# MaxSAT Evaluation 2018

## *Solver and Benchmark Descriptions*

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

University of Helsinki
Department of Computer Science
Series of Publications B
Report B-2018-2

Helsinki 2018

# 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 2018 brought on the 13th edition of the MaxSAT Evaluations. 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 2018 instantiation of the evaluation series follows closely the revised arrangements brought on by the new organization team in 2017.

The 2018 evaluation consisted of two main tracks, one for solvers focusing on unweighted and one for solvers focusing on weighted MaxSAT instances. As in 2017, no distinction was made between "industrial" and "crafted" benchmarks, and no track for purely randomly generated MaxSAT instances was organized. In addition to the main tracks, a special track for incomplete MaxSAT solvers was organized, 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 tracks.

Benchmark selection for the 2018 evaluation was done with the aim of making the benchmark set diverse and balanced. As benchmark pool, we considered: (i) all the benchmarks used in the MaxSAT Evaluation 2017, (ii) all benchmarks submitted to the MaxSAT Evaluation 2017, and (iii) all benchmarks submitted to the MaxSAT Evaluation 2018. We restricted the number of benchmarks per benchmark set to a maximum of 25 for benchmarks in categories (i) and (ii), and and a maximum of 40 for benchmarks in category (iii), and randomly picked benchmarks from each benchmark set until we had 600 benchmarks for both the unweighted and weighted tracks.

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 2017 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 2018 are collected together in this compilation.

Finally, we would like to thank everyone who contributed to MaxSAT Evaluation 2018 by submitting their solvers or new benchmarks. We are also grateful for the computational resources provided by the StarExec initiative which enabled running the 2018 evaluation smoothly.

*Fahiem Bacchus, Matti Järvisalo, & Ruben Martins*
MaxSAT Evaluation 2018 Organizers

# Contents

# SOLVER DESCRIPTIONS

# LinSBPS

Emir Demirović and Peter J. Stuckey
*Department of Computing and Information Systems*
*University of Melbourne*
Australia
(emir.demirovic ∨ pstuckey) @ unimelb.edu.au

## I. Introduction

The solver was created with the intention to study the effectiveness of local search inspired techniques for maxSAT. This is a long-term goal where the aim is to develop algorithms that use techniques similar to those of metaheuristics, in particular large neighbourhood search, but within a complete algorithmic setting. Thus, the overall objective would be to improve the *anytime* performance of solvers, which is especially important for large-scale problems where optimality guarantees seem impractical.

## II. The Algorithm

We start with the linear MaxSAT algorithm [1]. It computes the optimal solution to a maxSAT problem by repeatedly solving a series of SAT problems, each time adding constraints that force the new solution to be better than than the previously computed one. This algorithm, implemented in Open-WBO [2] with Glucose [3] as the backend solver, was the best solver for the *60 seconds unweighted incomplete track* in the last maxSAT evaluation 2017. However, it was outperformed in the same category with 300 seconds and did not provide competitive solutions for many benchmarks in the weighted incomplete track. The internal SAT solveris a complete backtracking algorithm: it selects a variable, assigns it a truth value, and then either backtracks if a conflict is found or recursively repeats the procedure.

Our approach uses the linear MaxSAT algorithm augmented with two important components: solution-based phase saving and varying resolution technique, where we start considering the problem in *low resolution* and with time increase the resolution.

### A. Solution-Based Phase Saving

The variable selection process partially mimics strategies used in local search algorithms: it selects a variable that was frequently involved in recent conflicts (*high activity*, the VSIDS scheme [4]). However, the truth value assignment procedure does not: it is based on *phase saving*, meaning it assigns the value used most recently for the variable. While phase saving is effective for pure SAT problems, *solution-based phase saving* has proven to be more efficient for optimisation [5], where the assignment is based on the best solution found so far. If the previous search was in a space where no better solution exists, time is effectively wasted with standard phase saving. Solution phase saving avoids this by searching around the best solution found. This is reminiscent of local search, as the algorithm is directed *near* the best solution. It can also be seen as a kind of Large Neighbourhood Search [6]. Indeed, assigning values to a set of variables based on the current best soluton and optimising for the remaining variables is a common strategy in metaheuristic algorithms and has been used for decades. Such a technique is particularly relevant for the incomplete track in the maxSAT competition, where solvers are expected to deliver high quality solution within tight time budgets.

To boost its performance, we incorporated solution-based phase saving in the linear algorithm. Solution-based phase saving is not widely used in MaxSAT solving. It is used by WPM3 [7] in a core-guided approach. However, we argue that the technique is more natural for a linear algorithm. As noted, the basic idea has been used in metaheuristic algorithms and even in MaxSAT solving [5] [7], but the position of the linear algorithm with solution-based phase saving among modern MaxSAT solvers is not clear. Thus, we implemented solution-based phase saving in Open-WBO [2] and evaluated its performance using benchmarks from the recent maxSAT evaluation 2017 and the international timetabling competition 2011. We do not present the results of our study in this short paper, but we do note that it provided an improvement over the baseline linear algorithm. In our recent CP paper [8], we studied solution-based phase saving for constraint programming solvers and its relation to *automated large neighbourhood search*. For CP, it provides substantial improvements. We note that we have investigated other phase saving variants, but as of now, the results remain inconclusive.

### B. Varying Resolution Approach

While solution-based phase saving does provide improvements, especially for certain classes of problems, it cannot be used effectively for a large set of the MaxSAT competition benchmarks. The reason is that the linear algorithm relies on encoding a single large cardinality constraint, which is directly dependant of the magnitude of the sum of the weights of soft clauses. As the sum grows, in the general case, so does the number of clauses and auxiliary variables that are needed to encode the cardinality constraint. Thus, the memory requirements can be significant. This has a direct impact on the performance of the linear algorithm and it some cases it completely dominates the solver. We note that this is not necessarily the case for core-guided approaches, which for a

large part are uneffected by the magnitude of the weights. Hence, we developed a simplification strategy where we initially consider the problem in *low resolution* where the weights of the MaxSAT problem are divided by a large value. After the simplified problem is solved optimally, the resolution of the problem is increased i.e. the division value is lowered. This continues iteratively until the full original problem is solved. Therefore, the technique is theoretically complete, but in practice for the benchmarks from the last MaxSAT evaluation and the short time limits, only one or two resolutions are typically considered. We note that solution-based phase saving is used during the algorithm, as well as in between resolutions. With this technique, intuitively, the most important constraints are dealt with in the beginning and with executation time other increasing important constraints are added the clause database, resembling local search style methods. It is related to the lexicographical optimisation approach for MaxSAT [9].

The main advantage is that the cardinality constraint that needs to be encoded is orders of magnitude smaller than from the original problem, offering substantial speed-ups. However, the varying resolution approach comes at the price of precision, as an optimum solution for the low-resolution problem does not necessarily correspond to the optimum for the higher resolutions and vice versa. Moreover, given two models for the low-resolution problem and their cost, it is not possible to determine which one of them is better based on their cost without consider the complete original problem. The tendency, heuristically speaking, is that better solutions to the low-resolution problem correlate with better solutions to the original problem.

## III. Conclusion

We presented LinSBPS, the algorithm we submitted for the MaxSAT Evaluation 2018. It uses a linear MaxSAT algorithm coupled with solution-based phase saving and a varying resolution approach. Our experimental results have shown that significant improvements could be achieved when compared with *maxroster*, one of the top performing solvers from the incomplete track last year. However, more detailed experimental results, such as those provided by the MaxSAT competition, are required to draw stronger conclusions.

## References


[1] 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.

[2] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a modular maxSAT solver," in *Proceedings of SAT-14*, pp. 438–445.

[3] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *Proceedings of IJCAI'09*.

[4] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of DAC'01*, pp. 530–535.

[5] I. Abío Roig, "Solving hard industrial combinatorial problems with SAT," *Ph.D. thesis, Technical University of Catalonia (UPC)*, 2013.

[6] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *Proceedings of CP'98* (M. Maher and J.-F. Puget, eds.), pp. 417–431, Springer.

[7] C. Ansótegui, F. Didier, and J. Gabàs, "Exploiting the structure of unsatisfiable cores in maxSAT," in *Proceedings of IJCAI-15*, pp. 283–289.

[8] E. Demirović and P. J. Stuckey, "Solution-based phase saving and large neighbourhood search," in *to appear in the proceedings of CP'18'*.

[9] 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.

# LMHS in MaxSAT Evaluation 2018

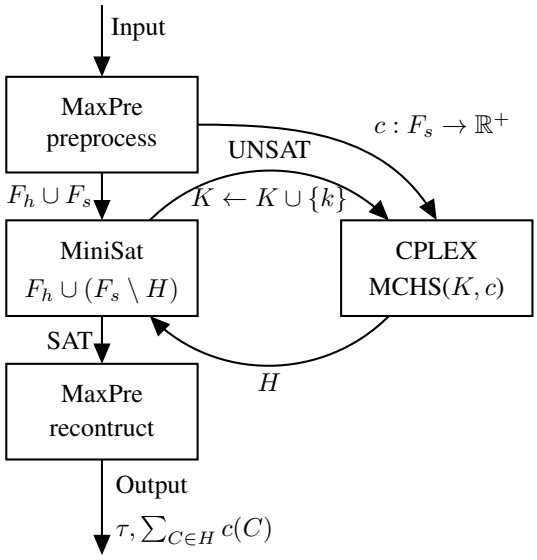
Paul Saikko and Tuukka Korhonen and Jeremias Berg and Matti Järvisalo
Department of Computer Science, HIIT
University of Helsinki, Finland

***Abstract*—We describe recent updates to the LMHS MaxSAT solver, submitted to the 2018 MaxSAT Evaluation.**

## I. Introduction

An incremental update of the LMHS MaxSAT solver [1] is submitted to the 2018 MaxSAT evaluation. Updates the the 2018 version consist of mostly bugfixes. A notable performance increase was gained by solving small MaxSAT instances (together with an initial disjoint set of cores) with the IP solver. The following description of key updates the the 2017 solver apply to the current version as well.

## II. Implicit Hitting Set Algorithm




LMHS implements the implicit hitting set algorithm [2] for MaxSAT [3], [4]. We apply MaxSAT preprocessing to simplify the problem before solving. After preprocessing, the MaxSAT cost function $c$ is input to the optimizer and the CNF formula (hard clauses $F_h$ and soft clauses $F_s$ ) is given to the satisfiability checker. MiniSat 2.2 [5] is used as the satisfiability checker, and CPLEX 12.7 [6] as the optimizer. In short, the implicit hitting set loop alternates between checking the satisfiability of the formula (excluding a hitting set $H$) to find an unsatisfiable core $k$. Unsatisfiable cores are accumulated in a set $K$, for which the optimizer finds a minimum-cost hitting set wrt. the cost function $c$. Upper bounds on the optimal solution cost (feasible solutions) are found during search LMHS's core minimization proceduce and non-optimal hitting set phase (not pictured). Lower bounds are proved by the optimizer.

## III. LCNF Preprocessing

LMHS has been updated with a new MaxSAT preprocessor, MaxPre. MaxPre implements a range of well-known and recent SAT-based preprocessing techniques as well as MaxSAT-specific techniques that make use of weights of soft clauses.

