted to a localized region of the search space.

3 Incomplete Track

We developed the solvers `sls-mcs` and `sls-lsu`, that integrate an SLS algorithm with the assignment correction and assignment improvement procedures [1], which we submitted to MaxSAT Evaluation 2019. As SATLike [3], an SLS algorithm, was one of the best performing solvers in the incomplete solver track in the MaxSAT Evaluation 2018, we used SATLike in our implementation [1]. We developed and submitted improved versions of `sls-mcs` and `sls-lsu`. The main difference to the versions proposed in [1] and submitted to MaxSAT Evaluation 2019 is in the first call to the assignment correction algorithm. If no satisfiable assignment was yet found by the SLS algorithm, then the set of assumptions $\mathcal{A}$ is empty in the first SAT call.

3.1 Unweighted Instances

Two solvers were submitted for the unweighted incomplete track: `sls-mcs` and `sls-lsu`. In `sls-mcs`, the `SATLike` solver[1] is extended with the assignment correction algorithm and the assignment improvement algorithm based on MCS enumeration. The difference from `sls-mcs` to `sls-lsu` is on the assignment improvement algorithm. In `sls-lsu`, the linear sat-unsat assignment improvement algorithm is used.

Both `sls-mcs` and `sls-lsu` use the Glucose SAT solver (version 4.1) on the assignment correction procedure. Moreover, the CLD [4] algorithm is used as the MCS algorithm in `sls-mcs`. The linear sat-unsat algorithm used in `sls-lsu` is an adapted version of the one available at the `open-wbo` open source MaxSAT solver. The conflict limits of the correction and the improvement algorithms were set to 10^5. In both `sls-mcs` and `sls-lsu`, the assignment correction/improvement algorithm is called when SATlike has reached half of the maximum number of iterations without improvement. In such a case, the correction algorithm is called if the current assignment ν does not satisfy all hard clauses, otherwise the improvement algorithm is directly called with approximately half of the literals in the current assignment ν as assumptions. These assumption literals are randomly chosen from ν.

[1] The source code of `SATLike` is publicly available at the 2018 MaxSAT evaluation `https://maxsat-evaluations.github.io/2018/descriptions.html`

3.2 Weighted Instances

Two versions of the solver were submitted for the weighted incomplete track: `sls-mcs` and `sls-mcs2`. In both versions, the stratified CLD algorithm [5] is used as the MCS algorithm. Unlike `sls-mcs`, `sls-mcs2` does not consider the assumptions $\mathcal{A}$ as hard clauses in the MCS enumeration procedure.

Acknowledgments

We would like to thank Zhendong Lei and Shaowei Cai for making the code of SATLike available online. This work was supported by national funds through FCT under projects UIDB/50021/2020 and PTDC/CCI-COM/31198/2017.

References

[1] Guerreiro, A.P., Terra-Neves, M., Lynce, I., Figueira, J.R., Manquinho, V.: Constraint-based techniques in stochastic local search maxsat solving. In: Schiex, T., de Givry, S. (eds.) Principles and Practice of Constraint Programming. pp. 232–250. Springer International Publishing (2019)

[2] Lang, J. (ed.): Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. ijcai.org (2018)

[3] Lei, Z., Cai, S.: Solving (weighted) partial maxsat by dynamic local search for SAT. In: Lang [2], pp. 1346–1352

[4] Marques-Silva, J., Heras, F., Janota, M., Previti, A., Belov, A.: On Computing Minimal Correction Subsets. In: International Joint Conference on Artificial Intelligence. pp. 615–622 (2013)

[5] Terra-Neves, M., Lynce, I., Manquinho, V.M.: Stratification for constraint-based multi-objective combinatorial optimization. In: Lang [2], pp. 1376–1382

SMAX – Implementing a Robust MaxSAT Interface

Norbert Manthey
nmanthey@conp-solutions.com
Dresden, Germany

I. Introduction

The main aim of the MaxSAT solver SMAX is to showcase how a MaxSAT solver can be used as a library. Consequently, it can be understood as a prototype implementation which does not focus on solver performance as the highest priority. The solver has to following properties:

- implement a C++interface as a library
- be easily consumable, by being available as MIT license
- implement in a robust, modular way
- support reproducable partial solving, based on solving steps.

The implementation allows to provide the solver as a shared library, which opens the door for integration into tools that require a Python or Java interface. Consequently, solver failures and exceptions are caught by the library itself, and errors are handled accordingly. All code is version controlled with git submodules, to allow reproduction of solver binaries or libraries. Continuous testing is used to make sure that updated code can be used as solving backend via a shared library, and furthermore checks for errors via valgrind. Finally, a Maxsat fuzzer[1] is used to test whether random input behaves as expected.

II. Solver Internals

The idea behind the solver is open to switch solving backends. The initial implementation follows an approach that makes software licenses simple, by focussing on software that is available as MIT license.

A. Solver Interface

The solver implements a C++interface to a MaxSAT engine. The interface is scatched in Figure 1. The interface aims at allowing failure inside the solver, as well as providing an assignment to start from and a number of steps to perform, so that partial and reproducible solver calls are possible. Discussions on the interface are welcome.

B. MaxSAT Solvers

The solver uses Open-WBO [1] as the current MaxSAT backend, which is an open source MaxSAT solver that supports several MaxSAT algorithms and SAT solvers [2], [3], [4]. To make sure all used code is available under MIT license, the Glucose 4.1 code of the Open-WBO package had to be removed.

[1]The fuzzer is available at https://github.com/conp-solutions/maxsat-fuzzer.git.

The used Open-WBO version is an older version, as initial changes have been added early and rebasing them to a current version has not been considered yet. Consequently, eventual bug fixes might be currently missing in SMAX. Furthermore, additional patches had to be added to Open-WBO to allow pragmatic access to internal data structures like the found bound or the last model to be handled internally. Some part of SMAX basically provides an input parser and an output generator based on the available data structures of Open-WBO, with the additional abstraction layer in between.

We currently use a single, predefined, configuration of Open-WBO to simplify the interface to the MaxSAT library.

C. SAT Solvers

Open-WBO is based on the data structures of MiniSat 2.2 [2], [5]. Therefore, solvers based on MiniSat 2.2 can be used as a potential back-end solvers.

MergeSAT [4] is a CDCL solver based on the SAT competition winner of 2018, MapleLCMDistChronoBT [6]. While being based on MiniSat 2.2, MergeSAT aims at providing recent solving techniques while being compatible with the solver API of MiniSat 2.2. Among others, MergeSAT implements partial backtracking, an efficient version of learned clause minimization, as well as inprocessing based on subsumption and self-subsuming resolution. Comparesed to MapleLCMDistChronoBT, the incremental search feature has to been enabled again.

D. Competition Tracks

The solver has been submitted to the complete tracks of the competition only, although the implementation supports partial solving. However, the current implementation does not support to forward a signal from the solver wrapper into the solver to obtain the currently best known solution. Furthermore, support for Top-K has not been integrated yet, also because the used Open-WBO version does not support this feature.

Two variants of the solver have been submitted. The only difference is the SAT backend, namely MiniSat 2.2 and MergeSAT. The reason to submit these two solvers is to show the performance difference when using a more recent SAT backend.

III. Availability

The solver SMAX is available under a MIT license in GitHub at https://github.com/conp-solutions/smax. The repository is setup to briefly check new changes with continuous integration.

```
class MaxSATSolver

    /** Return codes for the caller to compute a MaxSAT solution */
    enum ReturnCode

        UNKNOWN = 0,
        SATISFIABLE = 1,
        UNSATISFIABLE = 2,
        OPTIMAL = 3,
        ERROR = 4,
    ;

    /** This integer represents the version of the MaxSAT interface */
    unsigned getVersion () const;

    /** This string contains the name of the used backend */
    const char* getSolverName () const;

    /** Initialize the MaxSAT solver for a given formula */
    MaxSATSolver(int nVars, int nClausesEstimate = 8192);

    /** A call to this method frees all resources of the solver. */
    ~MaxSATSolver();

    /** Return error number code in case the last call to another method failed */
    int getErrno() const;

    /** Add a clause to the solver */
    bool addClause(const std::vector<int> &literals, uint64_t weight = 0);

    /** Add an at-most-k constraint to the solver */
    bool addAtMostK(const std::vector<int> &literals, const unsigned k);

    /** Compute a MaxSAT solution for the added (weighted) formula */
    ReturnCode compute_maxsat(std::vector<int> &model,
                              uint64_t &cost,
                              uint64_t maxCost = UINT64_MAX,
                              const std::vector<int> *startAssignment = 0,
                              int64_t maxMinimizeSteps = -1);
```

Fig. 1. This figure briefly summaries the interface that is offered to the MaxSAT solver backend implementation. A well documented version of this file can be found at https://github.com/conp-solutions/smax/blob/master/include/MaxSATSolver.h.

ACKNOWLEDGMENTS

The current solver uses OPEN-WBO as a backend engine, and is heavily based on that solver. Hence, the contibutors of that solver own a large part of this tool: Ruben Martins, as well as Miguel Terra-Neves, Vasco Manquinho and Inês Lynce. Additionally, we would like to thank all the collaborators on previous versions of OPEN-WBO, namely Saurabh Joshi and Mikoláš Janota.

REFERENCES

[1] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a Modular MaxSAT Solver," in *SAT*, ser. LNCS, vol. 8561. Springer, 2014, pp. 438–445.

[2] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*. Springer, 2003, pp. 502–518.

[3] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI*, 2009, pp. 399–404.

[4] N. Manthey, "Mergesat," in *Proceedings of SAT Competition 2019: Solver and Benchmark Descriptions*, 2019.

[5] N. Sörensson, N. Een, and N. Manthey. (2018, May) GitHub repository for MiniSat. https://github.com/conp-solutions/minisat.

[6] A. Nadel and V. Ryvchin, "Chronological backtracking," in *SAT*. Springer, 2018, pp. 111–121.

TT-Open-WBO-Inc-20: an Anytime MaxSAT Solver Entering MSE'20

Alexander Nadel
Email: alexander.nadel@cs.tau.ac.il

***Abstract*—This document describes the solver TT-Open-WBO-Inc-20, submitted to the four incomplete tracks of MaxSAT Evaluation 2020. TT-Open-WBO-Inc-20 is the 2020 version of our solver TT-Open-WBO-Inc [5], which came in first at both the weighted, incomplete categories at MaxSAT Evaluation 2019. TT-Open-WBO-Inc-20 has the following two major new features as compared to the previous version: 1) We now support *unweighted* anytime MaxSAT. We apply the Mrs. Beaver algorithm [3], enhanced by several heuristics from the WMB algorithm [4]; 2) We have integrated into TT-Open-WBO-Inc-20 our new Polosat algorithm for solving the problem of generic optimization in SAT [6].**

I. Introduction

In this document, we assume that a MaxSAT instance comprises a set of *hard* satisfiable clauses H and a *target bit-vector (target)* $T = \{t_n, t_{n-1}, \ldots, t_1\}$, where each *target bit* t_i is a Boolean variable associated with a strictly positive integer weight w_i. The *weight of a variable assignment* μ is $O(T, \mu) = \sum_{i=1}^{n} \mu(t_i) \times w_i$, that is, the overall weight of T's bits, satisfied by μ. Given a MaxSAT instance, a MaxSAT solver is expected to return a model having the minimum possible weight.

Below, we briefly document: 1) Our new Polosat algorithm for the OptSAT problem of optimizing a generic Pseudo-Boolean function in SAT. Polosat is applied in both the weighted and unweighted components of TT-Open-WBO-Inc-20; 2) The weighted component of TT-Open-WBO-Inc-20; 3) The unweighted component of TT-Open-WBO-Inc-20.

Although we did our best to document our solver as precisely as possible, inevitably, we had to omit some details due to space restrictions. OptSAT, Polosat and the weighted component of our solver are described in full detail in [6].

II. The Polosat Algorithm for Generic Optimization in SAT (OptSAT)

Recall that a Pseudo-Boolean (PB) function is a function that maps every full assignment to a real number.

We now state the OptSAT problem. Given a satisfiable formula $F(V)$ in CNF and an objective Pseudo-Boolean (PB) function Ψ, *OptSAT* returns a model μ to F, such that for every model μ' to F, it holds that $\Psi(\mu) \leq \Psi(\mu')$. OptSAT can be thought of as a generalization of MaxSAT, which supports arbitrary PB functions, whereas MaxSAT is restricted to linear PB functions.

In this document, we assume that the objective function Ψ is strictly monotone in a set of observable variables. More specifically, let $Obs \subseteq V$ be a set of *observables* and $\Psi\colon [0,1]^{|V|} \rightarrow \mathbb{R}$ be a PB function. Then: 1) Ψ is *restrictable* to *Obs*, iff for every two assignments θ and λ, such that $\theta(v) = \lambda(v)$ for every $v \in Obs$, it holds that $\Psi(\theta) = \Psi(\lambda)$; 2) Ψ is *strictly monotone in observables Obs*, iff Ψ is restrictable to *Obs* and for every assignment θ and every variable $v \in Obs$, such that $\theta(v) = 1$, it holds that $\Psi(\theta^{\neg v}) < \Psi(\theta)$.

Note that the objective function $O(T, \mu)$ in MaxSAT is strictly monotone in the target bits. First, $O(T, \mu)$ depends only on the target bits. Second, when one of the target bits decreases, $O(T, \mu)$ decreases too, hence $O(T, \mu)$. Hence, an OptSAT algorithm can be applied to solve MaxSAT.

Below, we present our anytime Polosat algorithm for solving OptSAT. It can be used incrementally under assumptions, similarly to modern SAT solvers. Our algorithm is incomplete; it works until a fixed-point, but does not guarantee that the eventual solution is optimal. In this document, we present the strictly monotone version of Polosat. As we shall see, to solve MaxSAT, we integrate Polosat into higher-level MaxSAT algorithms (rather than applying solely Polosat).

Fixing the polarity of a variable v to a Boolean value σ during SAT solver's invocation means assigning v the value σ, whenever v is chosen by the solver's decision heuristic. Intuitively, Polosat carries out a purely SAT-based local search algorithm, based on polarity-fixing.

Polosat is shown in Alg. 1. It receives three parameters: 1) A satisfiable CNF formula F (if the invocation is incremental, assume that F contains all the clauses, provided by the user so far); 2) A (possibly empty) set of assumptions *Asmp*. The assumptions are guaranteed to hold for one particular invocation of the algorithm; 3) The observables *Obs*; 4) The objective PB function to minimize $\Psi\colon [0,1]^{|V|} \rightarrow \mathbb{R}$.

The algorithm maintains an instance of an incremental SAT solver throughout its execution and the best model so far μ. Polosat starts with initializing μ with a model by invoking the SAT solver (line 3). Then, it operates in iterations, where each iteration is called an *epoch* (lines 5 to 16). Each epoch tries to improve μ. An epoch is *good* if it manages to improve μ, otherwise it is *bad*. Our incomplete algorithm finishes whenever a bad epoch is completed.

Each epoch tries to improve the best model so far μ in a loop (lines 8 to 16) by looking for a better solution near μ when, for each loop iteration, one of the variables is forced to flip its value. In addition, the observables are always fixed to 0. The

loop inside each epoch goes over a set of literals B, initialized by all the observable variables assigned to 1 by μ (line 6). For each variable v, `Polosat` tries to find a model near μ with v flipped. This is carried out by fixing the polarities of all the variables, except for the observables, to their values in μ (line 10), followed by a SAT invocation with $\neg v$ as a hard assumption (line 11). If the problem is satisfiable and a model σ better than μ if found, then: 1) μ is updated to σ, 2) the epoch is marked as good. In addition, any observable assigned to 0 in any model is removed from B.

Algorithm 1 `Polosat`

1: **function** SOLVE(CNF F; Literals *Asmp*; Variables *Obs*; $\Psi\colon [0,1]^{|V|} \to \mathbb{R}$)
Require: F is satisfiable and Ψ is strictly monotone in *Obs*
2: Fix the polarities of the observables *Obs* to 0
3: $\mu :=$ SAT(*Asmp*) ▷ μ: the best model so far
4: *is_good_epoch* $:= 1$
5: **while** *is_good_epoch* **do** ▷ One loop is an epoch
6: $B := \{v : v \in Obs, \mu(v) = 1\}$
7: *is_good_epoch* $:= 0$
8: **while** B is not empty **do**
9: $v :=$ *B.front*(); *B.dequeue*()
10: Fix the polarities of the observables *Obs* to 0 and all the other variables (that is, $V \setminus Obs$) to μ
11: $\sigma :=$ SAT($Asmp \cup \{\neg v\}$)
12: **if** SAT **then** ▷ Satisfiable
13: **if** $\Psi(\sigma) < \Psi(\mu)$ **then**
14: $\mu := \sigma$ ▷ Update the best model so far
15: *is_good_epoch* $:= 1$ ▷ Good epoch!
16: $B := \{v : v \in B, \sigma(v) = 1\}$

III. The Weighted Component

We have integrated `Polosat` into the Bounded Multilevel Optimization (BMO)-based anytime MaxSAT algorithm [2], which we call *BMO-based Clustering (BC)*, implemented already in [2], [5].

BC [2] clusters all the target bits to disjoint classes based on their weight. That is, all the targets of the same weight w belong to the same class. Then, the algorithm sorts the classes according to their weight and goes over them one-by-one starting with the class associated with the highest weight. BC tries to falsify as many target bits in each class as possible with incremental SAT invocations. After BC completes processing one class, it fixes the overall number of falsified target bits in that class.

Our implementation simply replaces every SAT invocation with a `Polosat` invocation in BC with the target T as the observables and $O(T, \mu)$ as the objective function. However, there are several subtleties: *a*) We exclude from the set of observables the bits which belong to the fixed classes; *b*) We sort the observables by their weight in decreasing order; *c*) We randomly shuffle the observables within each class before every `Polosat` invocation to diversify `Polosat`'s execution.

A. `Polosat` Enhancements

We modified `Polosat` to keep track of the number of *Models Per Second* (MPS) throughout its execution starting immediately after the initial SAT invocation. MPS is updated and tested after each SAT invocation. If MPS is lower than 1, the current invocation of `Polosat` is terminated, and the high-level BC algorithm falls backs to invoking a plain SAT solver instead of `Polosat` for the rest of its execution. Falling back to SAT makes sense, since SAT/`Polosat` queries tend to become more difficult as the algorithm advances towards the ideal, hence MPS is unlikely to increase.

Furthermore, we use the conflict threshold of 1000 for all the SAT invocations inside `Polosat`, except for the first one.

The two enhancements above are detailed in [6]. On top of that, we have modified `Polosat` to include an additional SAT invocation *before* the one which checks if the current observable v can be flipped (line 11). Let R be the subset of the literals in μ which correspond to all the observables that appear *before* v in *Obs* (where the polarity of each such literal is determined by μ). We add R to the assumptions for our additional SAT invocation. We proceed with the original SAT invocation at line 11, only if no model better than μ was found by the first invocation. Otherwise, we proceed to the next loop iteration. This adjustment essentially combines `Polosat` and `WMB`.

IV. The Unweighted Component

The unweighted component uses the `Mrs. Beaver` algorithm [3], enhanced by the following two heuristics from Sect. 4.1 in [4]: global stopping condition for OBV-BS and size-based switching to complete part.

We have integrated `Polosat` into our algorithm by simply replacing SAT invocations with `Polosat` invocations. We apply all the `Polosat` enhancements from Sect. III-A in our unweighted component, where the adapted strategy is used with the threshold of 2 (rather than 1).

In addition, we use chronological backtracking [7] (with the configuration $\{\texttt{T} = 100, \texttt{C} = 0\}$), which we have implemented in the underlying SAT solver Glucose 4.1 [1]. Furthermore, we take advantage of the `TORC` polarity selection heuristic [4] throughout the algorithm's execution.

References

[1] G. Audemard and L. Simon. On the glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018.
[2] S. Joshi, P. Kumar, S. Rao, and R. Martins. Open-wbo-inc: Approximation strategies for incomplete weighted maxsat. *J. Satisf. Boolean Model. Comput.*, 11(1):73–97, 2019.
[3] A. Nadel. Solving maxsat with bit-vector optimization. In *SAT 2018*, pages 54–72, 2018.
[4] A. Nadel. Anytime weighted maxsat with improved polarity selection and bit-vector optimization. In *FMCAD 2019*, pages 193–202, 2019.
[5] A. Nadel. TT-Open-WBO-Inc: Tuning Polarity and Variable Selection for Anytime SAT-based Optimization. Department of Computer Science Report Series B, Finland, 2019. Department of Computer Science, University of Helsinki.
[6] A. Nadel. On optimizing a generic function in SAT. In *FMCAD 2020*, 2020. To appear.
[7] A. Nadel and V. Ryvchin. Chronological backtracking. In *SAT 2018*, pages 111–121, 2018.

UWrMaxSat: an Efficient Solver in MaxSAT Evaluation 2020

Marek Piotrów
Institute of Computer Science, University of Wrocław
Wrocław, Poland
marek.piotrow@uwr.edu.pl

***Abstract*—UWrMaxSat has been created recently at the University of Wrocław. It is a complete solver for partial weighted MaxSAT instances. It incrementally uses COMiniSatPS by Chanseok Oh (2016) as an underlying SAT solver, but may be compiled with other MiniSat-like solvers. It was developed on the top of KP-MiniSat+ - our PB-solver that was presented at Pragmatics of SAT 2018 and which is an extension of the well-known MiniSat+ solver. In its main configuration, UWrMaxSat applies an unsatisfiability-core-based OLL procedure and uses the sorter-based pseudo-Boolean constraint encoding that was first implemented in kp-minisatp to translate cardinality and pseudo-Boolean constraints into CNF. It can switch to a binary search after a given time and then it uses our new encoding of a pseudo-Boolean goal function, where different bounds on its value are set only by assumptions.**

***Index Terms*—MaxSAT-solver, UWrMaxSAT, COMiniSatPS, sorter-based encoding, core-guided, complete solver**

I. Introduction

At Pragmatics of SAT 2018 workshop, Michał Karpiński and Marek Piotrów presented a new pseudo-Boolean constraint solver called KP-MiniSat+ [6] that was created as an extension of MiniSat+ 1.1 solver by Eén and Sörensson (2012) [3]. In the solver we replaced the encoding based on odd-even sorting networks by a new one using our construction of selection networks called 4-Way Merge Selection Networks [7]. We also optimized mixed radix base searching procedure and added a few other optimizations based on literature. This year the encoding was extended in such a way that a goal function is encoded only once and then SAT-solver assumptions are used to set different bounds on its value. Our experiments showed that the solver is competitive to other state-of-art pseudo-Boolean solvers.

In 2018 KP-MiniSat+ was extended to deal with MaxSAT instances and called UWrMaxSat. In 2019 the new solver was submitted to MaxSAT Evaluation, where it was ranked second places in both main tracks: Weighted Complete Track and Unweighted Complete Track.

II. Description

In this year version of UWrMaxSat (denoted as 1.1), there are several extension to the version 1.0 submitted to MaxSat Evaluation 2019. We give a brief description of them below. For the main features of UWrMaxSat 1.0 see [13]. We continue to use incrementally COMiniSatPS by Chanseok Oh (2016) [12] as an underlying SAT solver, but this year it was patched to better deal with assumptions: all of them are enqueued at once (at level 1) by the SAT solver. The technique was proposed by Hickey and Bacchus in [4].

The default search strategy used by UWrMaxSat is a core-guided linear unsat-sat one, where unsatisfiability cores are processed by the OLL procedure [1], [5], [10] and cardinality constraints generated by it are encoded with the help of 4-Way Merge Selection Networks [7] and Direct Networks [2]. The general description of search strategies used by MaxSat solvers can be found, for example, in [11].

If the linear unsat-sat searching is unsuccessful for a predefined time, it can be switched to a binary search similar to that of MiniSat+ [3] without restarting the SAT solver. In such a case, a pseudo-Boolean goal function is created for the relaxation variables of the remaining soft clauses and encoded by our new technique [8], where the function is translated into SAT clauses only once and the different bounds on its value are set and submitted to the SAT solver by assumptions. The lower and upper bounds found before the switching define an initial binary-search interval.

Due to a new "top-k" track and the corresponding requirements of MaxSat Evaluation 2020, UWrMaxSat can now output the top k models of an MaxSat instance (starting from the optimal one). To use this, an argument `-top=`k should be given to UWrMaxSat. In the solver, the hardening technique was changed in the following way: a decreasing sequence of upper bounds is recorded and if there are at least k of them, then the last k-th one is used in the hardening procedure. After each solution found, a blocking clause consisting of relaxation variables is added to the instance, the value of k is decreased and the search is continued. In addition, the models can be output as a binary 0-1 string (with the help of `-bm`).

Finally, the solver can deal with unbounded integer weights, when it is compiled with the -D BIG_WEIGHTS option. Moreover, it can be linked with MaxPre, an extended open-source preprocessor for weighted partial MaxSAT problems, which was created at University of Helsinki a few years ago [9]. To this end, it should be compiled with the -D MAXPRE option. Note that none of these two options was used in the competition version of UWrMaxSat 1.1, which is submitted to both complete tracks and both top-k tracks (weighted and unweighted). The switching of the search techniques is only used in the complete-weighted track.

ACKNOWLEDGMENTS

I would like to thank Chankseok Oh for his agreement to use COMiniSatPS in the MaxSAT Evaluation. I would like also to thank Niklas Eén and Niklas Sörensson for the development of MiniSat 2.2 and MiniSat+ 1.1.

REFERENCES

[1] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12*, volume 17, pages 212–221, 2012.

[2] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints An Int. J.*, 16(2):195–221, 2011.

[3] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[4] Randy Hickey and Fahiem Bacchus. Speeding up assumption-based sat. In Mikoláš Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing – SAT 2019*, pages 164–182, Cham, 2019. Springer International Publishing.

[5] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Rc2: an efficient maxsat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, pages 53–64, 2019.

[6] Michał Karpiński and Marek Piotrów. Competitive sorter-based encoding of pb-constraints into sat. In Daniel Le Berre and Matti Järvisalo, editors, *Proceedings of Pragmatics of SAT 2015 and 2018*, volume 59 of *EPiC Series in Computing*, pages 65–78. EasyChair, 2019.

[7] Michał Karpiński and Marek Piotrów. Encoding cardinality constraints using multiway merge selection networks. *Constraints An Int. J.*, Apr 2019.

[8] Michał Karpiński and Marek Piotrów. Incremental encoding of pseudo-boolean goal functions based on comparator networks. In *Proceedings of the 23th International Conference on Theory and Applications of Satisfiability Testing (SAT 2020)*, to appear in Lecture Notes in Computer Science. Springer, 2020.

[9] Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An extended MaxSAT preprocessor. In Serge Gaspers and Toby Walsh, editors, *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT 2017)*, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017.

[10] Antonio Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 564–573, Cham, 2014. Springer International Publishing.

[11] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided maxsat solving: A survey and assessment. *Constraints An Int. J.*, 18(4):478–534, 2013.

[12] Chanseok Oh. *Improving SAT Solvers by Exploiting Empirical Characteristics of CDCL*. PhD thesis, New York University, 2016.

[13] Marek Piotrów. Uwrmaxsat - a new minisat+-based solver in maxsat evaluation 2019. pages 11–12, 2019.

BENCHMARK DESCRIPTIONS

BNN verification dataset for Max-SAT Evaluation 2020

Masahiro Sakai
masahiro.sakai@gmail.com

***Abstract*—This article describes a MaxSAT benchmarks that encode the problem of finding minimal adversarial examples for binarized neural networks, and the dataset is submitted to MaxSAT Evaluation 2020.**

***Index Terms*—SAT, MaxSAT, binarized neural networks, adversarial examples**

I. Introduction

Binarized Neural Networks (BNN) [1] are neural networks that their parameters (weights) and inputs are restricted to binary value. Due to these characteristics they are memory efficient and computationally efficient but also are suitable for analysis using SAT-based methods.

One of the interested properties about neural networks is existence of *adversarial examples* [2], [3]. Given a neural network f and its input data x, an adversarial example is a slightly modified input data $x' = x + \tau$ which causes misclassification, *i.e.* $f(x) \neq f(x+\tau)$. Existence of adversarial example may pose a security challenge when deploying the model, therefore it is an important problem to analyze the robustness of neural networks against adversarisal examples.

Narodytska, *et al.* [4] proposed a SAT-based methods to certify the absence of adversarial examples for given BNN f and x within $\|\tau\|_\infty \leq \epsilon$. Our formulation is based on theirs, but we recast it to an optimization problem in the form of maximum satisfiablity problem (MaxSAT) instead of a satisfiability problem (SAT) and we also make some other modifications in encoding as well.

II. Input datasets and training

As in [4], we use digit images from MNIST dataset [5], and its two variants: the MNIST-rot and the MNIST-back-image [6]. Each input image is represented as 8-bit 784 ($= 28 \times 28$) dimension vectors $\{0, \dots, 255\}^{784}$.

Our binarized neural networks have the same architecture as in [4] (see the paper for details). We first scale input image $x \in \{0, \dots, 255\}^{784}$ to $[0, 1]^{784}$, and apply (trained) batch normalization and binarization layers to get binarized image $\{-1, +1\}^{784}$. We denote binarization part as

$$\mathrm{bin} : \{0, \dots, 255\}^{784} \to \{-1, +1\}^{784}$$
$$\mathrm{bin}(x_1, \dots, x_{784}) = (\mathrm{bin}_1(x_1), \dots, \mathrm{bin}_{784}(x_{784}))$$

$$\mathrm{bin}_i : \{0, \dots, 255\} \to \{-1, +1\}.$$

Note that the binarization is performed depending on components (*i.e.* pixel locations).

After that, several internal layers and a output linear layer follow. We denote this part as

$$g : \{-1, +1\}^{784} \to \mathbb{R}^{10},$$

and its output is called *logits*.

Then

$$f(x) = \mathrm{argmax}_{c \in \{0,\dots,9\}} g(\mathrm{bin}(x))_c$$

is the function that our binarized neural network computes.

We implemented and trained BNN models using deep learning framework Chainer [7]. Our implementation is available at https://github.com/msakai/bnn-verification .

III. SAT encoding

In the following we identify $\{\mathrm{false}, \mathrm{true}\}$ with $\{0, 1\}$ for notational simplicity.

A. Inputs and decision variables

In the encoding of [4], they use raw 8-bit image ($x' \in \{0, \dots, 255\}^{784}$) as decision variables and add constraints like $x'_i \in [x - \epsilon, x + \epsilon]$.

We instead use binarized image in $\{-1, +1\}^{784}$ as decision variables and represent $\mathrm{bin}_i(x_i)$ as $2z_i - 1$ using a boolean variables z_i. We consider original 8-bit image as dependent variables, and use them in cost function instead of defining them as decision variables.

B. Relationship between inputs and logits

The g part is encoded to CNF in a way similiar to [4], but we use *totalizer* [8] instead of *sequential counter* [9] for encoding cardinality constraints.

C. Relationship between logits and outputs

For the argmax part, let $\mathrm{logits} \in \mathbb{R}^{10}$ be output of g, and let $\{y_c\}_{c \in \{0,\dots,9\}}$ be ten boolean variables representing output of f (*i.e.* $y_i \Leftrightarrow f(x') = i$).

The fact that $\{y_c\}_c$ is a one-hot vector can be expressed as cardinality constraint $\sum_i y_i = 1$.

Also, if y_c then c-th logit must be largest, *i.e.*

$$y_c \to \mathrm{logits}_c \geq \mathrm{logits}_{c'} \text{ for all } c'.$$

Because logits_c is represented as $(\sum_j W_{c,j}(2u_j - 1)) + b_c$ where $W_{c,j} \in \{-1, +1\}$ and u_j are some boolean variables

introduced in encoding of g, this can be expressed as conditional cardinality constraints:

$$y_c \to \sum_j \frac{(W_{c,j} - W_{c',j})}{2} u_j \geq \left\lceil \frac{\sum_j (W_{c,j} - W_{c',j}) + b_{c'} - b_c}{4} \right\rceil.$$

We encode those (conditional) cardinality constraints using totalizer.

Finally, if $f(x) = c$ then we add $\neg y_c$ to require the input to be misclassified.

IV. Cost function

As we want to find an adversarial example with the smallest perturbation, we want to set a cost for modifying x_is. Because our decision variable is z_is, this corresponds to adding a soft constraint[1] $z_i = \frac{bin_i(x_i)+1}{2}$ with appropriate cost of violation.

First, we define δ_i as the smallest change of x_i to flip its binarized value, *i.e.* $\mathrm{bin}_i(x_i+\delta_i) \neq \mathrm{bin}_i(x_i)$. If it is impossible to flip (this is the case when bin_i is constant function), we define δ_i to be $\perp$.

Then it is easy to define cost functions to minimize L_p-norms including L_∞-norm.

A. L_p-norms for $p \neq \infty$

We add $z_i = \frac{bin_i(x_i)+1}{2}$ as a soft constraint with cost $|\delta_i|^p$ for each i if $\delta_i \neq \perp$, and add it as a hard constraint if $\delta_i = \perp$. Note that we get unweighted (partial) Max-SAT instances in case of L_0-norm.

B. L_∞-norm

Let $\Delta = \{|\delta_i| \mid \delta_i \neq \perp\}$ and $w_1 < \ldots < w_{|\Delta|} \in \Delta$ be sorted in ascending order, and we introduce new boolean variable r_k called *relaxation variable* for each w_k. Then we add following constraints.

- Unit soft clause $(\neg r_k)$ with cost $|w_k|$ for each $k \in \{1, \ldots, |\Delta|\}$.
- Relaxation variables are lower closed: $r_k \to r_{k-1}$ for each $k > 0$
- z_i is allowed to be flipped only when corresponding relaxation variable is true: $\neg r_k \to z_i = \frac{bin_i(x_i)+1}{2}$ for each i with $|\delta_i| = w_k$.
- z_i is fixed if it cannot be flipped: $z_i = \frac{bin_i(x_i)+1}{2}$ for each i with $\delta_i = \perp$.

V. Problem files

We choose 20 images (2 images for each digit class) from test data of each data set (mnist, mnist_rot, mnist_back_image) that trained BNN models predicted true label.

Adversarial example finding problem is generated for each image using the BNN model trained for the dataset. Generated instances have 1.8 M variables and 132 M clauses on average.

[1]As we assume a concrete x, right hand side is 0 or 1, therefore this constraint is represented as a unit clause $\neg z_i$ or z_i

File names are in the form of "`bnn_{dataset_name}_{instance number}_label{true label}_adversarial_norm_inf_totalizer.wcnf`" where

- `dataset name` is mnist, mnist_rot or mnist_back_image,
- `instance number` is image number in test dataset,
- `true label` is one of $0 \ldots 9$.