LMHS uses the MaxSAT preprocessor, MaxPre [7]. MaxPre implements a range of well-known and recent SAT-based preprocessing techniques as well as MaxSAT-specific techniques that make use of weights of soft clauses. MaxSAT specific techniques include group detection, label matching, group-subsumed label elimination, and binary core removal. Tight integration with MaxPre's C++ API eliminates unnecessary I/O overhead. LMHS solves the preprocessed instance directly as a labelled CNF formula [8], which avoids the addition of new auxiliary variables to soft clauses

## IV. Reduced-cost fixing

We implement recent reduced-cost fixing techniques for MaxSAT [9]. LP-based reduced-cost fixing together with bounds allow for some soft clauses to be hardened or relaxed during search, simplifying the problem. This inexpensive technique requires only that the LP relaxation of the hitting set IP is solved once per iteration.

## V. Availability

LMHS is open source and available at https://www.cs.helsinki.fi/group/coreo/lmhs/. MaxPre is available as a standalone preprocessor at https://www.cs.helsinki.fi/group/coreo/maxpre/.

## References

[1] P. Saikko, J. Berg, and M. Järvisalo, "LMHS: A SAT-IP hybrid MaxSAT solver," in *Proc. SAT*, ser. LNCS, vol. 9710. Springer, 2016, pp. 539–546.
[2] R. M. Karp, "Implicit hitting set problems and multi-genome alignment," in *Proc. CPM*, ser. LNCS, vol. 6129. Springer, 2010, p. 151.
[3] J. Davies and F. Bacchus, "Solving MAXSAT by solving a sequence of simpler SAT instances," in *Proc. CP*, ser. LNCS, vol. 6876. Springer, 2011, pp. 225–239.
[4] ——, "Postponing optimization to speed up MAXSAT solving," in *Proc. CP*, ser. LNCS, vol. 8124. Springer, 2013, pp. 247–262.
[5] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proc. SAT*, ser. LNCS, vol. 2919. Springer, 2003, pp. 502–518.
[6] IBM, "CPLEX Optimizer," 2017, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.
[7] T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo, "MaxPre: An extended MaxSAT preprocessor," in *Proc. SAT*, ser. LNCS, vol. 10491. Springer, 2017, pp. 449–456.
[8] J. Berg, P. Saikko, and M. Järvisalo, "Improving the effectiveness of sat-based preprocessing for maxsat," in *Proc. IJCAI*, 2015, pp. 239–245.
[9] F. Bacchus, M. Järvisalo, P. Saikko, and A. Hyttinen, "Reduced cost fixing in MaxSAT," in *Proc. CP*, ser. LNCS, vol. 10416. Springer, 2017, pp. 641–651.

# MaxHS in the 2018 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 that originated in the PhD work of Davies [4]. It was the first MaxSat solver to utilize the Implicit Hitting Set (IHS) approach, and its core components are described in [4], [2], [3], [5]. Additional useful insights into IHS are provided in [6], [7]. IHS solvers utilize both an integer programming (IP) solver and a SAT solver in a hybrid approach to MaxSat solving. 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 [2].

The 2018 version of MaxHS is unchanged from the 2017 submission, with the expection that the newest CPLEX-12.8 is being used. To make this document more self contained we repeat here the main features of v3.0, as compared to the prior published descriptions of MaxHS are as follows (familiarity with the basics of the IHS approach is assumed).

**1.0.1. Termination based on Bounding.** MaxHS v3.0 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.

**1.0.2. Early Termination of Cplex.** In previous 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.

**1.0.3. 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.

**1.0.4. 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. When all of these soft clauses have the same weight they can all be more compactly encoded with a single soft clause. This encoding does not always yield better performance due to some subtle effects. However, techniques were developed to better exploit such information, and a fuller description of these techniques is in preparation. With these techniques performance gains were achieved.

**1.0.5. 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.

**1.0.6. 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.

**1.0.7. Incomplete MaxSat Solving.** The solver maintains upper bounding models as described above, and in its normal operation it terminates only when it is able to prove that its best model is in fact optimal. However, often it is able to find very good upper bounding models or even optimal models long before termination (proving a model to be optimal is generally as hard or even harder than finding it). For the incomplete track we simply output the best model found so far at timeout.

## References


[1] Bacchus, F., Hyttinen, A., Järvisalo, M., Saikko, P.: Reduced cost fixing in maxsat. In: Proc. CP. p. in press (2017)

[2] 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)

[3] 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)

[4] 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

[5] 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)

[6] 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)

[7] 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
Email: alviano@mat.unical.it

***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).**

## 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 $\phi$ is the clause $\phi \vee \neg x$, where $x$ is a variable not occurring elsewhere; moreover, the weight associated with clause $\phi$ 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.

# MaxRoster:Solver Description

Takayuki Sugawara
Sugawara Systems
3-24-13 Kitanakayama Izumi-ku Sendai-City,Japan
nurse-support@sugawaras-systems.com

***Abstract*—In this document, we briefly describe the techniques employed by the MaxRoster solver participating in MaxSAT competition 2017.**

## I. INTRODUCTION

MaxRoster participates in Incomplete Track. MaxRoster has two engine,one is local search solver Ramp and another is MapleSAT with CHB. First, Ramp is used 6sec and then complete maxsat algorithm starts using MapleSAT. Our aim is to make feasible solution better, though it has ability of getting optimum solution.

## II. IMPLEMENTATION

*Weighted Instances:*

For weighted instances, either incremental version of OLL algorithm or model-based algorithm is used. Initially, MaxRoster makes a call to the SAT solver using solely the hard clauses. If SAT,the cost of this model represents an initial upper bound on the MaxSAT solution.The ratio of the cost mainly determines which algorithm should be invoked later.In model based algorithm, we implemented special clause counting the inputs with same weight in MapleSAT to address large and different weights for the instance.

*Unweighted Instances:*

For unweighted instances, either incremental version of MCU3 algorithm or model-based algorithm is used. Initially, MCU3 algorithm is invoked. If predefined timeout occurs in the process, then MaxRoster switches to model based algorithm dynamically.

## *References*

[1] Yi Fan, Zongjie Ma, Kaile Su, Abdul Sattar,Chengqian Li, “Ramp: A Local Search Solver based on Make-positive Variables “ MaxSAT Evaluation 2016.

[2] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, Krzysztof Czarnecki: Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. AAAI 2016: 3434-3440

[3] A. Morgado, A. Ignatiev, J. Marques-Silva: MSCG: Robust Core-Guided MaxSAT Solving. Special Issue on SAT 2014 Competitions and Evaluations. JSAT Volume 9, 2014.

[4] Martins, R., Joshi, S., Manquinho, V.M., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: CP (2014).

# Open-WBO-Inc in MaxSAT Evaluation 2018

Saurabh Joshi[‡], Prateek Kumar[‡], Vasco Manquinho[*], Ruben Martins[†], Alexander Nadel[⋆], Sukrut Rao[‡]
[‡]Indian Institute of Technology Hyderabad, India
[*]INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
[†]Carnegie Mellon University, USA
[⋆]Intel Corporation, Israel

## I. Introduction

Open-WBO-Inc is developed on top of Open-WBO [1], [2], [3], which is one of the best solvers in the MaxSAT Evaluations of 2014–2017. 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. Open-WBO-Inc starts with an incomplete stage where it is not guaranteed to converge to an optimal solution. Once this stage is completed, we switch to a complete algorithm that can further improve the solution and eventually find the optimal solution. Since Open-WBO-Inc is based on Open-WBO, it can use any MiniSAT-like solver [4]. For this evaluation we use Glucose 4.1 [5] as our back-end SAT solver.

## II. Unweighted Incomplete MaxSAT

For unweighted incomplete MaxSAT, we submitted two versions: Open-WBO-Inc-MCS and Open-WBO-Inc-OBV. The first version is based on Minimal Correction Subset (MCS) enumeration. A MCS of an unsatisfiable set of constraints is a minimal subset that, if removed, makes the constraint set satisfiable. We use a linear search algorithm [6] to enumerate MCSes. We impose a limit of 100,000 conflicts or a maximum of 30 MCSes when enumerating MCSes. Once this limit is reached or all MCSes are found, the solver will continue its search using a complete linear search algorithm SAT-UNSAT (LSU) [7] for MaxSAT starting from the best upper bound value found by MCS enumeration.

Open-WBO-Inc-OBV is based on bit-vector optimization and follows a similary strategy to the incomplete approach used in Mrs. Beaver [8]. This approach operates over a vector $\mathcal{T}$ that represents the relaxation variables introduced in each soft clause. We run 100 iterations of the following loop:

- Run the UMS-OBV-BS algorithm;
- Reverse $\mathcal{T}$. Run another UMS-OBV-BS iteration;
- Reverse $\mathcal{T}$. Run the OBV-BS algorithm;
- Reverse $\mathcal{T}$. Run another OBV-BS iteration.

At the end of each loop we randomly shuffle the vector $\mathcal{T}$. We also impose a limit of 10,000 conflicts when calling the UMS-OBV-BS and OBV-BS algorithms. For a detailed description of these algorithms we refer the reader to Mrs. Beaver paper [8]. If this algorithm terminates the incomplete stage, we continue the search by using LSU algorithm [7] for MaxSAT starting from the best upper bound value found by the bit-vector optimization stage.

To restrict the upper bound at each iteration for the LSU algorithm, we need to encode cardinality constraints into CNF. Both Open-WBO-Inc-MCS and Open-WBO-Inc-OBV versions use the Modulo Totalizer encoding [9] for cardinality constraints.

## III. Weighted Incomplete MaxSAT

For weighted incomplete MaxSAT, we submitted two versions: Open-WBO-Inc-Cluster and Open-WBO-Inc-BMO. Open-WBO-Inc-Cluster uses a technique described in [10] where it partitions the clauses in clusters and all the clauses in a cluster are given a weight equal to the representative weight of the cluster, which is a function of original weights of the clauses in the cluster. For the purpose of MaxSAT Evaluation 2018, we use arithmetic mean of the weights of clauses as representative weight of the cluster. The number of clusters is set to 2 for the purpose of this evaluation, as it is reported to strike a good balance between formula size and precision [10]. Open-WBO-Inc-Cluster uses LSU algorithm [7] with the modified weights after clustering. It uses the Generalized Totalizer Encoding (GTE) [11] to encode Pseudo-Boolean constraints that are generated to restrict weighted sum of the unsatisfied soft clauses. If an optimal solution is found for the modified MaxSAT instance, this will be an upper bound of the original MaxSAT instance. When this occurs, we revert the weights to the original weights and resume the search using the LSU algorithm starting from the best known solution.

Open-WBO-Inc-BMO version is based on bounded multi-level optimization [12] using a variant of linear search algorithm SAT-UNSAT [7] along with the partitioning of clauses as described earlier [10]. The algorithm used in Open-WBO-Inc-BMO performs optimization on each cluster in the descending order of its representative weight. This is done by performing a sequence of calls to a SAT solver and refining an upper bound $\mu$ on the number of unsatisfied soft clauses. To restrict $\mu$ at each iteration, we need to encode cardinality constraints into CNF, for which, incremental Totalizer encoding [2] has been used. Once for a given cluster the upper bound $\mu$ cannot be improved, it is frozen, and the next cluster in the order is optimized. For the purpose of MaxSAT Evaluation 2018, we set the number of clusters to the total number of different weights of the clauses of the input formula. Therefore, the

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

representative weight and the original weight remains the same in this case. As in Open-WBO-Inc-Cluster, if an optimal solution is found using this algorithm, then it is not necessarily an optimal solution of the input formula. When this occurs, we keep the best known solution and resume the search using the LSU algorithm which can potentially find better solutions and prove optimality.