References

[1] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4107–4115. [Online]. Available: http://papers.nips.cc/paper/6573-binarized-neural-networks.pdf

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1312.6199

[3] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[4] N. Narodytska, S. P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, "Verifying properties of binarized deep neural networks," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 6615–6624. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16898

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 473–480. [Online]. Available: https://doi.org/10.1145/1273496.1273556

[7] S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa, S. Saito, S. Suzuki, K. Uenishi, B. Vogel, and H. Yamazaki Vincent, "Chainer: A deep learning framework for accelerating the research cycle," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 2002–2011.

[8] O. Bailleux and Y. Boufkhad, "Efficient cnf encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming – CP 2003*, F. Rossi, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 108–122.

[9] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming - CP 2005*, P. van Beek, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 827–831.

MaxSAT Evaluation 2020 - Benchmark: Identifying Maximum Probability Minimal Cut Sets in Fault Trees

Martín Barrère and Chris Hankin
Institute for Security Science and Technology, Imperial College London, UK
{m.barrere, c.hankin}@imperial.ac.uk

***Abstract*—This paper presents a MaxSAT benchmark focused on the identification of Maximum Probability Minimal Cut Sets (MPMCSs) in fault trees. We address the MPMCS problem by transforming the input fault tree into a weighted logical formula that is then used to build and solve a Weighted Partial MaxSAT problem. The benchmark includes 80 cases with fault trees of different size and composition as well as the optimal cost and solution for each case.**

***Index Terms*—MaxSAT, Benchmark, Fault trees, Fault Tree Analysis, Reliability, Cyber-Physical Security, Dependability.**

I. Problem overview

Fault Tree Analysis (FTA) is an analytical tool aimed at modelling and evaluating how complex systems may fail. FTA is widely used as a risk assessment tool in safety and reliability engineering for a broad range of industries including aerospace, power plants, nuclear plants, and others high-hazard fields [1]. Essentially, a fault tree is a directed acyclic graph (DAG) which involves a set of basic events (e.g. component failures) that are combined using logic operators (e.g. AND and OR gates) to model how these events may lead to an undesired state of the system normally represented at the root of the tree (top level event).

Our work is focused on a novel measure for FTA in the form of a hybrid analysis technique that involves quantitative and qualitative aspects of fault trees. From a qualitative perspective, we focus on Minimal Cut Sets (MCS). An MCS is a minimal combination of events that together cause the top level event. As such, MCSs are fundamental for structural analysis. The problem is that, in large scenarios, computing all MCSs might be very expensive and there might be hundreds of MCSs, which makes it hard to handle and prioritise which MCSs should be attended first. In that context, the goal of this work is to identify the MCS with maximum probability. We call this problem the MPMCS. This is an NP-complete problem and we use a MaxSAT-based approach to address it.

II. Simple example

The fault tree shown in Fig. 1 illustrates the different combinations of events that may lead to the failure of an hypothetical Fire Protection System (FPS) based on [2]. The FPS can fail if either the fire detection system or the fire suppression mechanism fails. In turn, the detection system can fail if both sensors fail simultaneously (events x_1 and x_2), while the suppression mechanism may fail if there is no water ($x3$), the sprinkler nozzles are blocked (x_4), or the triggering system does not work. The latter can fail if neither of its operation modes (automatic (x_5) or remotely operated) works properly. The remote control can fail if the communications channel fails (x_6) or the channel is not available due to a cyber attack, e.g. DDoS attack (x_7). Each basic event has an associated value that indicates its probability of occurrence $p(x_i)$.

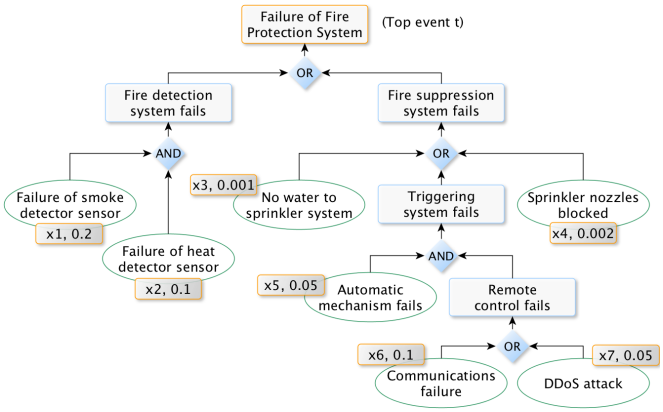


Fig. 1. Fault tree of a cyber-physical fire protection system (simplified)

A fault tree F can be represented as a Boolean equation $f(t)$ that expresses the different ways in which the top event t can be satisfied [3]. In our example, $f(t)$ is as follows:

$$f(t) = (x_1 \wedge x_2) \vee (x_3 \vee x_4 \vee (x_5 \wedge (x_6 \vee x_7)))$$

The objective is to find the minimal set of logical variables that makes the equation $f(t)$ *true* and whose joint probability is maximal among all minimal sets. In our example, the MPMCS is $\{x_1, x_2\}$ with a joint probability of 0.02.

III. MaxSAT formulation strategy

Given a fault tree and its logical formulation $f(t)$, we carry out a series of steps to compute the MPMCS as follows.

1. Logical transformation. Since we are interested in minimising the number of satisfied clauses, which is opposed to what MaxSAT does (maximisation), we flip all logic gates but keep all events in their positive form. In our example, we obtain: $g(t) = (x_1 \vee x_2) \wedge (x_3 \wedge x_4 \wedge (x_5 \vee (x_6 \wedge x_7)))$.

This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant No 739551 (KIOS CoE).

Then, the objective is to satisfy $\neg g(t)$ where the falsified variables will indicate the minimum set of events that must simultaneously occur to trigger the top level event. A more detailed explanation of this transformation can be found in [4]. We then use the Tseitin transformation to produce in polynomial time an equisatisfiable CNF formula [5].

2. MaxSAT weights. Due to the fact that MaxSAT is additive in nature and the MPMCS problem involves the multiplication of decision variables, we transform the probabilities into a negative log-space so the multiplication becomes a sum. In addition, many SAT solvers only support integer weights so we perform a second transformation by right shifting (multiplying by 10) every value until the smallest value is covered with an acceptable level of precision. For example, 0.001 and 0.00007 would become 100 and 7 (right shift 5 times). Additional variables introduced by the Tseitin transformation have weight 0. We then specify the problem as a Partial Weighted MaxSAT instance by assigning the transformed probability values as a penalty score for each decision variable.

3. Parallel SAT-solving architecture. Since different SAT solvers normally use different resolution techniques, some of them are very good at some instances and not that good at others. To address this issue, we run multiple SAT-solvers in parallel and pick the solution of the solver that finishes first. We have experimentally observed that the combination of different solvers provides good results in terms of performance and scalability. Once the solution has been found, we translate back the transformed values into their stochastic domain and output the MCS with maximum probability.

IV. Fault tree generation

The benchmark presented in this paper relies on our open source tool MPMCS4FTA [6]. We have used MPMCS4FTA to generate and analyse synthetic pseudo-random fault trees of different size and composition. We use AND/OR graphs as the underlying structure to represent fault trees. The benchmark presented in [7] also considers AND/OR graphs as a means to represent operational dependencies between components in industrial control systems [8]. However, the instances presented in this paper differ in that: 1) they are restricted to directed acyclic graphs (DAGs), 2) only the basic events represented at the leaves of the fault tree involve a probability of failure, and 3) leaves can have more than one parent in order to relax the definition of strict logical trees.

We control the size and composition of a random fault tree of size n according to a configuration $R = (R_{AT}, R_{AND}, R_{OR})$. $R_{AT} \in [0, 1]$ indicates the proportion of atomic nodes (basic events) with respect to size n (e.g. 0.2 means 20%) whereas R_{AND} and R_{OR} indicate the proportion of AND and OR nodes respectively. To create a fault tree of size n, we first create two lists: $L = \{l_1, \ldots, l_m\}$ and $A = \{a_1, \ldots, a_s\}$. L is a random sequence of AND and OR nodes with the specified proportions for each operator where $m = n * (R_{AND} + R_{OR})$. A is a list of atomic nodes where $s = n * R_{AT}$, thus $n = m + s$. In addition, each atomic node has a random probability of failure $p(a_i) \in [0, 1]$.

To ensure connectivity, we first create the root node t and connect l_1 to t ($l_1 \rightarrow t$). Then, for each logic node l_i in the sequence L, we randomly choose k nodes l_j ahead (thus $j > i$) and create k edges ($l_j \rightarrow l_i$) in the tree. When the remaining nodes in L are not enough to cover k nodes, we use random atomic nodes from A. At this point, we also make sure that l_i points to at least one previous node in the sequence L. If that is not the case, we choose a random node l_h (with $h < i$) and create an edge ($l_i \rightarrow l_h$). Once the sequence L has been processed, we traverse the list A and connect each atomic node a_i as follows. First, we draw a random value k' between 1 and k. Then, we add random edges ($a_i \rightarrow l_j$) from a_i to logic nodes l_j until we cover k' connections.

V. Benchmark description

Out dataset includes 80 cases in total, and can be obtained at [6]. It contains fault trees with four different sizes: 2500, 5000, 7500, and 10000 nodes (20 cases each). For each tree size, we consider two different graph configurations, $R_1 = (0.8, 0.1, 0.1)$ and $R_2 = (0.6, 0.2, 0.2)$, which determine the composition of the fault trees (10 cases each). Table I shows the identifiers of the cases within each one of these categories.

#Nodes/Configurations	$R_1 = (0.8, 0.1, 0.1)$	$R_2 = (0.6, 0.2, 0.2)$
2500	1 to 10	11 to 20
5000	21 to 30	31 to 40
7500	41 to 50	51 to 60
10000	61 to 70	71 to 80

TABLE I
Benchmark cases and configurations

Each case is specified in an individual **.wcnf** (DIMACS-like, weighted CNF) file named with the case id and the number of nodes involved. The weight for hard clauses (*top* value) has been set to 2.0×10^9. The weight of each soft constraint is an integer value that corresponds to the transformation (right shifting) of the probability value in $-log$ space. Tables II and III detail each case as well as the results obtained with our tool. The field **id** identifies each case. **gNodes** and **gEdges** indicate the total number of nodes and edges in the fault tree. **gAT**, **gAND**, and **gOR**, indicate the approximate composition of the graph in terms of atomic (basic events), AND, and OR nodes. **tsVars** and **tsClauses** show the number of variables and clauses involved in the MaxSAT formulation after applying the Tseitin transformation. **time** shows the resolution time reported by MPMCS4FTA in milliseconds. Currently, the MaxSAT solvers used in MPMCS4FTA are SAT4J [9] and a Python-based linear programming approach using Gurobi [10]. **size** indicates the number of nodes identified in the MPMCS solution. **intLogCost** indicates the cost of the solution in $-log$ space as an integer value (right shifted). **logCost** indicates the cost of the solution in $-log$ space. **MPMCS probability** indicates the joint probability of the MPMCS. These experiments have been performed on a MacBook Pro (16-inch, 2019), 2.4 GHz 8-core Intel Core i9, 32 GB 2666 MHz DDR4.

id	gNodes	gEdges	gAT	gAND	gOR	tsVars	tsClauses	time	size	intLogCost	logCost	MPMCS probability
1	2500	7151	2002	250	250	1978	6258	618	1	246	2.46E-4	0.999754
2	2500	7192	2002	250	250	4268	15026	850	447	464771733	464.771733	1.42239870668983E-202
3	2500	7196	2002	250	250	1207	3763	290	1	27591	0.027591	0.972787
4	2500	7140	2002	250	250	4211	14673	833	1	238	2.38E-4	0.999763
5	2500	7107	2002	250	250	3907	13325	821	1	7879	0.007879	0.992153
6	2500	7202	2002	250	250	3410	11350	749	70	81474531	81.474531	4.147681160335815E-36
7	2500	7126	2002	250	250	3304	10922	711	1	315	3.15E-4	0.999685
8	2500	7181	2002	250	250	3752	12713	826	1	2576	0.002576	0.997428
9	2500	7157	2002	250	250	3011	9847	625	1	4301	0.004301	0.995709
10	2500	7156	2002	250	250	642	1982	211	19	12423488	12.423488	4.0231156723921624E-6
11	2500	6831	1502	500	500	3873	14170	912	1	28842	0.028842	0.971571
12	2500	6782	1502	500	500	2377	7941	550	1	32680	0.03268	0.96785
13	2500	6814	1502	500	500	3216	11235	700	13	10769787	10.769787	2.1025796252653052E-5
14	2500	6700	1502	500	500	3268	11376	728	197	207945092	207.945092	4.9088521396478804E-91
15	2500	6897	1502	500	500	3063	10555	817	1	3262	0.003262	0.996744
16	2500	6849	1502	500	500	2044	6765	470	1	191116	0.191116	0.826037
17	2500	6787	1502	500	500	3158	10955	723	1	284520	0.28452	0.752376
18	2500	6872	1502	500	500	3433	12147	773	139	130484455	130.484455	2.1453798325228181E-57
19	2500	6821	1502	500	500	2506	8439	534	17	9662887	9.662887	6.36019885647539E-5
20	2500	6831	1502	500	500	3848	14095	821	1	3507	0.003507	0.996501
21	5000	14324	4002	500	500	4149	13224	932	229	217397271	217.397271	3.8565352927569054E-95
22	5000	14313	4002	500	500	8532	29961	925	614	641968767	641.968767	1.5912873405576694E-279
23	5000	14329	4002	500	500	6971	23338	842	240	251915559	251.915559	3.9351584673463555E-110
24	5000	14361	4002	500	500	8020	27645	843	1	793	7.93E-4	0.999209
25	5000	14370	4002	500	500	8965	32190	843	1	1858	0.001858	0.998144
26	5000	14317	4002	500	500	5443	17581	827	1	3615	0.003615	0.996391
27	5000	14407	4002	500	500	8113	28023	842	277	253971185	253.971185	5.035082961027143E-111
28	5000	14365	4002	500	500	8952	32153	837	1041	994658460	994.65846	0.0
29	5000	14321	4002	500	500	8859	31477	833	379	378308687	378.308687	5.051735441001231E-165
30	5000	14316	4002	500	500	7948	27315	830	1	970	9.7E-4	0.999032
31	5000	13607	3002	1000	1000	6384	22218	938	1	2530	0.00253	0.997474
32	5000	13730	3002	1000	1000	7330	26390	863	65	63984958	63.984958	1.62844121698006E-28
33	5000	13687	3002	1000	1000	3181	10354	683	1	25289	0.025289	0.975029
34	5000	13600	3002	1000	1000	6293	21870	834	407	424495269	424.495269	4.413071223454673E-185
35	5000	13712	3002	1000	1000	7361	26650	895	179	171277203	171.277203	4.1251154050451916E-75
36	5000	13709	3002	1000	1000	6231	21647	831	22	19249301	19.249301	4.366753474609794E-9
37	5000	13612	3002	1000	1000	6202	21523	931	257	273826234	273.826234	1.2035873310274229E-119
38	5000	13664	3002	1000	1000	4482	14952	824	1	4317	0.004317	0.995693
39	5000	13631	3002	1000	1000	7395	26641	827	83	89562456	89.562456	1.2695246380697898E-39
40	5000	13641	3002	1000	1000	7825	28775	831	1	5974	0.005974	0.994045

TABLE II
BENCHMARK DESCRIPTION - CASES 1 TO 40

id	gNodes	gEdges	gAT	gAND	gOR	tsVars	tsClauses	time	size	intLogCost	logCost	MPMCS probability
41	7500	21502	6002	750	750	8871	28951	965	1	160	1.6E-4	0.999841
42	7500	21515	6002	750	750	7191	23069	852	1	393	3.93E-4	0.999607
43	7500	21497	6002	750	750	5716	18114	843	1	1095	0.001095	0.998906
44	7500	21536	6002	750	750	6476	20645	849	600	607247314	607.247314	1.8912103369207186E-264
45	7500	21472	6002	750	750	10277	34266	859	251	235979386	235.979386	3.279829621872166E-103
46	7500	21607	6002	750	750	10235	34064	849	31	27638401	27.638401	9.927826703704467E-13
47	7500	21609	6002	750	750	11377	38597	920	689	644477962	644.477962	1.2810988897753624E-280
48	7500	21397	6002	750	750	4488	14083	815	1	18442	0.018442	0.981728
49	7500	21410	6002	750	750	12792	44789	1031	668	672741572	672.741572	6.81228495760467E-293
50	7500	21566	6002	750	750	13253	47290	851	1	9154	0.009154	0.990888
51	7500	20454	4502	1500	1500	11031	39763	972	1	2151	0.002151	0.997852
52	7500	20450	4502	1500	1500	8927	30739	855	1	738	7.38E-4	0.999263
53	7500	20616	4502	1500	1500	11843	43792	894	1	37	3.7E-5	0.999964
54	7500	20530	4502	1500	1500	9961	35071	1053	502	480184105	480.184105	2.8797108920892045E-209
55	7500	20563	4502	1500	1500	9462	32930	1368	769	739302414	739.302414	8.45E-322
56	7500	20493	4502	1500	1500	9084	31398	833	1	7545	0.007545	0.992484
57	7500	20491	4502	1500	1500	4922	16088	817	1	104472	0.104472	0.9008
58	7500	20594	4502	1500	1500	5943	19507	987	267	256660486	256.660486	3.4340775952647096E-112
59	7500	20406	4502	1500	1500	9340	32356	898	158	148111431	148.111431	4.74472781242486E-65
60	7500	20445	4502	1500	1500	8882	30572	827	1	14066	0.014066	0.986033
61	10000	28613	8002	1000	1000	16234	56222	1087	1	1904	0.001904	0.998099
62	10000	28675	8002	1000	1000	14261	47804	914	197	185985480	185.98548	1.6901841317920728E-81
63	10000	28558	8002	1000	1000	13755	45717	893	1	43	4.3E-5	0.999957
64	10000	28738	8002	1000	1000	13370	44343	882	1	127	1.27E-4	0.999874
65	10000	28752	8002	1000	1000	15537	53105	917	643	606121928	606.121928	5.826520007473361E-264
66	10000	28803	8002	1000	1000	9981	32065	852	1	796	7.96E-4	0.999205
67	10000	28632	8002	1000	1000	13418	44550	861	448	439405919	439.405919	1.4772121624185204E-191
68	10000	28830	8002	1000	1000	17774	63650	874	1	3047	0.003047	0.996959
69	10000	28717	8002	1000	1000	14505	48831	861	1	1691	0.001691	0.998311
70	10000	28604	8002	1000	1000	16032	55089	855	1	436	4.36E-4	0.999564
71	10000	27114	6002	2000	2000	15244	55476	2286	652	652324945	652.324945	5.016628484164324E-284
72	10000	27515	6002	2000	2000	10588	36029	867	1	15974	0.015974	0.984154
73	10000	27411	6002	2000	2000	9596	32332	862	422	440653751	440.653751	4.240514855635819E-192
74	10000	27271	6002	2000	2000	15985	59167	873	1	2033	0.002033	0.997969
75	10000	27228	6002	2000	2000	13506	47651	2223	621	639112478	639.112478	2.7423451190246526E-278
76	10000	27345	6002	2000	2000	12066	41598	1253	326	307525901	307.525901	2.779537506735469E-134
77	10000	27310	6002	2000	2000	10310	34812	835	1	10970	0.01097	0.989091
78	10000	27306	6002	2000	2000	12092	41711	1004	228	218680041	218.680041	1.0684631282749114E-95
79	10000	27315	6002	2000	2000	14069	50130	848	1	1447	0.001447	0.998555
80	10000	27375	6002	2000	2000	14851	53699	859	1	180	1.8E-4	0.999821

TABLE III
BENCHMARK DESCRIPTION - CASES 41 TO 80

REFERENCES

[1] E. Ruijters and M. Stoelinga, "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools," *Computer Science Review*, vol. 15-16, pp. 29 – 62, 2015.

[2] S. Kabir, "An overview of Fault Tree Analysis and its application in model based dependability analysis," *Expert Systems with Applications*, vol. 77, pp. 114 – 135, 2017.

[3] W. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick III, and J. Railsback, "Fault Tree Handbook with Aerospace Applications," *Office of Safety and Mission Assurance, NASA Headquarters, US*, 2002.

[4] M. Barrère and C. Hankin, "Fault Tree Analysis: Identifying Maximum Probability Minimal Cut Sets with MaxSAT," https://arxiv.org/abs/2005.03003, May 2020.

[5] G. S. Tseitin, "On the Complexity of Derivation in Propositional Calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part II*, A. Slisenko, Ed., 1970, pp. 234–259.

[6] M. Barrère, "MPMCS4FTA - Maximum Probability Minimal Cut Sets for Fault Tree Analysis," https://github.com/mbarrere/mpmcs4fta, March 2020.

[7] M. Barrère, C. Hankin, N. Nicolaou, D. Eliades, and T. Parisini, "MaxSAT Evaluation 2019 - Benchmark: Identifying Security-Critical Cyber-Physical Components in Weighted AND/OR Graphs. In MaxSAT Evaluation 2019 (MSE19)," https://arxiv.org/abs/1911.00516, 2019.

[8] M. Barrère, C. Hankin, N. Nicolaou, D. Eliades, and T. Parisini, "Measuring cyber-physical security in industrial control systems via minimum-effort attack strategies," *Journal of Information Security and Applications*, vol. 52, pp. 1–17, June 2020. [Online]. Available: https://doi.org/10.1016/j.jisa.2020.102471

[9] "SAT4J," http://www.sat4j.org/, Cited June 2020.

[10] Gurobi, "Gurobi Optimizer," https://www.gurobi.com/, 2020, Cited June 2020.

Partial (Un-)Weighted MaxSAT Benchmarks: Minimizing Witnesses for Security Weaknesses in Reconfigurable Scan Networks

Pascal Raiola, Tobias Paxian and Bernd Becker
Faculty of Engineering, University of Freiburg
{ raiolap | paxiant | becker }@informatik.uni-freiburg.de

***Abstract*—The proposed benchmarks describe optimization problems for a security application, which investigates data flow security weaknesses of recofigurable scan networks (RSNs). The goal of the optimization problem is to minimize both the amount of times so-called Scan Path Branching is used and to reduce the information contained in the witness of the data flow security weakness, so that the witness is of human-readable size.**

***Index Terms*—IEEE Std 1687, IJTAG, Reconfigurable Scan Networks, RSN, Partial Weighted MaxSAT, Benchmark, security weaknesses**

I. Introduction

For on-chip diagnosis (and other purposes) reconfigurable scan networks (RSNs, cf. IEEE Std 1687 [1]) are increasingly deployed in industrial designs. However, if no adequate precautions are taken by the designer, an attacker might exploit the powerful observability and controllability properties of an RSN, exploiting e.g. a so-called *data flow security weakness*; further details on data flow security weaknesses and how to resolve them are found e.g. in [2], [3]. For large RSNs taking those precautions without algorithmic support is virtually impossible.

The technique of [4] generates in a first step a set of conditions (called "witness") to instantiate an insecure data flow. Since such witnesses contain often by far too much (not-needed) information to be human-readable, the MaxSAT solver *Pacose* [5] was invoked to reduce the "witnesses" to "minimal witnesses", which are much more compact, but at the same time provide all the information necessary for a witness.

The proposed benchmarks allow to calculate the above described minimization, where each benchmark models the problem of minimizing a certain witness. The original instances vary in complexity, ranging from small benchmarks with only 47 soft clauses to very large benchmarks with more than 1.2 million soft clauses and nearly 30 million hard clauses.

II. Preliminaries

A. Reconfigurable Scan Network

An example *reconfigurable scan network* (RSN) is illustrated in Figure 1. In an RSN, different active scan paths can be configured. Figure 1 pictures the active scan path if all scan multiplexers are set to 1 (green dashed line). An assignment to the select signals of all scan multiplexers is called an *RSN configuration*.

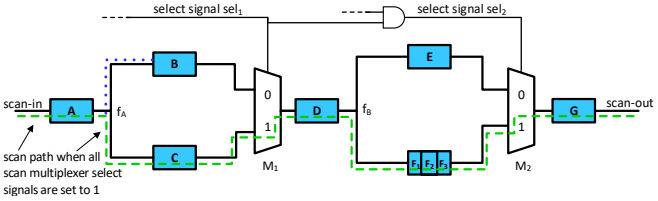


Fig. 1. Reconfigurable scan network with 9 scan flip-flops (A to G), 2 fan-outs (f_A and f_B) and 2 scan multiplexers (M_1 and M_2)

B. Attack Scenario

In the following we will present the *data flow* security threat, that has been investigated by [2] and [3]: If scan infrastructure is introduced, components deeply inside the circuit become connected with each other. Those components might be from different vendors and sources, where one is untrusted potentially due to lower security standards: A sensor module might be very vulnerable to side-channel attacks, as it hardly contains secret information. Thus, an attacker must be prevented from shifting confidential data over a scan path involving such an untrusted module. Similarly, automated tests must not shift confidential data over such a scan path.

It should also be noted, that the data stream in Figure 1 (green dashed line) not only follows the illustrated direct path, but additionally branches at each fan-out (illustrated for f_A as blue dotted line). If B is located inside a third-party module, it should be considered, that data on the scan path potentially also enters the third-party module. If this third-party module is untrusted, correct behavior of the scan infrastructure inside that module cannot be guaranteed.

Therefore, even though a module is not part of the current scan path, it might still leak sensitive data e.g. via a side-channel attack. If an attacker exploits such a leak, we say that *Scan Path Branching* (*SPB*) was used for the attack and speak of an *indirect* scan path.

III. Problem Description

Information on a security weakness can be expressed as a *witness*, i.e. instructions on how to exploit the security weakness. [4] generates such witnesses, which at first contain in general by far too much information to be human-readable. To provide condensed information on the weakness, a witness containing as little information as possible is to be determined.

This corresponds to formulating and solving a MaxSAT problem with a suitable MaxSAT solver.

A. Building the CNF(s)

The SAT-encoding of the presented RSN security problem is introduced in [4] and uses the encoding of [6] as basis. It models the state of the RSN during the scan configurations, which are used for the witnessed data flow weakness. Naturally all clauses of the RSN model are hard clauses. Two more (unit) hard clauses are added, which enforce the beginning and the end of the insecure data path.

The three possible values (0, 1 or *don't care*) of scan registers and signal lines of the RSN are denoted with two Boolean variables, where a helper variable is `true` if and only if the respective value is *don't care*. For each helper variable a soft clause of weight 1 is employed. For clarity, the set of these soft clauses is called S_{DC}.

These hard and soft clauses fully describe the problem of minimizing the so-called *direct* witness and are thus partial **unweighted** MaxSAT problems. Only for the minimization of the so-called *shortest* witness, more soft clauses were utilized, leading to partial **weighted** MaxSAT problems: Since the technique Scan Path Branching (*SPB*) is in general an elaborate technique, the witness-minimization also included soft clauses to minimize the use of *SPB*. To do so, for every potential point to employ *SPB* for an attack (i.e. for every scan multiplexer), one soft clause was added, which models, that *SPB* was *not* used at that point. The weight of every such soft clause was set to be higher than all the weights of S_{DC} combined, i.e. to $|S_{DC}| + 1$.

B. Interpreting the MaxSAT solution

The weight of the satisfied soft clauses can be used to show the effectiveness of the minimization. For the goal of the application, a designer needs the full variable assignment of the solution. Then, the designer can use all variables, which are not set to *don't care*, to straightforwardly see the core of the security weakness and choose a method to fix the weakness. It is also important to minimize the usage of *SPB* for the shortest witness, since a security weakness without *SPB* potentially requires more drastic countermeasures, which solve other security weaknesses on-the-fly.

C. File Name Convention

The file name consists of the following parts (in order, separated with dashes):

- Every file name starts with the description of the MaxSAT problem: "RSN_Security_Min_Witness"
- Then follows one of the two words "shortest" and "direct", stating which witness was minimized with the described MaxSAT problem.
- Next, the name of the RSN benchmark, for which the witness is generated. Only problems generated by large RSN benchmarks of the BASTION benchmark sets ITC '16 [7] and DATE '19 [3] were chosen for the proposed MaxSAT benchmark set.
- Lastly, the letter 'D', followed by a hexadecimal digit, describing which of 16 probability distributions was used to randomly generate security properties of the RSN.

For example, the proposed MaxSAT benchmark for the MaxSAT problem of minimizing a shortest witness for the RSN of the benchmark *Mingle* of probability distribution 8 is named:

RSN_Security_Min_Witness-Shortest-Mingle-D8.wcnf

References

[1] "IEEE standard for access and control of instrumentation embedded within a semiconductor device," *IEEE Std 1687-2014*, pp. 1–283, 2014.

[2] P. Raiola, M. A. Kochte, A. Atteya, L. R. Gómez, H. Wunderlich, B. Becker, and M. Sauer, "Detecting and resolving security violations in reconfigurable scan networks," in *24th IEEE International Symposium on On-Line Testing And Robust System Design, IOLTS 2018, Platja D'Aro, Spain, July 2-4, 2018*, 2018, pp. 91–96.

[3] P. Raiola, B. Thiemann, J. Burchard, A. Atteya, N. Lylina, H. Wunderlich, B. Becker, and M. Sauer, "On secure data flow in reconfigurable scan networks," in *Design, Automation & Test in Europe Conference & Exhibition, DATE*, 2019, pp. 1016–1021.

[4] P. Raiola, T. Paxian, and B. Becker, "Minimal witnesses for security weaknesses in reconfigurable scan networks," in *To be published in Proceedings of the 25th IEEE European Test Symposium (ETS)*, 2020.

[5] T. Paxian, S. Reimer, and B. Becker, "Dynamic polynomial watchdog encoding for solving weighted MaxSAT," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2018, pp. 37–53.

[6] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Reconfigurable scan networks: Modeling, verification, and optimal pattern generation," *ACM Trans. Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 2, pp. 30:1–30:27, 2015.

[7] A. Tsertov, A. Jutman, S. Devadze, M. S. Reorda, E. Larsson, F. G. Zadegan, R. Cantoro, M. Montazeri, and R. Krenz-Baath, "A suite of IEEE 1687 benchmark networks," in *IEEE International Test Conference, ITC*. IEEE, 2016, pp. 1–10.

Description of Benchmarks on Coalition Structure Generation

Xiaojuan Liao
Chengdu University of Technology
Chengdu, China
liao_xiaojuan@126.com

Miyuki Koshimura
Kyushu University
Fukuoka, Japan
koshi@inf.kyushu-u.ac.jp

I. Problem description

Coalition Structure Generation (CSG), which is one of the main research issues in the domain of coalition games, involves partitioning a set of agents to maximize the total value of all the coalitions. The general assumption for solving the CSG problem is that given any coalition structure, the coalition's value is not affected by how non-members are partitioned. Such settings are known as Characteristic Function Games (CFGs), where the value of a coalition is given by *characteristic function* $v : 2^A \rightarrow \mathbb{R}$ that assigns a real-valued payoff to each coalition $C \subseteq A$. The value of coalition structure CS is called *social welfare*, denoted as $V(CS)$, given by $V(CS) = \sum_{C_i \in CS} v(C_i)$. The objective of solving the CSG problem is to find an optimal coalition structure that maximizes the social welfare, i.e., given A, find CS^* such that $\forall CS \in \Pi(A), V(CS^*) \geq V(CS)$. When the number of agents becomes large, compact methods of representing characteristic functions are exploited, such as ***MC-net*** [1], which largely reduce the space necessary for representation.

An MC-net consists of set of *rules* R. Each $r_i \in R$ is expressed as $I_i \rightarrow w_i$, where $w_i \in \mathbb{R}$, and I_i is the condition of rule r_i, denoted by a conjunction of literals, i.e., $\{a_1 \wedge \cdots \wedge a_l \wedge \neg a_{l+1} \wedge \cdots \wedge \neg a_{m_i}\}$, where m_i is the number of agents in I_i. For r_i, P_i denotes the *positive literals* where $P_i = \{a_j\}_{j=1}^{l}$, and N_i denotes the *negative literals* where $N_i = \{a_j\}_{j=l+1}^{m_i}$. Rule r_i *applies to* coalition C if $P_i \subseteq C$ and $N_i \cap C = \emptyset$. Set of rules $R' \subseteq R$ is called *feasible* if there exists coalition structure CS where each rule $r \in R'$ applies to some $C \in CS$.

Based on MC-nets, Weighted Partial MaxSAT (WPM) encoding has shown high efficiency in solving the CSG problem [2], [3], [5], which encodes a set of constraints into Boolean propositional logic and employs an off-the-shelf WPM solver to find out the optimal solution.

II. Parameters used for generating the instances

The way of generating instances follows earlier works [4]–[6], summarized as follows. First, we created a coalition with one random agent and repeatedly added a new random agent with probability α until an agent is not added or the coalition includes all agents. The value of a rule is uniformly at random chosen between 0 and the number of agents in the rule. In addition, for each coalition that contains more than one agent, we convert an agent from positive to negative with p probability. Furthermore, we convert the value of a coalition from positive to negative with q probability. Throughout the experiment, we set the number of rules equal to the number of agents. Parameter $\alpha = 0.55$ and $p = q = 0.2$.

III. File name convention

We generated two types of instances, named *easy* and *hard*. Folder $\langle easy \rangle$ contains 98 instances, and folder $\langle hard \rangle$ has six instances. Instances in $\langle easy \rangle$ can be solved within 3600 seconds by sat4j [7] on a 2.0GHz quad-core Intel i7-4510U processor with 8GB RAM, while those in $\langle hard \rangle$ cannot. The name of each file follows the form $\langle \#rules - index.wcnf \rangle$, where $\#rules$ represents the number of rules.

References

[1] S. Ieong and Y. Shoham, "Marginal contribution nets: a compact representation scheme for coalitional games," In Proceedings 6th ACM Conference on Electronic Commerce, pp. 193C-202, 2005.

[2] X. Liao and M. Koshimura, "A Comparative Analysis and Improvement of MaxSAT Encodings for Coalition Structure Generation under MC-Nets," Journal of Logic and Computation, vol. 29, no. 6, pp. 913–931, 2019.

[3] X. Liao, M. Koshimura, K. Nomoto, S. Ueda, Y. Sakurai, and M. Yokoo, " Improved WPM Encoding for Coalition Structure Generation under MC-Nets," Constraints, vol. 24, no. 1, pp. 25–55, 2019.

[4] S. Ueda, T. Hasegawa, N. Hashimoto, N. Ohta, A. Iwasaki and M. Yokoo, "Handling negative value rules in MC-net-based coalition structure generation," In International Conference on Autonomous Agents and Multiagent Systems, pp. 795C804, 2012.

[5] X. Liao, M. Koshimura, H. Fujita and R. Hasegawa, "Solving the coalition structure generation problem with MaxSAT," In IEEE 24th International Conference on Tools with Artificial Intelligence, pp. 910C915, 2012.