## IV. Availability

We submit the source of Open-WBO-Inc as part of our submissions to the MaxSAT Evaluations 2018. The code will be later integrated into the main release of Open-WBO 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 in the MaxSAT Evaluation. We would also like to thank Mikoláš Janota, Inês Lynce and Miguel Terra-Neves for their authorship and contributions to Open-WBO on which Open-WBO-Inc is based.

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

[3] 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.

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

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

[6] J. Bailey and P. J. Stuckey, "Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization," in *PADL*. Springer, 2005, pp. 174–186.

[7] 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.

[8] A. Nadel, "Solving MaxSAT with Bit-Vector Optimization," in *SAT*. Springer, 2018.

[9] 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 *ICTAI*. IEEE, 2013, pp. 9 – 17.

[10] S. Joshi, P. Kumar, R. Martins, and S. Rao, "Approximation Strategies for Incomplete MaxSAT," in *CP*. Springer, 2018.

[11] S. Joshi, R. Martins, and V. M. Manquinho, "Generalized Totalizer Encoding for Pseudo-Boolean Constraints," in *CP*. Springer, 2015, pp. 200–209.

[12] 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.

# Open-WBO @ MaxSAT 2018

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 MaxSAT solvers [9], [10]. Open-WBO is particularly efficient for unweighted MaxSAT and has been one of the best solvers in the MaxSAT Evaluations from 2014 to 2017. Two versions of Open-WBO were submitted to the MaxSAT Evaluation 2018 (MSE2018): OPEN-WBO-RISS and OPEN-WBO-GLUC. The remainder of this document describes the MaxSAT algorithms and SAT solvers used in each version.

## II. SAT solvers

OPEN-WBO is based on the data structures of MINISAT 2.2 [9], [11]. Therefore, solvers based on MINISAT 2.2 can be used as potential backend, including formula simplification. The default SAT backend is GLUCOSE 4.1 [10], [12], which has been improved for incremental search [13]. Furthermore, formula simplification is typically disabled, as most work on incremental SAT solving with formula simplification, e.g. [14], has not been backported into MINISAT 2.2 or GLUCOSE 4.1.

Besides GLUCOSE 4.1, OPEN-WBO now supports MINISAT 2.2 and RISS [15], where MINISAT 2.2 [9] is the latest version from GitHub [11]. In this version, some data structures are different, for example the representation of the conflicting set of assumption literals. Also, the file structure changed. Both RISS and MINISAT 2.2 support reserving variables when a SAT solver is created, which allows to store them in a more compact way. Given the variety of solvers and features, we adapted OPEN-WBO to support solvers with both the old as well as the new file structure, and furthermore, allow to select whether the variable reservation feature is available during compile time. The different versions submitted to the MaxSAT Evaluation 2018 differ between themselves on the backend SAT solver. Namely, OPEN-WBO-RISS and OPEN-WBO-GLUC use RISS and GLUCOSE 4.1, respectively.

## III. MaxSAT Algorithms

In this section we briefly describe the algorithms used for the complete and incomplete tracks at the MSE2018.

### A. Complete Track

For the complete track, OPEN-WBO uses a variant of the unsatisfiability-based algorithm MSU3 [3] for unweighted problems and the OLL algorithm [7] for weighted instances. These algorithms work by iteratively refining a lower bound $\lambda$ on the number of unsatisfied soft clauses until an optimum solution is found. Both MSU3 and OLL use the Totalizer encoding for incremental MaxSAT solving [4]. For unweighted MaxSAT, we 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 MSU3 with 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), or (iii) the sparsity of the graph is too small ($<$ 0.04). For weighted MaxSAT, we use the OLL algorithm [7] without further improvements.

### B. Incomplete Track

For the incomplete track, OPEN-WBO uses a linear search algorithm SAT-UNSAT [16] with lexicographical optimization for weighted problems [17]. This algorithm works by performing a sequence of calls to a SAT solver and refining an upper bound $\mu$ on the number of unsatisfied soft clauses. To restrict $\mu$ at each iteration, we need to encode a cardinality constraint (pseudo-Boolean constraint) for unweighted (weighted) problems into CNF. The LSU version versions uses the Modulo Totalizer encoding [18] for cardinality constraints and the Adder [19] or Generalized Totalizer encoding (GTE) [20] for pseudo-Boolean constraints.

Relatively to the MSE17 version, we did the following improvements: (i) we incorporated solution-based phase saving [21], [22], and (ii) for weighted problems, we dynamically choose between the Adder encoding and the GTE encoding. We choose the former when the number of auxiliary clauses created by the GTE encoding exceeds 3,000,000.

## IV. Availability

The latest release of Open-WBO is available under a MIT license in GitHub at *https://github.com/sat-group/open-wbo*. To contact the authors please send an email to: `open-wbo@sat.inesc-id.pt`.

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

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

[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] A. Nadel, V. Ryvchin, and O. Strichman, "Ultimately incremental sat," in *SAT*, C. Sinz and U. Egly, Eds. Springer, 2014.

[15] N. Manthey. (2018, May) GitHub repository for Riss. https://github.com/conp-solutions/riss.

[16] 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.

[17] 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.

[18] 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 *ICTAI*. IEEE, 2013, pp. 9 – 17.

[19] 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.

[20] S. Joshi, R. Martins, and V. M. Manquinho, "Generalized Totalizer Encoding for Pseudo-Boolean Constraints," in *CP*. Springer, 2015, pp. 200–209.

[21] C. Ansótegui and J. Gabàs, "WPM3: An (in)complete algorithm for weighted partial MaxSAT," *Artificial Intelligence*, vol. 250, pp. 37–57, 2017.

[22] E. Demirović and P. J. Stuckey, "Local-Style Search in the Linear MaxSAT Algorithm: A Computational Study of Solution-Based Phase Saving," in *Pragmatics of SAT Workshop*, 2018.

# Pacose: An Iterative SAT-based MaxSAT Solver

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

***Abstract*—Pacose is a SAT-based MaxSAT solver using a CNF encoding for Pseudo-Boolean (PB) constraints [1]. It is an extension of the model guided QMaxSAT1702 [2] solver based on Glucose 3.0 [3] SAT solver. It uses a simple heuristic to choose between the Binary Adder [4] encoding of QMaxSAT and the Dynamic Global Polynomial Watchdog (DGPW) encoding which is based on [5].**

***Index Terms*—MaxSAT Solver, QMaxSAT, Glucose, Dynamic Global Polynomial Watchdog**

## I. TITLE

We use a new constraint encoding for PB-constraints solving the weighted MaxSAT problem with iterative SAT-based methods based on the Polynomial Watchdog (PW) CNF encoding called DGPW. The watchdog of the PW encoding indicates whether the bound of the PB constraint holds. In our approach, we lift this static watchdog concept to a dynamic one allowing an incremental convergence to the optimal result. Consequently, we formulate and implement a SAT-based algorithm for our new Dynamic Polynomial Watchdog (DPW) encoding which can be applied for solving the MaxSAT problem. Furthermore, we introduce three fundamental optimizations of the PW encoding also suited for the original version leading to significantly less encoding size.

We integrated this encoding into QMaxSAT (2nd place in the last MaxSAT Evaluation 2017) and adapt the heuristic of QMaxSAT to choose between the Binary Adder encoding of QMaxSAT and our DGPW approach.

## REFERENCES

[1] T. Paxian, S. Reimer, and B. Becker, "Dynamic polynomial watchdog encoding for solving weighted maxsat," *Theory and Applications of Satisfiability Testing–SAT 2018*, 2018.
[2] 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.
[3] G. Audemard and L. Simon, "On the glucose sat solver," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 01, p. 1840001, 2018.
[4] 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.
[5] 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.

This work is partially supported by the DFG project Algebraic Fault Attacks (funding id PO 1220/7-1, BE 1176 20/1, KR 1907/6-1).

# 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 $\phi$ 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 $\phi'$ is unsatisfiable, then $\phi$ 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 $\phi$. 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.

# RC2: a Python-based MaxSAT Solver

Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva
Faculty of Sciences, University of Lisbon, Portugal
{aignatiev,ajmorgado,jpms}@ciencias.ulisboa.pt

## 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 OLLITI (or *RC2*, i.e. *relaxable cardinality constraints*) MaxSAT algorithm [2], [3] originally implemented in the MSCG MaxSAT solver [3], [4]. The RC2 algorithm proved itself efficient in the previous editions of the MaxSAT Evaluation: 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).

## II. Description

RC2 supports *incrementally* a variety of SAT solvers provided by PySAT, and its competition version uses Glucose 3.0 [5] 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 [6], Boolean lexicographic optimization [7] and stratification [8] for weighted instances, unsatisfiable core exhaustion (originally referred to as cover optimization) [8].

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 [9]. 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 OLLITI/RC2 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.

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

## IV. 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
```

## 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, V. M. Manquinho, I. Lynce, and J. Marques-Silva, "Progression in maximum satisfiability," in *ECAI*, 2014, pp. 453–458.
[5] 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.
[6] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electr. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543–560, 2003.
[7] 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.
[8] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy, "Improving WPM2 for (weighted) partial maxsat," in *CP*, 2013, pp. 117–132.
[9] J. M. Silva, "Minimal unsatisfiability: Models, algorithms and applications (invited paper)," in *ISMVL*, 2010, pp. 9–14.

# SATLike:Solver Description

Zhendong Lei
*State Key Laboratory of Computer Science*
*Institute of Software Chinese Academy of Sciences*
Beijing, China
leizd@ios.ac.cn

Shaowei Cai
*State Key Laboratory of Computer Science*
*Institute of Software Chinese Academy of Sciences*
Beijing, China
caisw@ios.ac.cn

***Abstract*—In this document, we briefly describe the techniques employed by the SATLike solver participating in MaxSAT Evaluation 2018.**

## I. Introduction

SATLike participates in incomplete track. Our solver SATLike adopts a local search framework for SAT and does not need any specialized concept for (W)PMS. There are two main new ideas used in SATLike. The first one is a new clause weighting scheme, which works on both hard and soft clauses while at the same time takes into account the distinction between hard clauses and soft clauses. The second one is a novel variable selection heuristic, which adopts a two-mode dynamic local search framework. SATLike also uses the skill of decimation to initialize the solution [1].

## II. Implementation

SATLike is designed for solving both PMS and WPMS, and the only difference is the parameters setting. SALike is implemented in C++ and compiled by g++ with '-O3' option.

## References

[1] Shaowei Cai, Chuan Luo, Haochen Zhang: From Decimation to Local Search and Back: A New Approach to MaxSAT. IJCAI 2017: 571-577

# SATLike-c:Solver Description

Zhendong Lei
*State Key Laboratory of Computer Science*
*Institute of Software Chinese Academy of Sciences*
Beijing, China
leizd@ios.ac.cn

Shaowei Cai
*State Key Laboratory of Computer Science*
*Institute of Software 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 participating in MaxSAT Evaluation 2018.**

## I. Introduction

SATLike-c participates in incomplete track. SATLike-c has two engine, one is local search solver SATLike and another is complete sovler Open-WBO. First, SATLike is used 50sec. After that if SATLike fails to find feasible solution, it will run Open-WBO-LSU. Actually in more than 90% cases, SATLike can find feasible solution in 50sec. SATLike adopts a local search framework for SAT and does not need any specialized concept for (W)PMS. There are two main new ideas used in SATLike. The first one is a new clause weighting scheme, which works on both hard and soft clauses while at the same time takes into account the distinction between hard clauses and soft clauses. The second one is a novel variable selection heuristic, which adopts a two-mode dynamic local search framework. SATLike also uses the skill of decimation to initialize the solution [1]. The version of Open-WBO [3] is the same as MSE2017.

## II. Implementation

SATLike is designed for solving both PMS and WPMS, and the only difference is the parameters setting. SALike is implemented in C++ and compiled by g++ with '-O3' option.

## References

[1] Shaowei Cai, Chuan Luo, Haochen Zhang: From Decimation to Local Search and Back: A New Approach to MaxSAT. IJCAI 2017: 571-577
[2] Ruben Martins, Vasco M. Manquinho, Ins Lynce: Community-Based Partitioning for MaxSAT Solving. SAT 2013: 182-191
[3] Ruben Martins, Vasco M. Manquinho, Ins Lynce: Open-WBO: A Modular MaxSAT Solver, . SAT 2014: 438-445

# BENCHMARK DESCRIPTIONS

# Generalized Ising Model (Cluster Expansion) Benchmark

Wenxuan Huang[1]
[1]Department of Materials Science and Engineering
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
Key01027@mit.edu

***Abstract*— We constructed the benchmark set of generalized ising model for MAXSAT competition.**

***Keywords*— *Cluster Expansion, Ising Model, Computational Material Science***

## I. INTRODUCTION

Lattice models have wide applicability in science [1-10], and have been used in a wide range of applications, such as magnetism [11], alloy thermodynamics [12], fluid dynamics [13], phase transitions in oxides [14], and thermal conductivity [15]. A lattice model, also referred to as generalized Ising model [16] or cluster expansion [12], is the discrete representation of materials properties, e.g., formation energies, in terms of lattice sites and site interactions. In first-principles thermodynamics, lattice models take on a particularly important role as they appear naturally through a coarse graining of the partition function [17] of systems with substitutional degrees of freedom. As such, they are invaluable tools for predicting the structure and phase diagrams of crystalline solids based on a limited set of ab-initio calculations [18-22]. In particular, the ground states of a lattice model determine the 0K phase diagram of the system. However, the procedure to find and prove the exact ground state of a lattice model, defined on an arbitrary lattice with any interaction range and number of species remains an unsolved problem, with only a limited number of special-case solutions known in the literature [23-29].

In general systems, an approximation of the ground state is typically obtained from Monte Carlo simulations, which by their stochastic nature can prove neither convergence nor optimality. Thus, in light of the wide applicability of the generalized Ising model, an efficient approach to finding and proving its true ground states would not only resolve long-standing uncertainties in the field and give significant insight into the behavior of lattice models, but would also facilitate their use in ab-initio thermodynamics.

Until recently, we develop the strong links between ground state solving of cluster expansion with MAXSAT [30]. In this benchmark, we generated a Cluster expansion systems with by fitting Density Functional Theory (DFT) energies of $Li_xFe_{1-x}O_1$ systems with grid size of 5 by 5 by 5 with roughly about 100 types of interactions and try to test what is the best possible solution to the cluster expansion problem.

The general formulation of ground state problem of cluster expansion is

A lattice model is a set of fixed sites on which objects (spins, atoms of different types, atoms and vacancies, etc.) are to be distributed. Its Hamiltonian consists of coupling terms between pairs, triplets, and other groups of sites, which we refer to as "clusters". A formal definition of effective cluster interactions can be found in [12]. Before discussing the algorithmic details of our method, it is essential to establish a precise mathematical definition of a general lattice model Hamiltonian and the task of determining its ground states. The ground state problem can formally be stated as follows: Given a set of effective cluster interactions (ECI's) $J \in \mathbf{R}^{\mathbf{C}}$, where $\mathbf{C}$ is the set of interacting clusters and $\mathbf{R}$ is the set of real numbers, what is the configuration $s : \mathbf{D} \rightarrow \{0,1\}$, where $\mathbf{D}$ is the domain of configuration space, such that the global Hamiltonian $H$ is minimized:

$$\min_s H = \min_s \lim_{N\to\infty} \frac{1}{(2N+1)^3} \sum_{(i,j,k)\in\{-N,\ldots,N\}^3} \sum_{\alpha\in\mathbf{C}} J_\alpha \prod_{(x,y,z,p,t)\in\alpha} s_{i+x,j+y,k+z,p,t} \quad (1)$$

In the Hamiltonian of Eq. (1), each $\alpha \in \mathbf{C}$ is an individual interacting cluster of sites. In turn, each site within $\alpha$ is defined by a tuple $(x,y,z,p,t)$, wherein $(x,y,z)$ is the index of the primitive cell containing the interacting site, $p$ denotes the index of the sub-site to distinguish between multiple sub-lattices in that cell, and $t$ is the species occupying the site. To discretize the interactions, we introduce the ''spin'' variables $s_{x,y,z,p,t}$, where $s_{x,y,z,p,t} = 1$ indicates that the pth sub-site of the $(x,y,z)$ primitive cell is occupied by species $t$, and otherwise $s_{x,y,z,p,t} = 0$. The energy can be represented in terms of spin products, where each cluster $\alpha$ is associated with an ECI $J_\alpha$ denoting the energy associated with this particular cluster. To obtain the energy of the entire system, each cluster needs to be translated over all possible periodic images of the primitive cell, i.e., we have to consider all possible translations of the interacting cluster $\alpha$, defined as a set of $(x,y,z,p,t)$, by $(i,j,k)$ lattice primitive cells

translations, yielding the spin product $\prod_{(x,y,z,p,t)\in\alpha} s_{i+x,j+y,k+z,p,t}$.

Finally, the prefactor $\frac{1}{(2N+1)^3}$ normalizes the energy to one lattice primitive cell, and the limit of $N$ approaching infinity emphasizes our objective of minimizing the average energy over the entire infinitely large lattice. One remaining detail is that the Hamiltonian given in Eq. (1) is constrained such that that each site in the lattice must be occupied. For the sake of simplicity, lattice vacancies are included as explicit species in the Hamiltonian, so that all spin variables associated with the same site sum up to one:

$$\sum_{t\in\mathbf{c}(p)} s_{x,y,z,p,t} = 1 \;\forall (x,y,z,p)\in\mathbf{F} \qquad (2)$$

In Eq. (2), $\mathbf{F}$ is the set of all sites in the form of $(x,y,z,p)$, and $\mathbf{c}(p)$ denotes the set of species that can occupy sub-site $p$. The domain of configuration space $\mathbf{D}$ can be formally defined as the set of all $(x,y,z,p,t)$, with $t\in\mathbf{c}(p)$.

To further illustrate the notation introduced above, Figure 1 depicts an example of a two-dimensional lattice Hamiltonian for a square lattice with two sub-sites in each lattice primitive cell, i.e., $p\in\{0,1\}$. Each sub-site may be occupied by 3 types of species, so that $t\in\{0,1,2\}$, where $t=0$ shall be the reference (for example, vacancy) species. Hence, the energy of the system relative to the reference can be encoded into $t\in\{1,2\}$. Furthermore, the Hamiltonian shall be defined by only 2 different pairwise interaction types with the associated clusters $\alpha=\{(0,0,0,1,2),(1,2,0,0,1)\}$ and $\beta=\{(0,1,0,0,2),(0,0,0,1,2)\}$, and thus the set of all clusters is $\mathbf{C}=\{\alpha,\beta\}$. The first three of the five indices between "( )" brackets indicate the initial unit cell position, the forth index corresponds to the position in the unit cell (sub-site index), and the last index gives the species. The third component of the cell index (x,y,z) was retained for generality but set to 0 for this two-dimensional example. The example configuration shown in Figure 1 depicts three specific interactions: The interaction represented on the bottom left in in the figure is of type $\alpha$ with $(i,j,k)=(0,0,0)$, corresponding to the spin product $J_\alpha s_{0,0,0,1,2}\cdot s_{1,2,0,0,1}$. The interaction in the center of the figure also belongs to type $\alpha$ but with $(i,j,k)=(1,1,0)$, corresponding to the spin product $J_\alpha s_{0+1,0+1,0,1,2}\cdot s_{1+1,2+1,0,0,1} = J_\alpha s_{1,1,0,1,2}\cdot s_{2,3,0,0,1}$. Lastly, the interaction on the right represents an interacting $\beta$ cluster, with $(i,j,k)=(3,0,0)$, yielding a spin product of $J_\beta s_{0+3,1,0,0,2} s_{0+3,0,0,1,2} = J_\beta s_{3,1,0,0,2} s_{3,0,0,1,2}$.