[6] S. Ueda, A. Iwasaki, V. Conitzer, N. Ohta, Y. Sakurai, M. Yokoo, "Coalition structure generation in cooperative games with compact representations," Autonomous Agents and Multi-Agent Systems, vol. 32, pp. 503-533, 2018.

[7] Le Berre, D., and Parrain, A, "The sat4j library, release 2.2, system description," Journal on Satisfiability, Boolean Modeling and Computation, vol. 7, pp. 59-C64, 2010.

On the use of Max-SAT in RBAC maintenance: Description of Benchmarks

Marco Mori* and Marco Benedetti†
Bank of Italy
ICT Department, Centro Donato Menichella
Email: *marco.mori@bancaditalia.it, †marco.benedetti@bancaditalia.it

***Abstract*—Many organisations implement a Role-Based Access Control (RBAC) model to ease the management of permissions required to access digital resources and to enforce basic security principles, i.e., "Least Privilege" and "Separation of Duties". In this context, periodic technological and business changes, together with human errors, require a continuous and systematic revision of the assignment of permissions to users in order to incorporate missing assignments (*exceptions*) or to revoke permissions that were granted by mistake or are no longer required (*violations*).**

This note describes a set of benchmarks which encode such RBAC maintenance tasks as Weighted Partial Max-SAT (WPMS) instances, that once solved lead to optimal (w.r.t. metrics defined on the overall RBAC state) ways of *(i)* incorporating exceptions, *(ii)* excluding violations, and *(iii)* managing in a single step a set of multiple exceptions and violations.

A brief descriptions is provided to frame the problem domain along with details on the generation of the benchmark instances.

I. RBAC Maintenance

RBAC (Role-Based Access Control) model simplifies the management of permissions by defining roles which include the permissions required to execute a specific task (1). Users gain the permissions included in at least one of their assigned roles. In this context, an RBAC state can be seen as a couple of Boolean matrices: One representing the assignments of roles to users and the other representing the assignments of permissions to roles. The actual permission-to-user assignments result from the product of these two matrices.

Various causes may invalidate the set of deployed roles as described in an RBAC state: Roles may have to be revised to consider missing (exceptions) or revoked (violations) permission-to-user assignments which may follow either from technological/business changes or from possible errors.

We consider the RBAC maintenance process described in (2; 3; 4) to fix single and multiple exceptions and violations by balancing two conflicting objectives, i.e., (i) optimising the current RBAC state and (ii) reducing the transition cost with respect to the original state. In (5) the complete RBAC maintenance process has been formalised as a Weighted Partial Max-Sat (WPMS) problem and validated through an extensive set of experiments based on real-world instances. In (6), it is described how the WPMS benchmark can optimally incorporate single exceptions into an input RBAC state (7). This note extends the benchmark presented in (6) with WPMS instances aiming at excluding single violations and incorporating multiple exceptions/violations into a given RBAC state.

II. Benchmarks Description

We created our benchmarks starting from four different RBAC datasets of increasing size. The smallest one is a tiny permission-to-user assignment matrix representing a simple organization (*SmallComp*); the remaining three matrices belong to three different datasets available in the role mining literature (8), namely *Domino*, *University* and *Firewall1* (see Table I). In the remainder of this note, we describe the benchmark instances aiming at managing single exceptions/violations (Section II-A) and multiple exceptions/violations in one problem instance (Section II-B).

Dataset	Users	Permissions	Density
SmallComp	11	11	0.207
Domino	79	231	0.039
University	493	56	0.143
Firewall1	365	709	0.123

TABLE I: Original RBAC Datasets.

A. Single exceptions/violations

Starting from each of the four matrices, we generate (i) a list of randomly selected exceptions to incorporate, (ii) a list of randomly selected violations to revoke and (iii) an input RBAC state, created by applying the Fastminer algorithm (9) to the input matrix (after removing the exceptions).

Given these inputs, we generate a different WPMS instance for each single exception/violation and for twenty-one distinct values of the balancing parameter $\beta \in [0, 1]$ sampled at regular intervals (β close to 0 means "try to remain as close as possible to the origin state" and β close to 1 means "try to optimize as much as possible, regardless of the original state").

Table II.a and II.b show the number of exceptions/violations included, the values of β, and the characteristics of the Max-SAT formulas belonging to each dataset.

Naming Convention. We adopt the following naming convention: Each instance file is named by concatenating the strings "`role`" with the name of the dataset it refers to ("`smallcomp`" or "`domino`" or "`university`" or "`firewall1`") with the value of the parameter β with the index of the exception. For example, the file named:

"`role_domino_0.5_6.cnf`"

contains a WPMS instance in DIMACS format that encodes the problem of incorporating exception number *6* into the "`domino`" benchmark using *0.5* as balancing factor.

(a) Max-SAT encodings: single exceptions

Dataset	#Excs	#β	#V	#C	#C_h	#C_s
SmallComp	12	21	$6.860 \cdot 10^2$	$2.994 \cdot 10^3$	$2.629 \cdot 10^3$	$3.650 \cdot 10^2$
Domino	19	21	$7.366 \cdot 10^4$	$1.448 \cdot 10^6$	$1.424 \cdot 10^6$	$2.389 \cdot 10^4$
University	10	21	$3.238 \cdot 10^5$	$2.644 \cdot 10^6$	$2.597 \cdot 10^6$	$4.782 \cdot 10^4$
Firewall1	32	21	$1.912 \cdot 10^7$	$1.884 \cdot 10^8$	$1.888 \cdot 10^8$	$6.236 \cdot 10^5$

(b) Max-SAT encodings: single violations

Dataset	#Viols	#β	#V	#C	#C_h	#C_s
SmallComp	12	21	$6.580 \cdot 10^2$	$2.936 \cdot 10^3$	$2.571 \cdot 10^3$	$3.650 \cdot 10^2$
Domino	19	21	$7.351 \cdot 10^4$	$1.448 \cdot 10^6$	$1.424 \cdot 10^6$	$2.389 \cdot 10^4$
University	10	21	$3.237 \cdot 10^5$	$2.644 \cdot 10^6$	$2.596 \cdot 10^6$	$4.782 \cdot 10^4$
Firewall1	32	21	$1.918 \cdot 10^7$	$1.884 \cdot 10^8$	$1.878 \cdot 10^8$	$6.236 \cdot 10^5$

(c) Max-SAT encodings: multiple exceptions/violations

Dataset	#Runs	#β	#V	#C	#C_h	#C_s
SmallComp	10	11	$1.246 \cdot 10^4$	$4.928 \cdot 10^4$	$4.451 \cdot 10^4$	$4.774 \cdot 10^3$
Domino	10	11	$5.684 \cdot 10^7$	$1.117 \cdot 10^7$	$1.101 \cdot 10^7$	$1.658 \cdot 10^5$
University	10	11	$1.764 \cdot 10^6$	$1.436 \cdot 10^7$	$1.414 \cdot 10^7$	$2.169 \cdot 10^5$

TABLE II: Benchmark instances originated from single exceptions (a), single violations (b), and multiple exceptions/violations (c). $\#V$, $\#C$, $\#C_h$, and $\#C_s$ are the number of variables, clauses, hard clauses, and soft clauses in the Max-SAT encoding.

B. Multiple excpetions/violations

We generated a set of WPMS instances meant to fix several exceptions/violations in one single step, starting from the initial *SmallComp*, *Domino*, and *University* RBAC matrices.

For each dataset, we adopt a Markov chain model to generate multiple lists of exceptions and violations. According to this model, the selection of permissions which are related to one another is favoured: Either manipulating different permissions of the same user, or altering the same permission for different users. Then, for each list of exceptions and violations, we delete the exceptions from the corresponding input permission-to-user matrix and we generate an RBAC state as for the single exception/violation case (Section II-A).

Starting from such input, we generate WPMS instances for each list of exceptions/violations and for eleven values of the balancing parameter $\beta \in [0, 1]$ sampled at regular intervals.

Table II.c shows the number of exceptions/violations, the values of β and the WPMS encoding details for each dataset.

Naming Convention. Each instance is named by concatenating the string "`role`" with the name of the dataset, the string "`multiple`", the value of the balancing parameter β and, finally, the index of the exception/violation list to incorporate. For example, the file named:

"`role_university_multiple_0.1_7.cnf`"

contains a WPMS instance in DIMACS format that encodes the problem of incorporating exception/violation list number *7* into the "`university`" benchmark, with $\beta = 0.1$.

III. Acknowledgements

We thank the original creators of the role-mining datasets and Ian M. Molloy for providing us with the corresponding set of standardized *domino*, *university*, and *firewall1* matrices.

References

[1] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[2] A. Kern, M. Kuhlmann, A. Schaad, and J. Moffett, "Observations on the Role Life-cycle in the Context of Enterprise Security Management," in *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '02. ACM, 2002, pp. 43–51.

[3] H. L. Wachs, "How to succeed with role management and avoid common pitfalls," in *Gartner Database*. Research Document G00262708. Gartner, Inc., Stamford, CT 06902-7700, U.S., May 2014.

[4] B. Iverson, "Take control of enterprise role management," in *Gartner Database*. Research Document G00262285. Gartner, Inc., Stamford, CT 06902-7700, U.S., Feb 2015.

[5] M. Benedetti and M. Mori, "On the use of Max-SAT and PDDL in RBAC maintenance," *Cybersecurity*, vol. 2, no. 1, p. 19, Jul 2019.

[6] M. Benedetti and M. Mori, "Parametric RBAC Maintenance via Max-SAT: Benchmarks description," in *MaxSAT Evaluation 2019: Solver and Benchmark Descriptions*. Eds. Fahiem Bacchus and Matti Järvisalo and Ruben Martins. Series of Publications B-2019-2. Department of Computer Science, University of Helsinki. 2019.

[7] M. Benedetti and M. Mori, "Parametric RBAC Maintenance via Max-SAT," in *Proceedings of the 23rd ACM on Symposium on Access Control Models and Technologies*, ser. SACMAT '18. ACM, 2018, pp. 15–25.

[8] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo, "Evaluating Role Mining Algorithms," in *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*. ACM, 2009, pp. 95–104.

[9] J. Vaidya, V. Atluri, and J. Warner, "Roleminer: Mining Roles using Subset Enumeration," in *Proceedings of the 13th ACM Conference on Computer and Communications security*. ACM, 2006, pp. 144–153.

Automated synthesis of minimal hardware exploits with Checkmate and MaxSAT Solver

Changjian Zhang
School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
changjiz@andrew.cmu.edu

Ruben Martins
School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
rubenm@cs.cmu.edu

Marijn J.H. Heule
School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
mheule@cs.cmu.edu

Eunsuk Kang
School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
eunsukk@cs.cmu.edu

I. Introduction

This document provides descriptions for the WCNF instance files submitted to the MaxSAT Evaluation 2020, including *flush_reload_min.wcnf*, *meltdown_min.wcnf*, and *spectre_min.wcnf*.

II. Background

Alloy [1] is a modeling language based on first-order relational logic, and the Alloy Analyzer, with Kodkod [2] as its back-end solving engine, can perform fully automated analysis by bounded model checking. Given a formula and a bound on the number of atoms in the universe, the Alloy Analyzer checks whether there exists a model that satisfies the formula by translating it into a Boolean Satisfiability Problem (SAT problem).

CheckMate [3] is an automated tool based on Alloy [1] for synthesizing proof-of-concept exploit code for hardware security, such as Meltdown [4] and Spectre [5]. It uses "micro-architecturally happens-before" (μhb) graph [6] to model and analyze the execution process of micro-operations in a CPU. In a μhb graph (e.g., Figure 1d), the column refers to the instructions of one or more computer programs, and the row refers to the execution stages of an instruction in the CPU. A node is an event of an instruction at a certain stage, and a directed line represents the precedence of two events.

Figure 1 shows the overview process of CheckMate. For a given micro-architecture of a CPU, developers first create its corresponding Alloy specification, as shown in Figure 1a and 1b. An attack pattern is represented as a μhb sub-graph pattern as in Figure 1c. Then, by given the specification of a CPU and an attack pattern, CheckMate uses Alloy to synthesis a program (i.e., a μhb graph) which contains the attack pattern, as shown in Figure 1d and 1e.

In sum, CheckMate reduces the hardware exploit synthesis problem to a graph synthesis problem. It uses Alloy to model the problem, and Alloy solves the problem by using a SAT solver. According to the paper, CheckMate intends to synthesize the "smallest" graph representing a security litmus test. Security litmus tests are the most compact representation of an exploit program, meaning they contain the minimal number of operations necessary to produce the exploit pattern of interest [3].

However, SAT solvers usually do not provide guarantees on the solution of a problem. Thus, the authors of CheckMate have to enumerate all the solutions of a problem and use additional Python programs to process the results. At the same time, we are working on an extension of Alloy which allows developers to specify maximality or minimality in their Alloy problems by translating an Alloy problem into a MaxSAT problem. With this extension, developers can use Alloy to find out the optimal solutions to their problems. We use CheckMate as a case study to evaluate the practicality and performance of our extension. And *flush_reload_min.wcnf*, *meltdown_min.wcnf*, and *spectre_min.wcnf* are the MaxSAT problems generated by our extension and CheckMate in order to synthesis the security litmus tests.

III. Description of each file

Each WCNF file corresponds to a problem used in the original CheckMate evaluation [3] respectively.

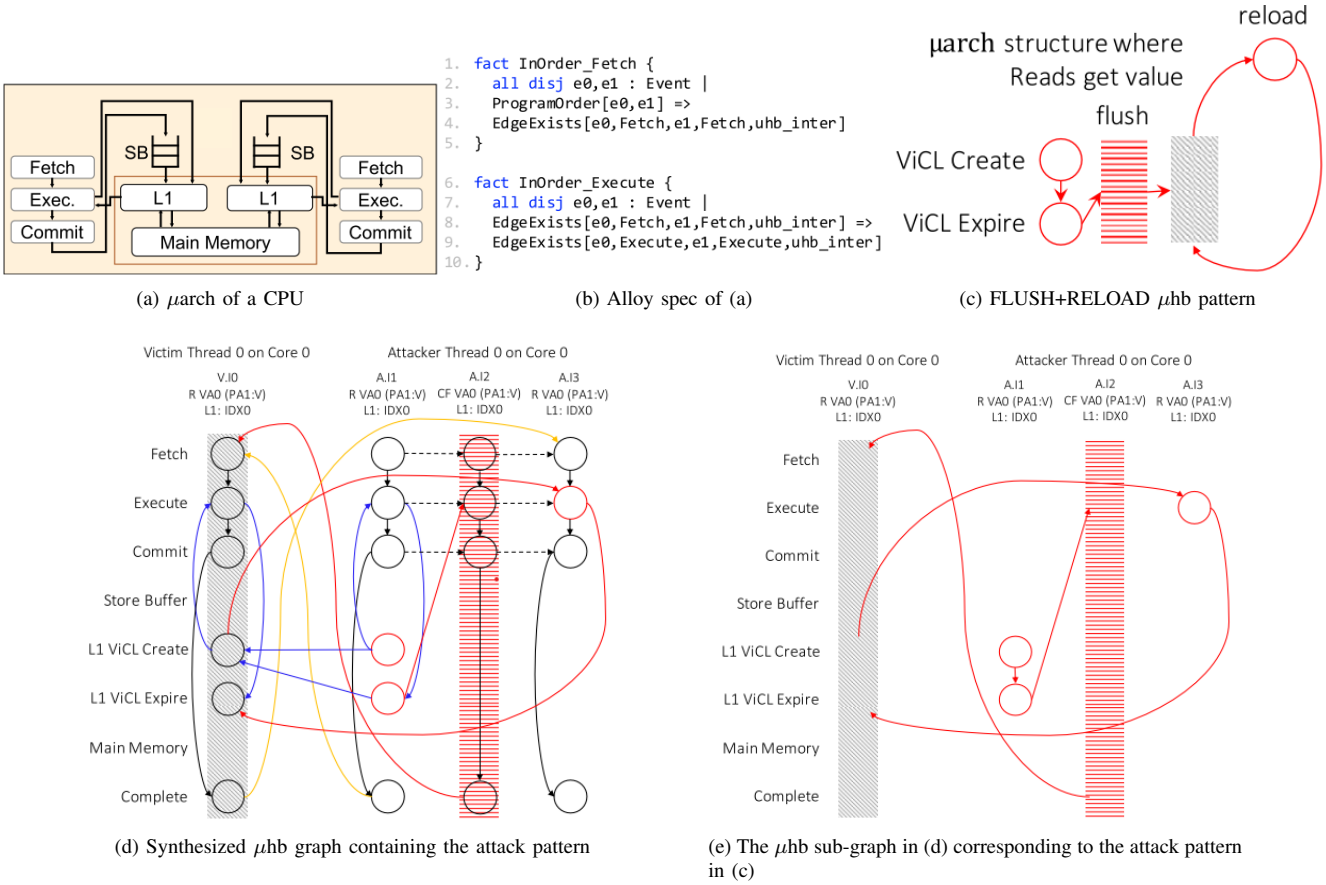
flush_reload_min.wcnf corresponds to the model with the FLUSH+RELOAD exploit pattern and 4 instructions as the bound, which should successfully produce a FLUSH+RELOAD attack. In our evaluation[1], SAT4J-MAXSAT [7] spent 3.89 minutes to solve this problem.

meltdown_min.wcnf corresponds to the model with the FLUSH+RELOAD exploit pattern and 5 instructions as the bound, which should successfully produce an attack representative of the Meltdown attack. In our evaluation, SAT4J-MAXSAT spent 25.10 minutes to solve this problem.

spectre_min.wcnf corresponds to the model with the FLUSH+RELOAD exploit pattern and 6 instructions as the bound, which should successfully produce an attack representative of the Spectre attack. In our evaluation, SAT4J-MAXSAT spent 58.75 minutes to solve this problem.