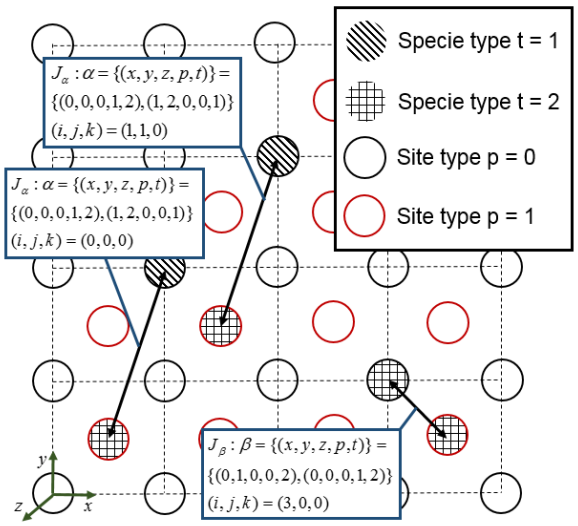



Figure 1: Illustration of a lattice Hamiltonian and examples of cluster interactions. The primitive unit of the lattice is indicated by a thin dashed line, and sites are represented by circles. Two different site types are distinguished by black and red borders, respectively. The non-vacancy species that can occupy the sites are indicated by two different hatchings.

## II. MAXSAT ENCODING

To illustrate this approach, we consider the example of a binary 1D system with a positive point term $J_0$ and a negative nearest-neighbor interaction $J_{NN}$, on a 2-site unit cell. For this system, the transformation is:

$$\begin{aligned}
E &= \min_{\hat{s}_0,\hat{s}_1}\left(J_0\hat{s}_0 + J_0\hat{s}_1 + J_{NN}\hat{s}_0\hat{s}_1\right)\\
&= -\max\left(J_0(1-\hat{s}_0) - J_0 + J_0(1-\hat{s}_1) - J_0 - J_{NN}(\hat{s}_0\hat{s}_1)\right)\\
&= -\max\left(J_0(\neg\hat{s}_0) - J_0 + J_0(\neg\hat{s}_1) - J_0 + (-J_{NN})\left((1-\neg\hat{s}_0)\hat{s}_1\right)\right)\\
&= -\max\left(J_0(\neg\hat{s}_0) - J_0 + J_0(\neg\hat{s}_1) - J_0 + (-J_{NN})\hat{s}_1 + (-J_{NN})(1-\neg\hat{s}_0\hat{s}_1) - (-J_{NN})\right)\\
&= \left(2J_0 - J_{NN}\right) - \mathrm{MAXSAT}\left(J_0(\neg\hat{s}_0)\wedge J_0(\neg\hat{s}_1)\wedge(-J_{NN})(\hat{s}_1)\wedge(-J_{NN})(\hat{s}_0\vee\neg\hat{s}_1)\right)
\end{aligned} \qquad (5)$$

where the indicator variable $\hat{s}_i$ is now also a Boolean variable in the MAX-SAT setting, and the $\wedge$, $\vee$ and $\neg$ operators correspond to logical "and", "or" and "not" respectively. Note that, although in a MAX-SAT problem the coefficient of each clause needs to be positive, it is still possible to transform an arbitrary set of cluster interactions $J_i$ into a proper MAX-SAT input, as in the example above.

The above encoding is the much much simpler version of our benchmark system. In our benchmark problems, we have many more types of interactions, for example, triplet, $J_{t1}s_0s_1s_2$ and quadruplets $J_{q1}s_0s_1s_2s_3$ etc.

## III. REFERENCE


1. Li, X., et al., *Direct visualization of the Jahn–Teller effect coupled to Na ordering in Na5/8MnO2.* Nature materials, 2014.
2. Garbulsky, G.D. and G. Ceder, *Linear-programming method for obtaining effective cluster interactions in alloys from total-energy calculations: Application to the fcc Pd-V system.* Physical Review B, 1995. **51**(1): p. 67.
3. Struck, J., et al., *Engineering Ising-XY spin-models in a triangular lattice using tunable artificial gauge fields.* Nature Physics, 2013. **9**(11): p. 738-743.
4. Aidun, C.K. and J.R. Clausen, *Lattice-Boltzmann method for complex flows.* Annual review of fluid mechanics, 2010. **42**: p. 439-472.
5. Mueller, T. and G. Ceder, *Effective interactions between the N-H bond orientations in lithium imide and a proposed ground-state structure.* Physical Review B, 2006. **74**(13): p. 134104.
6. Kremer, K. and K. Binder, *Monte Carlo simulation of lattice models for macromolecules.* Computer Physics Reports, 1988. **7**(6): p. 259-310.
7. Seko, A., et al., *Prediction of ground-state structures and order-disorder phase transitions in II-III spinel oxides: A combined cluster-expansion method and first-principles study.* Physical Review B, 2006. **73**(18): p. 184117.
8. Rothman, D.H. and S. Zaleski, *Lattice-gas cellular automata: simple models of complex hydrodynamics*. Vol. 5. 2004: Cambridge University Press.
9. van de Walle, A., *A complete representation of structure-property relationships in crystals.* Nat Mater, 2008. **7**(6): p. 455-458.
10. Van der Ven, A. and G. Ceder, *Vacancies in ordered and disordered binary alloys treated with the cluster expansion.* Physical Review B, 2005. **71**(5): p. 054102.
11. Casola, F., et al., *Direct Observation of Impurity-Induced Magnetism in a Spin-1 2 Antiferromagnetic Heisenberg Two-Leg Spin Ladder.* Physical review letters, 2010. **105**(6): p. 067203.
12. Sanchez, J.M., F. Ducastelle, and D. Gratias, *Generalized cluster description of multicomponent systems.* Physica A: Statistical Mechanics and its Applications, 1984. **128**(1): p. 334-350.
13. Frisch, U., B. Hasslacher, and Y. Pomeau, *Lattice-Gas Automata for the Navier-Stokes Equation.* Physical Review Letters, 1986. **56**(14): p. 1505-1508.
14. Li, W., J.N. Reimers, and J.R. Dahn, *Crystal structure of Li x Ni 2-x O 2 and a lattice-gas model for the order-disorder transition.* Physical Review B, 1992. **46**(6): p. 3236.
15. Chan, M.K.Y., et al., *Cluster expansion and optimization of thermal conductivity in SiGe nanowires.* Physical Review B, 2010. **81**(17): p. 174303.
16. Ising, E., *Beitrag zur Theorie des Ferromagnetismus.* Zeitschrift für Physik, 1925. **31**(1): p. 253-258.
17. Ceder, G., *A derivation of the Ising model for the computation of phase diagrams.* Computational Materials Science, 1993. **1**(2): p. 144-150.
18. Hinuma, Y., Y.S. Meng, and G. Ceder, *Temperature-concentration phase diagram of P 2-Na x CoO 2 from first-principles calculations.* Physical Review B, 2008. **77**(22): p. 224111.
19. Ozoliņš, V., C. Wolverton, and A. Zunger, *Cu-Au, Ag-Au, Cu-Ag, and Ni-Au intermetallics: First-principles study of temperature-composition phase diagrams and structures.* Physical Review B, 1998. **57**(11): p. 6427.
20. Asta, M. and V. Ozoliņš, *Structural, vibrational, and thermodynamic properties of Al-Sc alloys and intermetallic compounds.* Physical Review B, 2001. **64**(9): p. 094104.
21. Burton, B.P. and A. van de Walle, *First principles phase diagram calculations for the octahedral-interstitial system.* Calphad, 2012. **37**(0): p. 151-157.
22. Zhou, F., T. Maxisch, and G. Ceder, *Configurational electronic entropy and the phase diagram of mixed-valence oxides: The case of Li x FePO 4.* Physical review letters, 2006. **97**(15): p. 155704.
23. Dublenych, Y.I., *Ground states of the Ising model on the Shastry-Sutherland lattice and the origin of the fractional magnetization plateaus in rare-earth-metal tetraborides.* Phys Rev Lett, 2012. **109**(16): p. 167202.
24. Dublenych, Y.I., *Ground states of the lattice-gas model on the triangular lattice with nearest- and next-nearest-neighbor pairwise interactions and with three-particle interaction: Full-dimensional ground states.* Physical Review E, 2011. **84**(1).
25. Dublenych, Y.I., *Ground states of the lattice-gas model on the triangular lattice with nearest- and next-nearest-neighbor pairwise interactions and with three-particle interaction: Ground states at boundaries of full-dimensional regions.* Physical Review E, 2011. **84**(6): p. 061102.
26. Teubner, M., *Ground states of classical one-dimensional lattice models.* Physica A: Statistical Mechanics and its Applications, 1990. **169**(3): p. 407-420.
27. Kanamori, J. and M. Kaburagi, *Exact Ground States of the Lattice Gas and the Ising Model on the Square Lattice.* Journal of the Physical Society of Japan, 1983. **52**(12): p. 4184-4191.
28. Kaburagi, M. and J. Kanamori, *Ground State Structure of Triangular Lattice Gas Model with up to 3rd Neighbor Interactions.* Journal of the Physical Society of Japan, 1978. **44**(3): p. 718-727.
29. Finel, A. and F. Ducastelle, *On the phase diagram of the FCC Ising model with antiferromagnetic first-*

*neighbour interactions.* EPL (Europhysics Letters), 1986. **1**(3): p. 135.
30. Huang, W., et al., *Finding and proving the exact ground state of a generalized Ising model by convex optimization and MAX-SAT.* Physical Review B, 2016. **94**(13): p. 134424.

# MSE18 Benchmarks: DRMX-AtMostK

Alexey Ignatiev
Faculty of Sciences, University of Lisbon, Portugal
aignatiev@ciencias.ulisboa.pt

## I. Motivation

This benchmark set is motivated by the recent work on dual-rail based MaxSAT solving proposed and described in [1], [2] and further extended in [3]. The idea of [1], [2] enables reducing SAT into partial MaxSAT by transforming an arbitrary formula in conjunctive normal form (CNF) into a partial MaxSAT formula. The transformation used is a variant of the *dual-rail encoding* (*DRE*) known at least since [4] and applied in a number of practical settings. As shown in [1], applying the DRE and core-guided MaxSAT solving can tackle problems that are known to be hard for general resolution. More concretely, [1] studied the *pigeonhole principle* (*PHP*) formulas and showed that the dual-rail encoding followed by core-guided MaxSAT solving (or MaxSAT resolution [5]) can refute PHP formulas in polynomial time. Moreover, a similar result was shown to hold for the *doubled pigeonhole principle* (*2PHP*), which is a more general form of the pigeonhole principle known to be even harder than PHP.

## II. Description

The key idea of the polynomial time core-guided MaxSAT-based refutation of both PHP and 2PHP formulas is to divide the dual-rail encoded MaxSAT formula into individual parts that can be handled separately. More concretely, a PHP formula can be represented as a conjunction of disjoint AtLeast1 and AtMost1 constraints[1], which are unsatisfiable if conjoined together. As a result, each AtLeast1 and AtMost1 constraint together with the corresponding soft clauses of the DRE is studied separately in [1] when constructing a polynomial time procedure to refute PHP. Similarly, 2PHP can be seen as a conjunction of disjoint AtLeast1 and AtMost2 constraints that are handled separately in [3]. Moreover, a similar observation can be made for other classes of formulas conventionally studied in the area of propositional proof complexity, e.g. the renowned *parity principle* [3] also includes AtMost1 constraints. Also, generalizing these principles results in AtMostK constraints to be considered.

Surprisingly, when analyzing the PHP and Parity formulas, we noticed that dealing with only AtMostK constraints together with the corresponding soft clauses of the DRE can already be challenging for MaxSAT solvers. As a result, the DRMX-AtMostK benchmark set contains trivially constructed partial MaxSAT formulas $\mathcal{F}(m,k) = \mathcal{H}_{m,k} \wedge \mathcal{S}_m$, where $\mathcal{H}(m,k)$ is a CNF representation of a cardinality constraint $\sum_{i=1}^{m} x_i \leq k$ while $\mathcal{S}_m$ is a set of unit-size soft clauses, i.e. $\mathcal{S}_m = \{(x_1), \ldots, (x_m)\}$. Set $\mathcal{H}_{m,k}$ is a set of hard clauses.

The MaxSAT cost of each instance $\mathcal{F}(m,k)$ is equal to $m-k$. The instances of the DRMX-AtMostK benchmark set are constructed by varying the value of $m$ and $k$ and considering different kinds of cardinality encodings. The following pairs of $(m,k)$ are considered: $(40,12)$, $(45,16)$, $(50,20)$, $(55,24)$, $(60,28)$, $(70,32)$. Cardinality encodings used to represent the hard part $\mathcal{H}_{m,k}$ are sequential counters [6], sorting networks [7], cardinality networks [8], totalizer [9], modulo totalizer [10], and $k$-cardinality modulo totalizer [11]. Note that the hardness of the DRMX-AtMostK formulas (and so the performance of MaxSAT solvers) depends not only on the parameters $m$ and $k$ but also the cardinality encoding used.

## III. Weighted Variant

Additionally to the unweighted instances described above, their weighted variant was created: (1) all clauses of $\mathcal{S}_m$ have weight 1, (2) all clauses of $\mathcal{H}_{m,k}$ are not hard anymore but have weight $m+1$ instead. As a result, the weighted instances exhibit the Boolean lexicographic optimization property [12].

[1] Recall that an AtLeastK constraint (AtMostK constraint, resp.) is a constraint of the form $\sum_{i=1}^{m} x_i \geq K$ ($\sum_{i=1}^{m} x_i \leq K$, resp.) given some $m > K$ and s.t. each $x_i \in \{0,1\}$.

## References

[1] A. Ignatiev, A. Morgado, and J. Marques-Silva, "On tackling the limits of resolution in SAT solving," in *SAT*, 2017, pp. 164–183.
[2] J. Marques-Silva, A. Ignatiev, and A. Morgado, "Horn maximum satisfiability: Reductions, algorithms and applications," in *EPIA*, 2017, pp. 681–694.
[3] M. L. Bonet, S. Buss, A. Ignatiev, J. Marques-Silva, and A. Morgado, "MaxSAT resolution with the dual rail encoding," in *AAAI*, 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16782
[4] R. E. Bryant, D. L. Beatty, K. S. Brace, K. Cho, and T. J. Sheffler, "COSMOS: A compiled simulator for MOS circuits," in *DAC*, 1987, pp. 9–16.
[5] M. L. Bonet, J. Levy, and F. Manyà, "Resolution for Max-SAT," *Artif. Intell.*, vol. 171, no. 8-9, pp. 606–618, 2007.
[6] C. Sinz, "Towards an optimal CNF encoding of Boolean cardinality constraints," in *CP*, 2005, pp. 827–831.
[7] K. E. Batcher, "Sorting networks and their applications," in *AFIPS Conference*, 1968, pp. 307–314.
[8] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, "Cardinality networks and their applications," in *SAT*, 2009, pp. 167–180.
[9] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of Boolean cardinality constraints," in *CP*, 2003, pp. 108–122.
[10] 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 *ICTAI*, 2013, pp. 9–17.
[11] A. Morgado, A. Ignatiev, and J. Marques-Silva, "MSCG: Robust core-guided MaxSAT solving," *JSAT*, vol. 9, pp. 129–134, 2015.
[12] 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.

# MSE18 Benchmarks: DRMX-CryptoGen

Alexey Ignatiev
University of Lisbon, Portugal
aignatiev@ciencias.ulisboa.pt

Oleg Zaikin
ISDCT SB RAS, Irkutsk, Russia
zaikin.icc@gmail.com

## I. Introduction

Recent work on dual-rail based MaxSAT solving [1], [2] proposed a simple formula transformation, which given a propositional formula in conjunctive normal form (CNF) creates a partial CNF formula containing solely Horn clauses. This formula transformation is referred to as *dual-rail encoding* (*DRE*) to MaxSAT and follows the ideas of the original dual-rail encoding known at least since [3] and applied in a number of practical settings. The transformation enables one to solve SAT with the use of the state-of-the-art MaxSAT solvers. Moreover, as shown in [4], MaxSAT resolution applied to the dual-rail encoded MaxSAT formulas is strictly stronger than general resolution, which gives hope of devising generic solutions for SAT that would be practically more efficient than *conflict-driven clause learning* (*CDCL*) since the latter is known [5] to be as powerful as resolution. The DRMX-CryptoGen benchmark set comprises partial and weighted partial benchmark instances created using the DRE of CNF formulas encoding cryptanalysis of a few stream cipher generators, namely Geffe [6], Threshold [7], and Wolfram [8] generators.

## II. Dual-rail Encoding

Given a CNF formula $\mathcal{F}$ over $N$ variables $X = \{x_1, \ldots, x_N\}$, the dual-rail MaxSAT encoding [1], [2], [4] creates a (Horn) MaxSAT problem $\langle \mathcal{S}, \mathcal{H} \rangle$, where $\mathcal{H}$ is the set of hard clauses and $\mathcal{S}$ is the set of soft clauses s.t. $|\mathcal{S}| = 2N$. Each variable $x_i \in X$ is encoded by a distinct pair of variables $p_i$ and $n_i$ s.t. $p_i$ is true iff $x_i$ is true while $n_i$ is true iff $x_i$ is false. Moreover, for each variable $x_i \in X$ the DRE creates a pair of unit soft clauses, i.e. $\mathcal{S} = \{(p_i), (n_i) \,|\, x_i \in X\}$. Additionally, a new hard clause $(\neg p_i \vee \neg n_i)$ is added to $\mathcal{H}$. Each clause $c$ of $\mathcal{F}$ is encoded into a hard clause $c' \in \mathcal{H}$ s.t. $c' = \{\neg p_i \,|\, \neg x_i \in c\} \cup \{\neg n_j \,|\, x_j \in c\}$. It holds [1] that $\mathcal{F}$ is satisfiable iff there exists an assignment that satisfies $\mathcal{H}$ and at least $N$ clauses in $\mathcal{S}$.

## III. Generators Considered

The DRE procedure was applied to CNF instances encoding cryptanalysis of a few stream cipher generators. In order to make the benchmarks more challenging, the original CNF instances are guaranteed to have exactly one model. All instances were created with the use of an automated encoder TRANSALG [9]. The parameters of the generators are selected such that the SAT formulas are relatively easy to solve by a modern SAT solver [10]. However, the dual-rail encoded MaxSAT instances are expected to be harder to deal with using a MaxSAT solver. The following description of the used generators assumes some familiarity with the basic concepts used in cryptography, e.g. *linear feedback shift register* (*LFSR*) [11].

### A. Geffe Generator

We considered the Geffe generator [6] based on LFSRs with the following primitive polynomials:

- LFSR 1: $X^{38} + X^{41} + 1$;
- LFSR 2: $X^{37} + X^{38} + X^{42} + X^{43} + 1$;
- LFSR 3: $X^{17} + X^{18} + X^{43} + X^{44} + 1$;

The following cryptanalysis problem was encoded into SAT: given the first 200 keystream bits produced by the generator, it is required to find a secret key of 128 bits, i.e. the initial state of the generator's registers, that "matches" this keystream.

### B. Threshold Generator

The Threshold generator [7] considered is based on the same 3 LFSRs as those of the Geffe generator described above. And again, the cryptanalysis problem encoded is to find a secret key of size 128 given the first 200 bits of a known keystream.

### C. Wolfram Generator

The Wolfram generator [8] is based on a one-dimensional cellular automaton [12]. Two cryptanalysis problems were constructed: to find a secret key of size 72 (80, resp.) given the first 144 (160, resp.) bits of a known keystream.

All the constructed CNF instances encoding the cryptanalysis of the considered generators can be reproduced using the corresponding algorithmic implementations of the generators written for the TRANSALG encoder [13]. Regarding the Geffe and Threshold generators, one should replace the primitive polynomials with the ones described above and change the size of the keystream. In the case of the Wolfram generator, the size of the secret key and the keystream should be changed.

## IV. Weighted Variant

Additionally to the unweighted instances described above, their weighted variant was created: (1) all clauses of $\mathcal{S}$ have weight 1, (2) all clauses of $\mathcal{H}$ are not hard anymore but have weight $2N + 1$ instead. As a result, the weighted instances exhibit the Boolean lexicographic optimization property [14].

## REFERENCES


[1] A. Ignatiev, A. Morgado, and J. Marques-Silva, "On tackling the limits of resolution in SAT solving," in *SAT*, 2017, pp. 164–183.

[2] J. Marques-Silva, A. Ignatiev, and A. Morgado, "Horn maximum satisfiability: Reductions, algorithms and applications," in *EPIA*, 2017, pp. 681–694.

[3] R. E. Bryant, D. L. Beatty, K. S. Brace, K. Cho, and T. J. Sheffler, "COSMOS: A compiled simulator for MOS circuits," in *DAC*, 1987, pp. 9–16.

[4] M. L. Bonet, S. Buss, A. Ignatiev, J. Marques-Silva, and A. Morgado, "MaxSAT resolution with the dual rail encoding," in *AAAI*, 2018. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16782

[5] P. Beame, H. A. Kautz, and A. Sabharwal, "Towards understanding and harnessing the potential of clause learning," *J. Artif. Intell. Res.*, vol. 22, pp. 319–351, 2004.

[6] P. Geffe, "How to protect data with ciphers that are really hard to break," *Electronics*, vol. 46, no. 1, pp. 99–101, Jan. 1973.

[7] J. O. Bruer, "On pseudo random sequences as crypto generators," in *Proc. International Zurich Seminar on Digital Communication, Switzerland*, 1984, pp. 157–161.

[8] S. Wolfram, "Random sequence generation by cellular automata," *Advances in Applied Mathematics*, vol. 7, no. 2, pp. 123 – 169, 1986.

[9] I. Otpuschennikov, A. Semenov, I. Gribanova, O. Zaikin, and S. Kochemazov, "Encoding cryptographic functions to SAT using Transalg system," in *ECAI*, 2016, pp. 1594–1595.

[10] 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.

[11] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[12] J. von Neumann, "The general and logical theory of automata," in *Cerebral Mechanisms in Behavior – The Hixon Symposium*, L. A. Jeffress, Ed. John Wiley & Sons, 1951, pp. 1–31.

[13] I. Otpuschennikov, "Programs for the Transalg SAT encoder. URL: https://gitlab.com/satencodings/satencodings." [Online]. Available: https://gitlab.com/satencodings/satencodings

[14] 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.

# MaxSAT Benchmarks: Maximum Realizability for Linear Temporal Logic Specifications

Rayna Dimitrova
University of Leicester
rd307@leicester.ac.uk

Mahsa Ghasemi
The University of Texas at Austin
mahsa.ghasemi@utexas.edu

Ufuk Topcu
The University of Texas at Austin
utopcu@utexas.edu

## I. Problem Overview

Linear-time temporal logic (LTL) [4] is a standard specification language for formalizing requirements on the behaviour of reactive systems. The *synthesis problem for LTL* asks to automatically construct a system (i.e., an implementation) that satisfies a given LTL formula, or determine that such a system does not exist, i.e., the specification is *unrealizable*.

In [1] we studied the synthesis problem in settings where the overall specification is unrealizable, more precisely, when some of the desirable properties have to be (temporarily) violated in order to satisfy the system's objective. In such cases it is desirable that the synthesis procedure provides a "best-effort" implementation, either according to some user-given criteria, or according to the semantics of the specification language. This motivates the development of synthesis methods for *maximum realizability*, where the input to the synthesis tool consists of a *hard specification* which *must* be satisfied, and *soft specifications* which describe other desired, possibly prioritized properties. More precisely, we considered hard specifications given as LTL formulas and soft specifications of the form $\square\varphi_1, \ldots, \square\varphi_n$, where $\square$ is the LTL "globally" operator, and each $\varphi_i$ is a syntactically safe LTL formula.

In order to give a formal meaning to the notion of "best-effort" implementation, we defined a quantitative semantics for soft specifications, which accounts for how often each $\varphi_i$ is satisfied. In particular, we considered truth values corresponding to $\varphi_i$ being satisfied at every point of an execution, being violated only finitely many times, being both violated and satisfied infinitely often, or being continuously violated from some point on. Based on this semantics, we defined the numerical value $val(\mathcal{S}, \square\varphi_1 \wedge \ldots \wedge \square\varphi_n)$ of a conjunction $\square\varphi_1 \wedge \ldots \wedge \square\varphi_n$ of soft specifications in a given implementation $\mathcal{S}$. We proposed a method for synthesizing an implementation that maximizes the value of the soft specifications, based on a reduction to partial weighted MaxSAT.

We showed that the maximum realizability problem that we studied can be reduced to its bounded version where the size of the sought implementation is bounded from above by some constant. In turn, the bounded maximum realizability problem can be reduced to a partial weighted MaxSAT problem.