Although we can now produce the *minimal* exploit programs by using MaxSAT solvers, CheckMate still has the need to enumerate all or partial of the solutions because of that a model may produce several distinct litmus tests. Compared to enumerating the solutions in an arbitrary order, it might still be helpful for developers to enumerate the solutions in a particular

[1]We use a machine with a 6 core, 12 thread, 3.6GHz CPU and 32 GB memory to conduct the evaluation.



(a) μarch of a CPU

(b) Alloy spec of (a)

(c) FLUSH+RELOAD μhb pattern

(d) Synthesized μhb graph containing the attack pattern

(e) The μhb sub-graph in (d) corresponding to the attack pattern in (c)

Fig. 1: Overview of CheckMate [3]

order. Especially when there are too many possible solutions, developers may be able to find enough useful information from the first few iterations. However, in our evaluation, the performance for SAT4J-MAXSAT to enumerate solutions is extremely bad, especially it cannot find all the solutions for the last two problems, *meltdown_min.wcnf* and *spectre_min.wcnf*, after 24 hours.

Acknowledgment

Special thanks to Caroline Trippel for engaging with us for providing insights on the current challenges of CheckMate.

References

[1] D. Jackson, *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[2] E. Torlak and D. Jackson, "Kodkod: A relational model finder," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2007, pp. 632–647.

[3] C. Trippel, D. Lustig, and M. Martonosi, "CheckMate: Automated synthesis of hardware exploits and security litmus tests," *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, vol. 2018-Octob, pp. 947–960, 2018.

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[5] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19.

[6] D. Lustig, M. Pellauer, and M. Martonosi, "Pipecheck: Specifying and verifying microarchitectural enforcement of memory consistency models," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 635–646.

[7] D. Le Berre and A. Parrain, "The sat4j library, release 2.2, system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, pp. 59–64, 2010.

MaxSAT Benchmarks for Finding Most Compatible Phylogenetic Trees over Multi-State Characters

Tuukka Korhonen, Jeremias Berg, Matti Järvisalo
HIIT, Department of Computer Science, University of Helsinki, Finland

The benchmark set contains MaxSAT instances related to the NP-hard *maximum compatibility* problem of phylogenetic characters. In this problem the input is a set of characters (attributes) on a set of taxa (species). The task is to find the largest subset of characters that admits a *perfect phylogeny* on the taxa. A set of characters on a set of taxa admits a perfect phylogeny if there is an evolutionary tree describing the evolution of the taxa so that each character state evolves only once [9].

The maximum compatibility problem can be formulated in terms of triangulations of colored graphs [1]. Recently [6] a state-of-the-art hybrid approach combining MaxSAT and the so called Bouchitté-Todinca algorithm [2], [3] has been proposed for solving the problem. In short, the hybrid approach works by casting the maximum compatibility problem as the problem of computing a specific minimal triangulation of a colored graph G. The triangulation problem is solved by first enumerating the so called potential maximal cliques (PMCs) of G and then using them in order to form a MaxSAT instance $\mathcal{F}$ s.t. an optimal solution to $\mathcal{F}$ can be mapped into an optimal solution of the maximal compatibility problem. After some very recent improvements to PMC enumeration [5], currently the bottleneck of this hybrid approach is solving the MaxSAT instance. Hence improvements in MaxSAT solver technology on these benchmarks will directly translate to improvements in the state-of-the-art in solving maximum compatibility.

I. Problem Description

For more discussion on the maximum compatibility problem itself, we direct the reader to [9]. Here we focus on describing the MaxSAT encoding for solving the formulation of the problem in terms of minimal triangulations of undirected graphs. We assume familiarity with propositional logic, Boolean satisfiability and partial maximum satisfiability.

An undirected graph G consists of a set $V(G)$ of vertices and a set $E(G)$ of edges. The graph G is chordal if every cycle of length at least 4 has a chord, i.e. an edge joining two non-adjacent vertices. A chordal graph H is a triangulation of G if $V(H) = V(G)$ and $E(G) \subseteq E(H)$. H is a minimal triangulation of G if no triangulation H' of G has $E(H') \subset E(H)$. We denote the set of minimal triangulations of G by $\mathrm{MT}(G)$. The fill edges of a triangulation $H \in \mathrm{MT}(G)$ are all edges in the set $E(H) \setminus E(G)$. A set $S \subset V(G)$ is a clique of G if G contains an edge between each pair of vertices in S and maximal if no $S' \supset S$ is a clique (of G). A set $\Omega \subseteq V(G)$ is a potential maximal clique of G if there exists a $H \in \mathrm{MT}(G)$ such that Ω is a maximal clique of H. We denote the set of PMCs of G by $\Pi(G)$. A coloring $c\colon V(G) \to \mathbb{N}$ of G is a function that assigns a color (represented by a natural number) to each node in G s.t. no adjacent vertices of G are assigned the same color.

The MaxSAT encoding used to form the benchmarks makes use of the fact that, informally speaking, every $H \in \mathrm{MT}(G)$ can be formed by i) selecting a subset $\Pi(G)_s \subseteq \Pi(G)$ that satisfies some criteria, and ii) completing each $\Omega \in \Pi(G)_s$ into a clique, i.e. adding an edge between each pair $u, v \in \Omega$ [2].

With this, the problem that these MaxSAT benchmarks focus on can be stated as follows. Given an input graph G, a coloring $c\colon V(G) \to \mathbb{N}$ of G and $\Pi(G)$, compute a minimal triangulation $H' \in \mathrm{MT}(G)$ that minimises the number of colors with fill edges between them, i.e.

$$H' \in \operatorname*{arg\,min}_{H \in \mathrm{MT}(G)} |\{c(u) \mid \{u, v\} \in E(H) \setminus E(G) \wedge c(u) = c(v)\}|.$$

II. MaxSAT encoding

Given a graph G, the set $\Pi(G)$ and a coloring $c\colon V(G) \to \mathbb{N}$ the encoding used to make these benchmarks forms a MaxSAT instance $\mathcal{F}$ s.t. each solution τ to $\mathcal{F}$ corresponds to a minimal triangulation $H^\tau \in \mathrm{MT}(G)$ and the graph H^{τ_o} corresponding to an optimal solution τ_o of $\mathcal{F}$ minimizes the number of fill edges added between nodes of the same color.

In more detail, the central variables of $\mathcal{F}$ are $Y_{c'}$ for each color c' and P_Ω for each $\Omega \in \Pi(G)$. Given a solution τ to $\mathcal{F}$ the H^τ is formed by completing each Ω for which $\tau(P_\Omega) = 1$ into a clique. The hard clauses of $\mathcal{F}$ ensure that $H^\tau \in \mathrm{MT}(G)$ and that $\tau(Y_c) = 0$ if H^τ contains an edge between two nodes both having the color c. The soft clauses of $\mathcal{F}$ encode the optimisation criterion by including a soft clause of form (Y_c) for each color c that is satisfied only if H^τ doesn't include any fill-edges between nodes that both have the color c.

We note that the number of potential maximal cliques can be exponential in the number of vertices of the graph, and therefore the encoding can have an exponential size compared to the original input. However, this does not make solving the encoding trivial: In [6] it was shown that the maximum compatibility problem remains NP-hard even in the case when the encoding is considered to be a part of the input.

More details on the encoding can be found in [6]. An interesting aspect of the benchmarks in the context of the Evaluation is that they are all Horn MaxSAT [7].

III. Benchmark Description

This benchmark set contains 90 MaxSAT benchmarks based on graphs experimented on in [6], [5]. Out of these 89 are synthetic, generated by the ms generator (MS_XXX_YY_ZZ-T.WCNF) [4] and one corresponds to an application in Indo-European languages (INDO.WCNF) [8]. The instances generated with the ms generator each have an equal number of taxa and characters (XXX), which ranges between 50 and 400 in the instances. This is also the number of soft clauses in these instances. The number of character states (YY) ranges between 4 and 40 and the recombination parameter (ZZ/10) ranges between 1 and 4.

The instances are quite large, the number of clauses in them ranges between 5774045 and 73789633 and the number of variables between 121142 and 3343513. In spite of their size, we expect at least 60 of them to be solvable within a few hours by current state-of-the-art MaxSAT solvers.

References

[1] M. Bordewich, K. T. Huber, and C. Semple, "Identifying phylogenetic trees," *Discrete Mathematics*, vol. 300, no. 1-3, pp. 30–43, 2005.

[2] V. Bouchitté and I. Todinca, "Treewidth and minimum fill-in: Grouping the minimal separators," *SIAM Journal on Computing*, vol. 31, no. 1, pp. 212–232, 2001.

[3] ——, "Listing all potential maximal cliques of a graph," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 17–32, 2002.

[4] R. R. Hudson, "Generating samples under a Wright-Fisher neutral model of genetic variation," *Bioinformatics*, vol. 18, no. 2, pp. 337–338, 2002.

[5] T. Korhonen, "Finding optimal tree decompositions," Master's thesis, University of Helsinki, https://ethesis.helsinki.fi/en/, June 2020.

[6] T. Korhonen and M. Järvisalo, "Finding most compatible phylogenetic trees over multi-state characters," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*. AAAI Press, 2020, pp. 1544–1551.

[7] J. Marques-Silva, A. Ignatiev, and A. Morgado, "Horn maximum satisfiability: Reductions, algorithms and applications," in *Proceedings of the 18th EPIA Conference on Artificial Intelligence*, ser. Lecture Notes in Computer Science, E. C. Oliveira, J. Gama, Z. A. Vale, and H. L. Cardoso, Eds., vol. 10423. Springer, 2017, pp. 681–694.

[8] D. Ringe, T. Warnow, and A. Taylor, "Indo-European and computational cladistics," *Transactions of the Philological Society*, vol. 100, no. 1, pp. 59–129, 2002.

[9] C. Semple and M. Steel, *Phylogenetics*. Oxford University Press, 2003.

Railway Timetabling Benchmarks

Julian Reisch
Synoptics GmbH Dresden, Germany
julian.reisch@synoptics.de

Peter Großmann
Synoptics GmbH Dresden, Germany
peter.grossmann@synoptics.de

***Abstract*—We describe the railway timetabling benchmark instances submitted for the MaxSAT competition 2020. The instances are encodings from the real-world application of railway timetabling. Each instance encodes a maximum independent set (MIS) problem where the vertices of the graph correspond to slots of trains and are joint by edges if the slots have conflicts. Then, a solution of the MIS is a conflict-free timetable. The slots are constructed as columns during a column generation procedure.**

***Index Terms*—MaxSAT, benchmarks, railway timetabling**

I. Problem Description

We study the problem of scheduling all freight trains in a railway network simultaneously. That is, for each train, construct a slot such that no two slots of different trains have a conflict [2]. As there are numerous possible slots for each train, we apply a (heuristic) column generation approach where iteratively, new slots for each train are constructed [1], [3]. In each iteration of the column generation, the set of slots for each train increases and pairs of slots can have a conflict meaning that at most one of them can be part of a solution. The task is to assign to as many trains as possible one of their slots such that no two chosen slots of different trains have a conflict. Moreover, in the weighted case, each slot has a weight that reflects the ratio of the slot's traveling time to the optimal traveling time of the train. Then, the task is to assign at most one slot per train such that the sum of weights is maximized and no two conflicting slots are assigned.

II. Encoding

In each iteration of the column generation approach, we have an undirected graph $G = (V, E)$ whose vertices correspond to slots and two vertices are joint by an edge either if the two slots belong to the same train or if they have a conflict. In the weighted case, each vertex has a weight. The task is to find an independent set of maximum size (or weight, respectively) in G. Therefore, we encode each edge $\{v, w\}$ as a hard clause $\neg v \vee \neg w$ and each vertex as a soft clause v which yields an (weighted) MaxSAT instance where $v = true$ translates to vertex v being in the independent set. Then, an optimal solution of the (weighted) MaxSAT instance is an optimal solution of the (weighted) MIS.

III. Instances

In Table I, we see the instances and bounds on their optimal solution, from three runs of the column generation, one for a subnetwork, one for single day and one for multiday timetabling. The value of an optimal solution is given by the gap which is the sum (of weights) of unsatisfied soft clauses. The indices indicate the iteration number of the column generation. Hence, there are at most k many slots for each train in the k-th iteration. Table II shows the bounds on the optimal solution on the same instances where the slots are weighted. The lower and upper bounds on the optimal solutions were taken from an LP-relaxation by Gurobi and our best known solution, respectively.

TABLE I
Sizes of unweighted instances and bounds on optimal solutions.

Graph Instance	#trains	#vars	#clauses	lower bound	upper bound
Subnetwork_7	2327	10736	104111	0	6393
Subnetwork_9	2327	12317	155443	0	7903
SingleDay_2	5359	10655	46641	0	6150
SingleDay_3	5359	14175	106732	0	9448
SingleDay_{15}	5359	45761	1001983	0	25507
SingleDay_{37}	5359	87039	11150309	0	82255
MultiDay_0	56558	113116	340362	0	27643
MultiDay_1	56558	169572	1108668	0	73903
MultiDay_2	56558	215657	2312475	0	116350
MultiDay_3	56558	253930	3854064	0	152915
MultiDay_4	56558	287405	5697062	0	185424

TABLE II
Bounds on optimal solutions of weighted instances.

Graph Instance	lower bound	upper bound
Subnetwork_7_weighted	29998	43247
Subnetwork_9_weighted	64321	82885
SingleDay_2_weighted	0	12300
SingleDay_3_weighted	17571	35474
SingleDay_{15}_weighted	3450793	3647369
SingleDay_{37}_weighted	43209438	44038586
MultiDay_0_weighted	0	27643
MultiDay_1_weighted	0	147806
MultiDay_2_weighted	230156	460028
MultiDay_3_weighted	690890	1053071
MultiDay_4_weighted	1493000	2018469

References

[1] F. Dahms, D. Pöhle, A.-L. Frank, S. Kühn, "Transforming automatic scheduling in a working application for a railway infrastructure manager", Rail Norrköping Conference, 2019

[2] K. Nachtigall, J. Opitz, "Modelling and solving a train path assignment model", in Proceedings of the International Conference on Operations Research, 2014

[3] J. Reisch, P. Großmann, D. Pöhle, N. Kliewer, "Deriving application-speific column generation principles for automatic freight train timetabling", 3rd Conference on the EURO Working Group on the Practice of Operational Research, 2020

Description of Benchmarks on Single-Machine Scheduling

Xiaojuan Liao
Chengdu University of Technology
Chengdu, China
liao_xiaojuan@126.com

Miyuki Koshimura
Kyushu University
Fukuoka, Japan
koshi@inf.kyushu-u.ac.jp

I. Problem description

In real-time systems where tasks have timing requirements, once the workload exceeds the system's capacity, missed deadlines may incur system overload. In this situation, finding optimal scheduling that maximizes the number of on-time tasks is critical in both theory and practice. A real-time system is comprised of a finite set of real-time tasks $\{\tau_1, \ldots, \tau_n\}$ that are waiting to be executed. All the tasks request a uniprocessor for execution when they arrive in the system. Each task τ_j can be represented by a 3-tuple $\tau_j = (r_j, p_j, d_j)$, where j is a task's index; r_j is the release time, i.e., the earliest time at which τ_j can start; p_j is the required processing time, and d_j is the deadline, i.e., the time by which τ_j must be completed. Naturally, $r_j + p_j \leq d_j$. Each task is divided into several non-divisible fragments, between which preemption may occur. Tasks may have dependency relations. If task τ_j depends on τ_i, then τ_j can only start after τ_i is completed. If a task is finished before its deadline, it is called on-time; otherwise, it is late and worthless to the system. The scheduling objective is to maximize the number of on-time tasks in real-time systems.

Formulating the single-machine scheduling problem as partial MaxSAT has been shown to be efficient [1]. In the MaxSAT formulation, scheduling features are encoded as hard clauses and the constraint of meeting task deadlines are considered to be soft. An off-the-shelf MaxSAT solver is employed to satisfy as many deadlines as possible, provided that all the hard clauses are met.

II. Parameters used for generating the instances

The method of creating scheduling problems is summarised as follows [2]. Tasks' release times are determined according to uniform distribution with arriving rate λ, which represents the number of tasks that arrive per 100 time units. Clearly, a larger λ indicates more tasks arriving in the system during a specific period of time, thus leading to more serious overload. For each task τ_j, the processing time p_j and the number of fragments in τ_j are given. The value of deadline d_j is calculated by the formula $d_j = r_j + sf_j * p_j$, where sf_j is the slack factor that reflects the tightness of the deadline, randomly ranging from 1 to 4. The number of rule pairs with dependency relations was set 10% to the total number of tasks.

III. File name convention

We generated four different types of instances. For each type, 100 problem instances were generated. The parameter setting of each type is given as follows.

(1) Instances in folder $\langle lam20 - n500 - p13 - frag3\rangle$[1]
- *lam20:* $lambda = 20$.
- *n500:* The number of tasks to be scheduled is 500.
- *p13:* The processing time of each task is randomly generated from 1 to 13.
- *frag3:* The number of non-preemptive fragments of each task is randomly generated from 1 to 3.