Formally, the *bounded maximum realizability problem* asks, given an LTL formula $\varphi$ and formulas $\square\varphi_1, \ldots, \square\varphi_n$, where each $\varphi_i$ is a syntactically safe LTL formula, and a bound $b \in \mathbb{N}_{>0}$, to synthesize a system $\mathcal{S}$ with at most $b$ states that satisfies $\varphi$ and such is that $val(\mathcal{S}, \square\varphi_1 \wedge \ldots \wedge \square\varphi_n)$ is maximal among the systems of size not exceeding $b$ satisfying $\varphi$.

## II. MaxSAT Encoding

Our reduction to MaxSAT is similar to the SAT-based technique for bounded synthesis [3]. Similarly to [3], LTL specifications are translated to automata, which are then used to construct a constraint system that encodes the existence of an implementation with the desired properties.

*1) Variables:* The MaxSAT formulation, similarly to [2] has variables that represent the sought transition system, and variables that represent *annotations*, which are required to witness the satisfaction of the LTL specifications.

*2) Hard constraints for valid annotations:* The hard clauses in the MaxSAT formulation express the requirements for the annotation corresponding to the hard LTL specification, as well a part of the requirements for each annotation for a soft LTL specification. The remaining part of the requirements for the soft specifications are expressed by the soft clauses.

*3) Soft constraints for valid annotations:* For each soft specification $\square\varphi_i$ there are three soft clauses, corresponding to the different "levels of satisfaction" of $\varphi_i$. their weights are $1$, $n$ and $n^2$, where $n$ is the number of soft specifications in $\square\varphi_1, \ldots, \square\varphi_n$. The weights reflect the ordering of implementations with respect to their satisfaction of $\square\varphi_1 \wedge \ldots \wedge \square\varphi_n$.

## III. Benchmark Instances

In [1] we applied our method for maximum realizability to two examples, resulting in two sets of MaxSAT benchmarks.

*1) Robotic Navigation:* We applied our method to the strategy synthesis for a robotic museum guide. The full set of hard and soft LTL specifications is given in [1]. The smallest bound on the implementation size for which the hard LTL specification is realizable is $8$. The MaxSAT instances are named

`robot_navigation_<bound>.wcnf`

where `<bound>`$\in \{8, 9, 10\}$ is the bound on the implementation size (number of states in the transition system).

*2) Power Distribution Network:* We considered also the problem of dynamic reconfiguration of power distribution networks. A power network consists of a set of power supplies (generators) and a set of loads (consumers). The network is a bipartite graph with edges between supplies and loads, where

each supply is connected to multiple loads and each load is connected to multiple supplies. Each power supply has an associated capacity, which determines how many loads it can power at a given time. It is possible that not all loads can be powered all the time. Some loads are critical and must be powered continuously, while others are not and should be powered when possible. Some loads can be initializing, meaning they must be powered only initially for several steps. Power supplies can become faulty during operation, which necessitates dynamic network reconfiguration.

The hard LTL specification asserts that the critical loads must always be powered, the initializing loads should be powered initially, a load is powered by at most one supply, the capacity of supplies is not exceeded, and when a supply is faulty it is not in use. The soft LTL specifications state that non-critical loads are always powered, and possibly also that a powered load should remain powered unless its supply fails. The full formal specifications are given in [1].

We considered 12 different instances determined by

- the network connectivity (full or sparse),
- the number of generators,
- the number of loads,
- the capacity of the generators,
- the numbers of critical and initial loads,
- the number/type of soft LTL specifications.

The resulting MaxSAT instances are named

`power-distribution_<id>_<bound>.wcnf`

where `<id>`$\in \{1, 2, \ldots, 12\}$ is the benchmark id, and `<bound>`$\in \{2, 3, \ldots, 8\}$ is the implementation size bound.

## REFERENCES


[1] Rayna Dimitrova, Mahsa Ghasemi, and Ufuk Topcu. Maximum realizability for linear temporal logic specifications. *CoRR*, abs/1804.00415, 2018.

[2] Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of bounded synthesis. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 354–370, 2017.

[3] Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *STTT*, 15(5-6):519–539, 2013.

[4] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

*MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, volume B-2018-2 of Department of Computer Science Series of Publications B, University of Helsinki 2018.

# MaxSAT Instances of the Team Composition Problem in a Classroom

Felip Manyà
*IIIA, CSIC*
Bellaterra, Spain

Santiago Negrete
*Universidad Autónoma Metropolitana (DCCD, Cuajimalpa)*
CDMX, Mexico

Joan Ramon Soler
*IIIA, CSIC*
Bellaterra, Spain

***Abstract*—Given a classroom containing a fixed number of students and a fixed number of tables that can be of different sizes, as well as a list of preferred classmates to sit with for each student, the team composition problem in a classroom (TCPC) is the problem of finding an assignment of students to tables in such a way that preferences are maximally-satisfied. In this paper, we formally define the the TCPC, describe how this problem can be encoded to MaxSAT and give the most relevant features of the instances submitted to the MaxSAT Evaluation.**

## I. The Problem

The version of the TCPC that we use in this paper has the following constraints:

- The classroom has $n$ students.
- The classroom has tables of 2 and 3 students with a combined capacity for $n$ students.
- Each student has provided a list of classmates she would prefer to sit with.

The objective is to find an assignment of students to tables such that preferences are maximally-satisfied. Notice that the first two constraints are hard whereas the last one is soft. We will say that a solution is *fully-satisfied* if, and only if, all the students in the same table have the rest of the students of the table in their list of preferences. We will say that a solution is *maximally-satisfied* if, and only if, the number of students who have their preferences satisfied is maximized. Note that a fully-satisfied solution is also a maximally-satisfied solution. It was proved in [2] that the TCPC is NP-hard.

## II. The Encoding

We present two different ways of encoding the TCPC in the weighted partial MaxSAT formalism [1]. In the first approach, the objective is to maximize the quality of the solution and we refer to it as the maximizing encoding. In the second approach, the objective is to minimize the quality loss and we refer to it as the minimizing encoding.

### A. The maximizing encoding

To illustrate how to model the problem, we will consider that the classroom has 28 students and there are 8 tables of 2 students and 4 tables of 3 students. This is a typical classroom distribution in many secondary schools.

This work was partially supported by the project LOGISTAR from the EU H2020 Research and Innovation Programme under Grant Agreement No. 769142 and the MINECO-FEDER project RASO TIN2015-71799-C2-1-P.

First of all, we define the set of Boolean variables of our encoding: $\{x_{ij}|1 \leq i < j \leq 28\} \cup \{x_{ijk}|1 \leq i < j < k \leq 28\} \cup \{y_i|1 \leq i \leq 28\}$. These variables have the following intended meaning: $x_{ij}$ is true iff students $i$ and $j$ sit together in a table of 2; $x_{ijk}$ is true iff students $i, j$ and $k$ sit together in a table of 3; and $y_i$ is true if student $i$ sits in a table of 2 and is false if student $i$ sits in a table of 3.

Using the previous Boolean variables, we create a Weighted Partial MaxSAT instance that encodes the constraints of the problem. The proposed encoding has the following hard clauses:

(i) For each student $i$, where $1 \leq i \leq 28$, the encoding contains a set of hard clauses that encode the following cardinality constraint:

1) If $i = 1$, then
$$\sum_{j=2}^{28} x_{1j} + \sum_{j=2}^{27} \sum_{k=j+1}^{28} x_{1jk} = 1$$
2) If $2 \leq i \leq 27$, then
$$\sum_{j=1}^{i-1} x_{ji} + \sum_{j=i+1}^{28} x_{ij} + \sum_{j=1}^{i-1} \sum_{k=i+1}^{28} x_{jik} + \sum_{j=i+1}^{27} \sum_{k=j+1}^{28} x_{ijk} = 1$$
3) If $i = 28$, then
$$\sum_{j=1}^{27} x_{j28} + \sum_{j=1}^{26} \sum_{k=j+1}^{27} x_{jk28} = 1$$

This cardinality constraint states that student $i$ sits exactly in one table, and the table is either of 2 or 3.

(ii) For each variable $x_{ij}$, the encoding contains the hard clauses $\neg x_{ij} \vee y_i$ and $\neg x_{ij} \vee y_j$. These clauses state that if $x_{ij}$ is true, then students $i$ and $j$ sit in a table of 2.

(iii) For each variable $x_{ijk}$, the encoding contains the hard clauses $\neg x_{ijk} \vee \neg y_i$, $\neg x_{ijk} \vee \neg y_j$ and $\neg x_{ijk} \vee \neg y_k$. These clauses states that if $x_{ijk}$ is true, then students $i$, $j$ and $k$ sit in a table of 3.

(iv) The encoding contains a a set of hard clauses that encode the following cardinality constraints: $\sum_{i=1}^{28} y_i = 16$ and $\sum_{i=1}^{28} \neg y_i = 12$. These cardinality constraints state that there are 16 students sitting in tables of 2 and 12 students sitting in tables of 3.

In practice, it is sufficient to add either the constraint $\sum_{i=1}^{28} y_i = 16$ or the constraint $\sum_{i=1}^{28} \neg y_i = 12$ because

if there are exactly 16 (12) variables $y_i$, $1 \leq i \leq 28$, that evaluate to true (false), then the remaining 12 (16) variables must evaluate to false.

The submitted instances encode the previous cardinality constraints using PBLib[1], which is a C++ tool for efficiently encoding pseudo-Boolean constraints to CNF.

The soft clauses of our encoding are the following weighted unit clauses:

1) For each variable $x_{ij}$, $1 \leq i < j \leq 28$, the encoding contains the weighted unit clause $(x_{ij}, w_{ij})$.
2) For each variable $x_{ijk}$, $1 \leq i < j < k \leq 28$, the encoding contains the weighted unit clause $(x_{ijk}, w_{ijk})$.

A key aspect of our encoding is how weights are assigned to the variables of the form $x_{ij}$ and $x_{ijk}$. First of all, we build a directed graph $G = (V, E)$, where $V$ contains a vertex $i$ for each student $i$ in the classroom, and $E$ contains an edge $(i, j)$ if student $i$ wants to sit with student $j$. The weight associated with each student $i$ in $G$, denoted by $w(i)$, is the out-degree of the vertex $i$ of $G$.[2] The weight associated with the variable $x_{ij}$, denoted by $w_{ij}$, is $2(w(i) \times w(j))$, where $w(i)$ and $w(j)$ are the weights associated with vertices $i$ and $j$, respectively, in the subgraph of $G$ induced by the set of vertices $\{i, j\}$ (i.e.; the weight of student $i$ and $j$ in $G(\{i, j\})$). The weight associated with the variable $x_{ijk}$, denoted by $w_{ijk}$, is $3(w(i) \times w(j) \times w(k)/8)$, where $w(i)$, $w(j)$ and $w(k)$ are the weights associated with vertices $i$, $j$ and $k$, respectively, in $G(\{i, j, k\})$. The value of $w(i) \times w(j)$ ranges from 0 to 1 and the value of $w(i) \times w(j) \times w(k)$ ranges from 0 to 8. This explains the fact that $w(i) \times w(j) \times w(k)$ is divided by 8. Moreover, we multiply the weights by 2 in the tables of 2 and by 3 in the tables of 3. In this way, we maximize the number of satisfied students. Note that if the weight assigned to $x_{ij}$ is 2, there are 2 satisfied students if they sit together in a table of 2, whereas if the weight assigned to $x_{ijk}$ is 3, there are 3 satisfied students if they sit together in a table of 3. The weight $w_{ij}$ ($w_{ijk}$) associated with a table of 2 (3) indicates the quality of the assignment of students $i$ and $j$ ($i$, $j$ and $k$) to a table of 2 (3): the bigger the weight, the better the assignment of students to tables.[3]