(2) Instances in folder $\langle lam20 - n100 - p13 - frag12\rangle$:
- *lam20:* $lambda = 20$.
- *n100:* The number of tasks to be scheduled is 100.
- *p13:* The processing time of each task is randomly generated from 1 to 13.
- *frag12:* The number of non-preemptive fragments of each task is randomly generated from 1 to 12.

(3) Instances in folder $\langle lam100 - n100 - p13 - frag3\rangle$:
- *lam100:* $lambda = 100$.
- *n100:* The number of tasks to be scheduled is 100.
- *p13:* The processing time of each task is randomly generated from 1 to 13.
- *frag3:* The number of non-preemptive fragments of each task is randomly generated from 1 to 3.

(4) Instances in folder $\langle lam20 - n100 - p100 - frag3\rangle$:
- *lam20:* $lambda = 20$.
- *n100:* The number of tasks to be scheduled is 100.
- *p100:* The processing time of each task is randomly generated from 1 to 100.
- *frag3:* The number of non-preemptive fragments of each task is randomly generated from 1 to 3.

References

[1] X. Liao, H. Zhang, M. Koshimura, R. Huang, and W. Yu, "Maximum Satisfiability Formulation for Optimal Scheduling in Overloaded Real-Time Systems," Pacific Rim International Conference on Artificial Intelligence, pp. 618–631, 2019.

[2] Z. Chen, H. Zhang, Y. Tan, and Y. Lim, "SMT-based scheduling for overloaded real-time systems," IEICE Transactions on Informations and Systems, vol. E100-D, no. 5, pp. 1055–1066, 2017.

[1]This parameter setting is consistent with previous work [1].

Datasets of Networks for Benchmarking MaxSAT Evaluation 2020

Said Jabbour
CRIL - CNRS UMR 8188
Université d'Artois
France
jabbour@cril.fr

Nizar Mhadhbi
CRIL - CNRS UMR 8188
Université d'Artois
France
mhadhbi@cril.fr

Badran Raddaoui
SAMOVAR, Télécom SudParis
Institut Polythechnique de Paris
France
badran.raddaoui@telecom-sudparis.eu

Lakhdar Sais
CRIL - CNRS UMR 8188
Université d'Artois
France
sais@cril.fr

The analysis of large networks has become very usefull in a wide range of applications including social sciences, biology and complex systems. This paper describes a set of instances of networks for benchmarking MaxSAT Evaluation 2020. These datasets were used in [1]–[3]. All instances provided here are wcnf formulae encoded in DIMACS wcnf format.

Amazon:
This instance represents a network that was collected by crawling Amazon website. It is based on Customers Who Bought This Item Also Bought feature of the Amazon website. If a product i is frequently co-purchased with product j, the graph contains an edge from i to j.

DBLP:
The DBLP computer science bibliography provides a comprehensive list of research papers in computer science. This instance represents a co-authorship network where two authors are connected if they publish at least one paper together.

Youtube:
Youtube is a video-sharing web site that includes a social network. This instance represents the Youtube social network where users form friendship each other and users can create groups which other users can join.

Railway:
This instance represents the Indian Railway network that consists of nodes representing stations, where two stations a and b are connected by an edge if there exists at least one train-route such that both a and b are scheduled halts on that route.

Football:
This instance represents the network of American football games between Division IA colleges during regular season Fall of 2000. The vertices in the graph represent teams (identified by their college names), and edges represent regular-season games between the two teams they connect.

Karate:
This instance represents the karate social network where the data was collected from the members of a university karate club by Wayne Zachary in 1977. Each node represents a member of the club, and each edge represents a tie between two members of the club.

RiskMap:
This instance represents a graph which is a map of the popular strategy board game,Risk. It is a political map of the Earth, divided into 42 territories, which are grouped into 6 continents. Therefore, the graph is comprised of 42 vertices and 83 edges.

Politics Book:
This instances describes a network where nodes represent books about USpolitics sold by the online bookseller Amazon.com while edges represent frequent co-purchasing of books by the samebuyers on Amazon.

References

[1] S. Jabbour, N. Mahdhbi, B. Raddaoui, and L. Sais, "Triangle-Driven Community Detection in Large Graphs Using Propositional Satisfiability," 32nd IEEE International Conference on Advanced Information Networking and Applications. pp. 437–444, 2018.

[2] S. Jabbour, N. Mahdhbi, B. Raddaoui, and L. Sais, "Detecting Highly Overlapping Community Structure by Model-based Maximal Clique Expansion," IEEE International Conference on Big Data. pp. 1031–1036, 2018.

[3] S. Jabbour, N. Mahdhbi, B. Raddaoui, and L. Sais, "SAT-based models for overlapping community detection in networks," Computing. pp. 1275–1299, 2020.

Rail: Benchmark Description

1st Zhendong Lei
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
leizd@ios.ac.cn

2nd Shaowei Cai
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
caisw@ios.ac.cn

***Abstract*—In this document, we briefly describe the benchmark *Rail* submitted in MaxSAT Evaluation 2020.**

I. Problem Domain

The Set Covering Problem (SCP) are NP-hard and have many real world applications. Given an universal set X and a set Y which contains many subsets of X with $\cup_{\forall y \in Y} = X$. Each element in Y is associated with a weight $w(y)$ and the goal is to find a set $F \subseteq Y$ of the smallest total weight but still contains all elements in X, that is, $\cup_{\forall y \in F} = X$. We use $U = (X, Y, W)$ to denote a SCP instance.

II. Benchmark description

The benchmark Rail contains real-world weighted SCP instances that arise from an application in Italian railways. Rail instances are encoded into Weighted Partail MaxSAT [1] which have two significant features:

- Hard clauses only contain positive literals.
- Soft clauses are unit clauses which only contain a negative literal.

The file name convention of each instance is "rail" plus a number n where n is the number of hard clauses of each instance, e.g. "rail2536.wcnf" (2536 is the number of hard clauses of this instance).

References

[1] Z. Lei and S. Cai, "Solving set cover and dominating set via maximum satisfiability," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 1569–1576, AAAI Press, 2020.

STS: Benchmark Description

1st Zhendong Lei
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
leizd@ios.ac.cn

2nd Shaowei Cai
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
caisw@ios.ac.cn

***Abstract*—In this document, we briefly describe the benchmark *STS* submitted in MaxSAT Evaluation 2020.**

I. Problem Domain

The Set Covering Problem (SCP) are NP-hard and have many real world applications. Given an universal set X and a set Y which contains many subsets of X with $\cup_{\forall y \in Y} = X$. Each element in Y is associated with a weight $w(y)$ and the goal is to find a set $F \subseteq Y$ of the smallest total weight but still contains all elements in X, that is, $\cup_{\forall y \in F} = X$. We use $U = (X, Y, W)$ to denote a SCP instance.

II. Benchmark description

The benchmark (STS) which is from Steiner Triple Systems, contains unweighted SCP instances. STS instances are encoded into unweighted Partail MaxSAT [1] which have two significant features:

- Hard clauses only contain positive literals.
- Soft clauses are unit clauses which only contain a negative literal.

The file name convention of each instance is "data" plus a number n where n is the number of variables of each instance, e.g. "data.135.wcnf" (135 is the number of variables of this instance).

References

[1] Z. Lei and S. Cai, "Solving set cover and dominating set via maximum satisfiability," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 1569–1576, AAAI Press, 2020.

Program disambiguation using MaxSAT

Daniel Ramos*, Ines Lynce*, Vasco Manquinho*, and Ruben Martins†
* INESC-ID/Instituto Superior Técnico, Universidade de Lisboa, Portugal
{drr,ines,vmm}@sat.inesc-id.pt
† Carnegie Mellon University, USA
rubenm@cs.cmu.edu

I. Introduction

Programming-by-example [1] (PBE) is a field of program synthesis that has gained increasing interest in recent years. One of the goals of PBE is to allow people with little programming knowledge to automate tedious tasks without having to write programs. The idea is of PBE simple: the user specifies his intent using a set of input-output examples, and the program synthesizer searches for a program that maps the inputs into the outputs.

One major concern in PBE is that specifying user-intent through input-output examples usually leads to ambiguity. In other words, there might be multiple non-equivalent programs that satisfy the input-output examples the user provides. Moreover, these programs can behave very differently when provided with a different set of inputs. In the worst-case, the programs might only produce the correct output for the user-provided input-output examples. To ascertain that program synthesizers do not return spurious programs to the user, we propose two user-interaction models to disambiguate the programs that synthesizers find throughout the search process [2]. To minimize the number of user interactions, we use an unweighted MaxSAT formulation.

II. Benchmark Description

To disambiguate the programs generated by synthesizers, we propose two interaction models: Options (based on multiple-choice questions), and Yes/No (based on yes/no questions). In both interaction models, our goal is to disambiguate the programs while minimizing the number user interactions.

A. Options *Interaction Model*

In the Options interaction model, in order to minimize the number of interactions, the goal is to find a small input such that all programs provide a different output. Thus, in each interaction, the user is asked to pick the correct output for a given input.

Given a set of n programs $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$, we start by encoding the symbolic representation of all programs in SAT:

$$\phi_{\mathcal{P}_1} \wedge \ldots \wedge \phi_{\mathcal{P}_n} \tag{1}$$

We also force the inputs of the programs to be the same:

$$\forall_{i,j \in \{1..n\}, i<j} : I_i = I_j \tag{2}$$

Subsequently, for each pair of programs $\mathcal{P}_i$ and $\mathcal{P}_j$, we create a Boolean variable b_{ij} such that b_{ij} is true if and only if $\mathcal{P}_i$ and $\mathcal{P}_j$ produce the same output:

$$\forall_{i,j \in \{1..n\}, i<j} : (o_i = o_j) \Leftrightarrow (b_{ij}) \tag{3}$$

Finally, we add to the formula a soft unit clause $\neg b_{ij}$ for every variable b_{ij}. The optimal solution to this formula corresponds to the input that maximizes the pair-wise differences between the programs $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$.

B. Yes/No *Interaction Model*

In the Yes/No interaction model, the goal is to identify a small input such that the programs are split in two evenly sized sets A and B according to the following rules:

1) every program must be in either set A or B;
2) if two programs output the same, then they must be in the same set.
3) if two programs produce different outputs, at least one of them must be in set B.

After we generate such input, we can simply ask the user if the output of the desired program corresponds to the output of the programs in A. If it is, we can discard all programs in B, otherwise we discard all programs in A.

Similarly to the Options model, we start by encoding the symbolic representation of the n programs $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ in SAT. We force the programs' inputs to be equal, and create Boolean variables b_{ij} such that b_{ij} is true if and only if $\mathcal{P}_i$ and $\mathcal{P}_j$ output the same:

$$\phi_{\mathcal{P}_1} \wedge \ldots \wedge \phi_{\mathcal{P}_n} \tag{4}$$

$$\forall_{i,j \in \{1..n\}, i<j} : I_i = I_j \tag{5}$$

$$\forall_{i,j \in \{1..n\}, i<j} : (o_i = o_j) \Leftrightarrow (b_{ij}) \tag{6}$$

Additionally, for each program $\mathcal{P}_i$, we create two Boolean variables p_i^A and p_i^B, denoting if program $\mathcal{P}_i$ belongs to either set A (p_i^A is true) or B (p_i^B is true). According to the rules previously discussed, each program must be in either A and B, thus we add the following constraints:

$$\forall_{i \in \{1..n\}} : (p_i^A + p_i^B = 1) \tag{7}$$

Furthermore, if two programs output the same, they must be in the same set. Otherwise, if two programs produce different outputs, at least one of them must be in B:

$$\forall_{i,j \in \{1..n\}, i<j} : (b_{ij}) \Rightarrow \left((p_i^A \wedge p_j^A) \vee (p_i^B \wedge p_j^B)\right) \tag{8}$$

$$\forall_{i,j \in \{1..n\}, i<j} : (\neg b_{ij}) \Rightarrow (p_i^B \vee p_j^B) \tag{9}$$

Finally, our formulation is to find an input that evenly assigns the programs to the two sets. Thus, we add a set of soft constraints corresponding to the minimization of the following absolute value:

$$\left| \sum_{i=1}^{n} p_i^A - \sum_{i=1}^{n} p_i^B \right| \tag{10}$$

III. Dataset Description

Our dataset contains 195 benchmarks from real disambiguation tasks. From the 195 benchmarks, 58 are formulas generated for OPTIONS interactions, and the remaining 137 benchmarks are from YES/NO interactions.

The benchmarks correspond to interactions in the domain of table transformations. Thus, the optimal solution to each benchmark is an input table that produces the best possible user-interaction for the respective model. To make the interactions user-friendly, we bounded the number of rows of the possible input tables to 5. The benchmarks have different sizes and levels of difficulty: in some benchmarks we try to disambiguate as many as 10 programs, whereas in others we only need to distinguish between 2 programs.

IV. Acknowledgements

This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under projects UIDB/50021/2020, DSAIPA/AI/0044/2018 and project ANI 045917 financed by FEDER and FCT.

References

[1] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. *Found. Trends Program. Lang.*, 4(1-2):1–119, 2017.

[2] Daniel Ramos, Jorge Pereira, Ines Lynce, Vasco Manquinho, and Ruben Martins. Unchartit: An interactive framework for program recovery from charts. *Manuscript submitted for publication.*

MSE20 Benchmark: The inference of tumor evolutionary history from single-cell DNA sequencing data

Farid Rashidi Mehrabadi
Cancer Data Science Laboratory
Center for Cancer Research
National Cancer Institute
National Institutes of Health
Bethesda, USA
farid.rashidimehrabadi@nih.gov

Salem Malikić
Department of Computer Science
Indiana University
Bloomington, USA
salemmalikic05@gmail.com

S. Cenk Sahinalp
Cancer Data Science Laboratory
Center for Cancer Research
National Cancer Institute
National Institutes of Health
Bethesda, USA
cenk.sahinalp@nih.gov

I. Introduction

Single-cell sequencing (SCS) of DNA yields high resolution data for studying a history of tumor evolution. However, due to elevated noise rates present in the available SCS datasets, inferring tumor evolutionary history from SCS data is not a straightforward task and requires the development of sophisticated computational tools. In [1], given a binary genotype matrix obtained from SCS data of some tumor, the search for the most likely tree of tumor evolution is performed in the space of mutation trees by the use of Markov chain Monte Carlo based approach. In [2], we show that this problem can also be expressed as an instance of Constraint Satisfaction Programming and solved deterministically by utilizing the available CSP solvers. Here we present the formulation from [2] and provide 400 problem instances of varying difficulty.

II. Problem Overview

We assume that the input is given in the form of a ternary genotype matrix I with n rows (corresponding to cells) and m columns (corresponding to mutations). The set of values that $I_{i,j}$ can take consists of 0, 1 and ?. These values represent the status of mutation j in cell i as inferred by mutation calling from raw SCS data. $I_{i,j} = 0$ indicates absence of mutation j in cell i, whereas $I_{i,j} = 1$ indicates its presence. In some cases, we have insufficient information to call any of the former two states and set $I_{i,j}$ to ? (these are also known as *missing entries*). Due to technical limitations of the sequencing process and imperfect mutation calling, the matrix I usually contains false negative mutation calls, typically at the rate between 0.10 and 0.30 (i.e., between 10% and 30%). In addition, some false positive mutation calls might also be present, but usually at a lower rate (< 0.01).

Given the observed matrix I, our goal is to infer the most likely tree of evolution of the sequenced tumor. In [2] we show that finding the optimal tree can be reduced to search in the space of conflict-free matrices, which consists of all binary matrices X of size $n \times m$ such that

$$\begin{vmatrix} X_{i,a} & X_{i,b} \\ X_{j,a} & X_{j,b} \\ X_{h,a} & X_{h,b} \end{vmatrix} \neq \begin{vmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{vmatrix}.$$

for each triplet of rows (i, j, h) and each pair of columns (a, b). More precisely, we are looking for conflict free matrix Y for which $P(I \mid Y)$ is maximized, with $P(I \mid Y)$ defined as follows

$$P(I \mid Y) = \prod_{(i,j)\in\mathcal{S}} P(I_{i,j} \mid Y_{i,j}) \tag{1}$$

where $\mathcal{S}$ is set of all pairs of integers (i, j) such that $1 \leq i \leq n$, $1 \leq j \leq m$ and $I_{i,j} \in \{0, 1\}$. Individual terms $P(I_{i,j} \mid Y_{i,j})$ are computed according to the following scoring scheme

$$\begin{aligned} P(I_{i,j} = 0 \mid Y_{i,j} = 0) &= (1 - \alpha) \\ P(I_{i,j} = 0 \mid Y_{i,j} = 1) &= \beta \\ P(I_{i,j} = 1 \mid Y_{i,j} = 0) &= \alpha \\ P(I_{i,j} = 1 \mid Y_{i,j} = 1) &= (1 - \beta). \end{aligned} \tag{2}$$

where α and β denote the false positive and false negative error rates of the SCS data and are given as input parameters.

III. wMax-SAT Formulation

We will now express the problem presented in the previous section as an instance of wMax-SAT. Observe first that (2) can be rewritten in its equivalent form

$$\begin{aligned} P(I_{i,j} = 0 \mid Y_{i,j}) &= \beta \cdot \left(\frac{1-\alpha}{\beta}\right)^{1-Y_{i,j}}, \\ P(I_{i,j} = 1 \mid Y_{i,j}) &= \alpha \cdot \left(\frac{1-\beta}{\alpha}\right)^{Y_{i,j}}. \end{aligned} \tag{3}$$

Now, for each entry of the matrix $I_{i,j}$ we declare matching Boolean variable $Y_{i,j}$. For these entries of $I_{i,j}$ that are equal

to 0 we also declare Boolean variables $Z_{i,j}$ and add hard constraint which ensures that $Z_{i,j}$ is negation of $Y_{i,j}$. An example of such constraint is

$$(Y_{i,j} \vee Z_{i,j}) \wedge (\neg Y_{i,j} \vee \neg Z_{i,j}) .$$

Note that maximizing $P(I \mid Y)$ is equivalent to maximizing $\log(P(I \mid Y))$. Here, we assume that each log is taken with base 10. Combining this with equations (1) and (3) and dropping constant terms we get that this maximization is equivalent to maximizing the weight of the following "soft" constraints:

$$\text{if } I_{i,j} = 0 \text{ weight for } Z_{i,j} \text{ is: } \log \frac{1-\alpha}{\beta}$$
$$\text{if } I_{i,j} = 1 \text{ weight for } Y_{i,j} \text{ is: } \log \frac{1-\beta}{\alpha}.$$

It remains to add constraints that ensure that variables Y form a conflict free matrix. To achieve this, for each pair of mutations (p, q) and each $(a, b) \in \{(0, 1), (1, 0), (1, 1)\}$ we introduce Boolean variable $B_{p,q,a,b}$. Our aim is that variable $B_{p,q,a,b}$ is set to 1 if there exists row r such that $Y_{r,p} = a$ and $Y_{r,q} = b$. This can be enforced by adding the following constraints for all $1 \leq i \leq n$ and $1 \leq p, q \leq m$:

$$\neg Y_{i,p} \vee \neg Y_{i,q} \vee B_{p,q,1,1}$$
$$Y_{i,p} \vee \neg Y_{i,q} \vee B_{p,q,0,1}$$
$$\neg Y_{i,p} \vee Y_{i,q} \vee B_{p,q,1,0}.$$

Finally, to ensure that Y is conflict-free matrix we add the following constraint

$$\neg B_{p,q,0,1} \vee \neg B_{p,q,1,0} \vee \neg B_{p,q,1,1}.$$

We refer to [2] and [3] for more details about the latest set of constraints related to the variables $B_{p,q,a,b}$.

Values of coefficients $\log \frac{1-\alpha}{\beta}$ and $\log \frac{1-\beta}{\alpha}$ presented above are real numbers, whereas the specifications in *MaxSAT Evaluation 2020* require all coefficients to be positive integers smaller than 2^{63}. In addition, the sum of all coefficients must be smaller than $2^{64}-1$. In the practical applications, we usually have $\alpha < 0.5$ and $\beta < 0.5$ which guarantees that the values of coefficients are positive. Note that these values increase as α and β decrease. But even in the case of extremely small values of α and β, namely $\alpha = 10^{-10}$ and $\beta = 10^{-10}$ we have that $\log \frac{1-\alpha}{\beta}$ and $\log \frac{1-\beta}{\alpha}$ are both smaller than 2^4. Assuming $n \leq 500$ and $m \leq 500$, which is the case in all of the instances described below, the sum of all coefficients is not greater than

$$500 \cdot 500 \cdot 2^4 < 2^{22}.$$

Looking at the other extreme, even for unrealistically high values of noise rates, namely $\alpha = \beta = 0.4$, we have that both coefficients values are greater than 0.10.

In order to obtain instances that satisfy all requirements from *MaxSAT Evaluation 2020* and yield solutions that are identical or very close (in terms of the likelihood defined in (1)) to the optimal solution obtained when solving the instance with real weights, we multiply coefficients $\log \frac{1-\alpha}{\beta}$ and $\log \frac{1-\beta}{\alpha}$ with 2^{38} and round the obtained values to the nearest integer. Due to the previously presented inequalities, we have that each coefficient is positive integer smaller than 2^{42} and the sum of all coefficients is smaller than 2^{60}, which is well within the specified bounds.

IV. Benchmark Instances

Every instance file name contains the information about the following parameters:

- `simNo`: simulation number
- `s`: number of subclones
- `m`: number of mutations
- `n`: number of single-cells
- `fp`: false positive error rate of SCS data
- `fn`: false negative error rate of SCS data

An example filename is:

```
simNo_2-s_15-m_300-n_300-fp_0.01-fn_0.20.wcnf
```

where simulation number is 2, number of subclones is 15, number of mutations is 300, number of single-cells is 300, false positive error rate is 0.01 and false negative error rate is 0.20.

In our benchmarking dataset the value of `simNo` ranges from 1 to 10, the number of subclones is 5 or 15, false positive rate of SCS data is 0.01 or 0.0001 and false negative rate of SCS data is 0.05 or 0.20. This gives in total $10 \cdot 2 \cdot 2 \cdot 2 = 80$ distinct combinations of (`simNo`, `s`, `fp`, `fn`). For each of these combinations, (`n`,`m`) take each of the values from the set $\{(50, 50), (50, 100), (100, 100), (300, 300), (500, 100)\}$, which gives $80 \cdot 5 = 400$ instances in total. In all instances we also introduced missing entries at 0.05 rate.

The benchmark can be downloaded at https://drive.google.com/open?id=1zaKUrXbuQz1FZNNeycrlrS_iP4sZu2BX

Note that full details of generating simulated data can be found in [2] and [4]. In case you are interested in additional simulated data please contact us directly at the following email address: `salemmalikic05@gmail.com`.

For further reading, we refer to [5], where similar approach for studying tumor evolution from SCS data was presented and to [6], where the potential of SAT-solving in finding solutions to some hard problems in Computational and Systems Biology was explored.

References

[1] K. Jahn, J. Kuipers, and N. Beerenwinkel, "Tree inference for single-cell data," *Genome Biology*, vol. 17, no. 1, May 2016. [Online]. Available: http://dx.doi.org/10.1186/s13059-016-0936-x

[2] S. Malikic, F. R. Mehrabadi, S. Ciccolella, M. K. Rahman, C. Ricketts, E. Haghshenas, D. Seidman, F. Hach, I. Hajirasouliha, and S. C. Sahinalp, "PhISCS: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data," *Genome Research*, vol. 29, no. 11, p. 1860–1877, Oct. 2019. [Online]. Available: http://dx.doi.org/10.1101/gr.234435.118

[3] D. Gusfield, Y. Frid, and D. Brown, *Integer Programming Formulations and Computations Solving Phylogenetic and Population Genetic Problems with Missing or Genotypic Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 51–64. [Online]. Available: https://doi.org/10.1007/978-3-540-73545-8_8

[4] S. Malikic, K. Jahn, J. Kuipers, S. C. Sahinalp, and N. Beerenwinkel, "Integrative inference of subclonal tumour evolution from single-cell and bulk sequencing data," *Nature Communications*, vol. 10, no. 1, Jun. 2019. [Online]. Available: http://dx.doi.org/10.1038/s41467-019-10737-5

[5] E. Sadeqi Azer, F. Rashidi Mehrabadi, X. C. Li, S. Malikić, A. A. Schäffer, E. M. Gertz, C.-P. Day, E. Pérez-Guijarro, K. Marie, M. P. Lee, and et al., "PhISCS-BnB: A Fast Branch and Bound Algorithm for the Perfect Tumor Phylogeny Reconstruction Problem," *bioRxiv*, Feb. 2020. [Online]. Available: http://dx.doi.org/10.1101/2020.02.06.938043

[6] H. Brown, L. Zuo, and D. Gusfield, "Comparing Integer Linear Programming to SAT-Solving for Hard Problems in Computational and Systems Biology," *Lecture Notes in Computer Science*, p. 63–76, 2020. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-42266-0_6

Benchmarking UAQ Solvers: MAXSAT Instances

Alessandro Armando
DIBRIS
University of Genova
Genova, Italy
alessandro.armando@unige.it

Giorgia Gazzarata
DIBRIS
University of Genova
Genova, Italy
giorgia.gazzarata@dibris.unige.it

Fatih Turkmen
University of Groningen
Groningen, Netherlands
f.turkmen@rug.nl

***Abstract*—The User Authorization Query problem is an important optimization problem that arises in the context of Role-Based Access Control. Although the problem is intractable in the worst case, a number of approaches to tackle the problem have been put forward, including the reduction to MAXSAT. We propose a set of benchmarks of MAXSAT problems obtained by encoding a number of synthetically generated, yet realistic, UAQ problems of increasing complexity.**

I. Introduction/Problem overview

Role-based Access Control (RBAC) [1] is one of the most popular access control models. Instead of assigning permissions directly to subjects (e.g. applications), in RBAC permissions are assigned to roles and roles are assigned to subjects. It is widely recognized that the use of roles simplifies the definition and administration of the policy. To illustrate consider the RBAC policy where roles *Admin*, *DBReader*, *DBUpdater* are assigned permissions *WriteTape*, *ReadDB* and *WriteDB* respectively, while application *BackupApp* is assigned roles *Admin* and *DBReader* and application *WebApp* is assigned roles *DBReader* and *DBUpdater*. This policy grants *BackupApp* the permissions *ReadDB* and *WriteTape*, while it grants *WebApp* the permission *ReadDB* and *WriteDB*.

Not all roles *assigned* to a subject need to be readily available to that subject: they must be *activated* first. In the RBAC model a *session* represents the set of active roles. Thus, in a given session a subject can only use the permissions associated to the roles that are active in that session. RBAC policies may also include *Dynamic Mutually Exclusive Roles (DMER)* constraints, i.e. contraints of the form

$$DMER(rs, t)$$

stating that $\widehat{rs} = |rs|$ roles from the set of (conflicting) roles $rs = \{R_1, ..., R_{rs}\}$ cannot be simultaneously active in any session. DMER constraints are useful to enforce Separation of Duty constraints. For instance, let *App* be assigned roles *Admin*, *DBReader* and *DBUpdater*. The constraint $DMER(\{\textit{Admin}, \textit{DBUpdater}\}, 2)$ ensures that in any given session *App* cannot possibly activate roles *Admin* and *DBUpdater*, whereas $DMER(\{\textit{Admin}, \textit{DBReader}, \textit{DBUpdater}\}, 3)$ ensures that in any given session *App* can activate at most two out of the roles in the given set.

Let P_{lb} and P_{ub} be two sets of permissions such that $P_{lb} \subseteq P_{ub}$. The *User Authorization Query (UAQ) Problem* [2] is the problem of determining an optimum set of roles to activate in order to grant the subject the permissions in P_{lb}, while satisfying a given set of DMER constraints. The selected roles *may* additionally grant a subset of permissions in P_{ub}, but this is subject to the objective of either *minimizing* or *maximizing* this subset.

The UAQ problem is key for systems offering *permission level* user-system interaction (as opposed to *role level* interaction, where the user must explicitly tell the roles she wants to activate). The UAQ problem has received a growing attention in the last few years by the scientific community: the problem has been shown to be intractable in the worst case [3], yet a number of procedures have been put forward.

An encoding of the UAQ problem into MAXSAT is proposed in [4] along with experimental results obtained by using zChaff [1] as solver (following the maximal satisfaction algorithms introduced in [3] to implement the minimal and maximal satisfaction cases) with solving times in the order of seconds even for relatively simple problems. For instance, finding a minimal solution to UAQ problems with 33 roles takes more than 7 seconds on average. More recently [5] extends the encoding proposed in [4] so to support a wider class of constraints, including DMER constraints spanning over the session histories as well as over multiple sessions of the same user. The experimental results, obtained with a state-of-the-art solver, namely QMaxSAT [6], show than even large UAQ problems can be solved with ease.

Unsurprisingly, UAQ problems whose MAXSAT encodings are difficult to solve even by state-of-the-art solvers do exist. The MAXSAT instances in our porposed benchmark set have been obtained by encoding samples from four families of synthetically generated UAQ problems.

II. MAXSAT Evaluation 2020

The benchmarks submitted to MaxSAT Evaluation 2020 are the ones designed and used for the experiments in [7]. In these experiments, the benchmarks are used to compare the following solvers:

- 2D-Opt-Search [8], which is a search-based solver;
- 2D-Opt-CNF [8], which combines the reduction of the UAQ Decision Problem to SAT, a state-of-the-art SAT solver, and a binary search;
- AQUA [9], a SAT-based solver that implements a reduction of the UAQ problem to PMAXSAT and uses any state-of-the-art PMAXSAT solver to tackle the problem.

The results show that AQUA outperforms the other two solvers over the vast majority of the instances. The design of the benchmarks was guided by the methodology introduced in [7]. The methodology leverages the asymptotic complexity results provided in [10] to discriminate classes of UAQ problems that should be easy to solve (polynomial time) from the ones that should be hard to solve (exponential time). As a consequence, in case of a hard benchark (a ploynomial-time technique to solve it *is not* known):

- If the benchmark is solved in exponential time, the benchmark represents the UAQ problem complexity;
- If the benchmark is solved in polynomial time, the benchmark does not represent the UAQ problem complexity.

Otherwise, if the benchmark is easy (a ploynomial-time technique to solve it *is* known):

- If it is solved in polynomial time, the solver performs well over that class of problems;
- If it is solved in exponential time, the solver does not perform well over that class of problems.

The methodology then leads to:

- Benchmarks capable to stress-test solvers along dimensions of the problem for which no polynomial-time technique is known;
- Benchmarks capable to check the solvers effectiveness, by determining whether they efficiently solve problems that are known to be solvable in polynomial time.

The benchmarks submitted to MaxSAT Evaluation 2020 are an improvement of the ones submitted in 2018 [11], when the methodology described above was not available yet. The suite of benchmarks `uaq` consists of 23 parametric benchmarks of UAQ problems parametric in one of the following dimensions:

- $|R|$: the number of roles;
- $|P|$: the number of permissions. In the proposed benchmarks, $|P| = |P_{ub}|$. In fact, as explained in [7], it is easy to reduce any UAQ problem to an equivalent problem in which this condition holds;
- $\widehat{R_P}$: the number of roles to which each permission is assigned;
- $|C|$: the number of DMER constraints;
- $\widehat{rs}$: the number of roles involved in every DMER constraint;
- $\widehat{t}$: the bound used in every DMER constraint;
- $|P_{lb}|$: the number of permissions required by the user.

The other dimensions are fixed and set following the methodology. 11 benchmarks have the safety objective (minimization of permissions in $P_{ub} \setminus P_{lb}$) and 12 have the availability objective (maximization of permissions in $P_{ub} \setminus P_{lb}$).

The instances are named using the following convention: uaq-*family-obj-nrNR-npNP-rppRPP-nc*NC-*rsRS-tT*-plb*PLB_n*.dimacs, where:

- *obj* is the optimization objective, namely minimization (min) or maximization (max) of permissions in $P_{ub} \setminus P_{lb}$;
- *family* is the family of the problem instance, namely the benchmark parameter: `nr` for $|R|$, `np` for $|P| = |P_{ub}|$, `rpp` for $\widehat{R_P}$, `nc` for $|C|$, `rs` for $\widehat{rs}$, `t` for $\widehat{t}$, and `plb` for P_{lb};
- *NR* is the number of roles, $|R|$;
- *NP* is the number of permissions, $|P| = |P_{ub}|$;
- *RPP* is the number of roles to which each permission is assigned, $\widehat{R_P}$;
- *NC* is the number of DMER constraints, $|C|$;
- *RS* and *T* are the features of the DMER constraints, $\widehat{rs}$ and $\widehat{t}$;
- *PLB* is the number of permissions whose activation is requested, $|P_{lb}|$;
- *n* is the number identifying the instance.

The interested user can find more information in [7] and [12].

REFERENCES

[1] R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann, "Role-Based Access Control Models," *IEEE Computer*, vol. 2, no. 29, pp. 38–47, 1996.

[2] Y. Zhang and J. B. D. Joshi, "UAQ: a framework for user authorization query processing in RBAC extended with hybrid hierarchy and constraints," in *SACMAT*, 2008, pp. 83–92.

[3] L. Chen and J. Crampton, "Set covering problems in role-based access control," in *Proceedings of the 14th European conference on Research in computer security*, ser. ESORICS'09, 2009, pp. 689–704.

[4] G. T. Wickramaarachchi, W. H. Qardaji, and N. Li, "An efficient framework for user authorization queries in RBAC systems," in *SACMAT*, 2009, pp. 23–32.

[5] A. Armando, S. Ranise, F. Turkmen, and B. Crispo, "Efficient run-time solving of rbac user authorization queries: pushing the envelope," in *Second ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2012, pp. 241–248.

[6] M. Koshimura, "Qmaxsat: Q-dai maxsat solver," in *http://sites.google.com/site/qmaxsat/*, 2011.

[7] A. Armando, G. Gazzarata, and F. Turkmen, "Benchmarking uaq solvers," in *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, 2020, pp. 145–152.

[8] N. Mousavi and M. V. Tripunitara, "Mitigating the intractability of the user authorization query problem in role-based access control (rbac)," in *NSS*, 2012, pp. 516–529.

[9] A. Armando, G. A. Gazzarata, and F. Turkmen, "Aqua: An efficient solver for the user authorization query problem," in *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, 2020, pp. 153–154.

[10] N. Mousavi and M. V. Tripunitara, "Hard instances for verification problems in access control," in *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, Vienna, Austria, June 1-3, 2015*, 2015, pp. 161–164.

[11] A. Armando and G. Gazzarata, "Rbac user query authorization problem: Maxsat instances," *MaxSAT Evaluation 2018*, p. 41.

[12] G. Gazzarata, "Extensions and experimental evaluation of sat-based solvers for the uaq problem," Ph.D. dissertation, University of Genova (Italy), 5 2020. [Online]. Available: https://github.com/GioGazza/uaq_prolem/tree/master/publications

Solver Index

Benchmark Index

Author Index