In the previous encoding, if the weight associated with a variable is 0, then the negation of this variable is added as a unit clause in the hard part. Moreover, an optimal solution corresponds to a fully-satisfied solution if, and only if, all the satisfied soft clauses of the form $(x_{ij}, w_{ij})$ and $(x_{ijk}, w_{ijk})$ have weight 2 and 3, respectively.

For fully-satisfied instances, if we add to the hard part the negation of $x_{ij}$ (i.e., the unit hard clause $\neg x_{ij}$) for each variable $x_{ij}$ whose associated weight is different from 2 and the negation of $x_{ijk}$ (i.e., the unit hard clause $\neg x_{ijk}$) for each variable $x_{ijk}$ whose associated weight is different from 3, then we do not need to add any soft clause. Moreover, any satisfying assignment of the hard part allows us to derive a fully-satisfied solution. This case can be solved either with a SAT solver or with a MaxSAT solver fed with a MaxSAT instance that only contains hard clauses. Actually, to find a fully-satisfied solution is a decision problem.

If there is no fully-satisfied solution, the problem becomes an optimization problem and the objective is to find a solution that satisfies students as much as possible. Because of that, in the general case, we add the clauses $(x_{ij}, w_{ij})$ and $(x_{ijk}, w_{ijk})$ such that $w_{ij} \neq 0$ and $w_{ijk} \neq 0$ in the soft part of the encoding. In this way, we provide a solution that maximizes the number of satisfied students. In this case, we say that we have a maximally-satisfied solution.

An optimal solution to the TCPC is obtained from a MaxSAT optimal interpretation by assigning students $i$ and $j$ to the same table of 2 if, and only if, the literal $x_{ij}$ is satisfied by the optimal interpretation; and by assigning students $i$, $j$ and $k$ to the same table of 3 if, and only if, the literal $x_{ijk}$ is satisfied by the optimal interpretation.

If an optimal interpretation satisfies the soft clause $(x_{ij}, w_{ij})$, then this interpretation falsifies all the soft clauses $(x_{lm}, w_{lm})$ and $(x_{lmn}, w_{lmn})$ such that $l$, $m$ or $n$ are equal to $i$ or $j$ because of the cardinality constraint that states that every student sits exactly in one table. A similar situation happens when the satisfied clause is of the form $(x_{ijk}, w_{ijk})$, corresponding to a table of 3. Thus, the number of falsified soft clauses is usually greater than the number of satisfied soft clauses, and the maximum sum of weights of satisfied clauses indicates the maximum quality that can be reached taking into account the preferences of the students.

### *B. The minimizing encoding*

The minimizing encoding focus on minimizing the quality loss instead of maximizing the quality of the solution as in the maximizing encoding. So, the challenge now is to adequately represent the notion of quality loss in the TCPC and derive a more efficient encoding.

The minimizing encoding is defined over the same set of Boolean variables and has the hard constraints of the maximizing encoding. The soft clauses are derived from the soft clauses of the maximizing encoding as follows:

1) each soft clause $(x_{ij}, w_{ij})$ is replaced with the soft clause $(\neg x_{ij}, w_{max} - w_{ij})$, and
2) each soft clause $(x_{ijk}, w_{ijk})$ is replaced with the soft clause $(\neg x_{ijk}, w'_{max} - w_{ijk})$,

where $w_{max}$ is the maximum weight that can be assigned to a table of 2 and $w'_{max}$ is the maximum weight that can be assigned to a table of 3. In our encoding, $w_{max} = 2$ and $w'_{max} = 3$.

An optimal solution to the TCPC is obtained from a MaxSAT optimal interpretation by assigning students $i$ and $j$ to the same table of 2 if, and only if, the literal $\neg x_{ij}$ is falsified by the optimal interpretation; and by assigning students $i$, $j$ and $k$ to the same table of 3 if, and only if, the

[1]http://tools.computational-logic.org/content/pblib.php

[2]The out-degree of a vertex is the number of edges going out of a vertex in a directed graph.

[3]Since most of the MaxSAT solvers deal with weights that are positive integers, in the experiments we multiply the weights by 100 and take the integer part.

literal $\neg x_{ijk}$ is falsified by the optimal interpretation. Note that $\neg x_{ij}$ and $\neg x_{ijk}$ are falsified if, and only if, $x_{ij}$ and $x_{ijk}$ are satisfied. If an optimal interpretation falsifies the soft clause $(\neg x_{ij}, w'_{ij})$, then it satisfies all the soft clauses $(\neg x_{lm}, w'_{lm})$ and $(\neg x_{lmn}, w'_{lmn})$ such that $l$, $m$ or $n$ are equal to $i$ or $j$ because of the cardinality constraint that states that every student sits exactly in one table. A similar situation happens when the falsified clause is of the form $(\neg x_{ijk}, w'_{ijk})$.

In contrast to the maximizing encoding, the number of satisfied soft clauses in an optimal solution of the minimizing encoding is usually greater than the number of falsified soft clauses. This implies that the number of conflicts that a MaxSAT solver has to identify for finding an optimal solution is greater in the maximizing encoding than in the minimizing encoding and may have some impact on the performance of the solver.

The weight of the soft clause $(\neg x_{ij}, w_{max} - w_{ij})$ $((\neg x_{ijk}, w'_{max} - w_{ijk}))$ indicates the quality loss if students $i$ and $j$ ($i$, $j$ and $k$) sit together in a table of 2 (3): the smaller the weight, the better the assignment of students to tables. In fact, the weight $w_{max} - w_{ij}$ ($w'_{max} - w_{ijk}$) is the penalty to be paid by students $i$ and $j$ ($i$, $j$ and $k$) if they sit in the same table. So, the minimum sum of weights of falsified clauses indicates the minimum quality loss that can be reached taking into account the preferences of the students.

If the minimum sum of weights of falsified clauses in an optimal solution is 0, then this solution is fully-satisfied. Note that the clauses of the form $(\neg x_{ij}, 0)$ correspond to tables of 2 in which students $i$ and $j$ prefer to sit together, and the clauses of the form $(\neg x_{ijk}, 0)$ correspond to tables of 3 in which students $i$, $j$ and $k$ prefer to sit together. In practice, the clauses $(\neg x_{ij}, 0)$ and $(\neg x_{ijk}, 0)$ can be removed from the soft part and the encoding remains correct.

It is worth mentioning that the minimization approach proposed here can be extended to other combinatorial optimization problems. It is particularly useful when the resulting MaxSAT encoding has subsets of soft unit clauses whose literals are involved in cardinality constraints in the hard part, because it can reduce considerably the number of conflicts needed to find an optimal solution. The main difficulty of the minimizing encoding is to define a suitable weighting function that preserves the optimal solutions between the maximizing and the minimizing encodings.

### C. The Instances

We submitted 60 TCPC instances encoded to weighted partial MaxSAT using the minimizing encoding. There 4 sets of 15 instances. The first set corresponds to instances with 91 students, the second set corresponds to instances with 98 students, the third set corresponds to instances with 105 students and the fourth set corresponds to instances with 112 students. In all the instances the number of tables of 2 is the double of the number of tables of 3. For example, the instances with 91 students contain 26 tables of 2 and 13 tables of 3.

## REFERENCES


[1] Li CM, Manyà F. MaxSAT, Hard and Soft Constraints. In: Biere A, van Maaren H, Walsh T (eds.), Handbook of Satisfiability, pp. 613–631. IOS Press, 2009.

[2] Manyà F, Negrete S, Roig C, Soler JR. A MaxSAT-Based Approach to the Team Composition Problem in a Classroom. In: Autonomous Agents and Multiagent Systems - AAMAS 2017 Workshops, Visionary Papers, São Paulo, Brazil, Revised Selected Papers. Springer LNCS 10643, 2017 pp. 164–173.

*MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, volume B-2018-2 of Department of Computer Science Series of Publications B, University of Helsinki 2018.

# Approximately Propagation Complete and Approximately Conflict Propagating SAT Encoding Computation MaxSAT Benchmarks

Rüdiger Ehlers
University of Bremen & DFKI GmbH
Germany

## I. Description

This benchmark set contains MaxSAT instances that encode the problem of finding approximately propagation complete and approximately conflict propagating conjunctive normal form (CNF) encodings for a couple of interesting constraints. The approach for reducing this problem to MaxSAT is explained in a paper [1] to be published at the $21^{st}$ International Conference on Theory and Applications of Satisfiability Testing (SAT 2018).

The benchmark set is based the experimental evaluation of the paper mentioned above, where constraint encodings for the propagation quality tuples $(\infty, \infty)$, $(1, \infty)$, $(2, \infty)$, $(3, \infty)$, $(3, 3)$, and $(\infty, 1)$ are computed. The meaning of these tuples is also mentioned in the said paper.

To keep the benchmark set small, a couple of benchmarks for which both the solvers `LMHS` [2] (in the version from March 2018) and `maxino-2015-k16` [3] both need less than 3 seconds of solving time on a moderately modern computer (using an Intel(R) Core(TM) i5-4200U CPU with a 1.60 GHz clock rate) have been removed. Furthermore, MaxSAT instances whose computation takes more than 30 minutes on the said computer are also left out.

The benchmark file names contain:

- the name of the constraint that is to be encoded into conjunctive normal form, and
- the propagation quality tuple, where the elements are concatenated and $\infty$ elements are replaced by "`99`".

Some of the constraints to be encoded into CNF are taken from the set of examples supplied with the GenPCE tool by Brain et al. [4]. That tool computes propagation complete CNF encodings of constraints.

The program to compute the MaxSAT benchmarks is available at https://github.com/progirep/optic in the branch "`MaxSATEvaluationBenchmarkGeneration`".

## References

[1] R. Ehlers and F. Palau Romero, "Approximately propagation complete and conflict propagating constraint encodings," in *$21^{st}$ International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2018, accepted paper.

[2] P. Saikko, J. Berg, and M. Järvisalo, "LMHS: A SAT-IP hybrid MaxSAT solver," in *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016*, 2016, pp. 539–546.

[3] M. Alviano, C. Dodaro, and F. Ricca, "A MaxSAT algorithm using cardinality constraints of bounded size," in *Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 2015, pp. 2677–2683.

[4] M. Brain, L. Hadarean, D. Kroening, and R. Martins, "Automatic generation of propagation complete SAT encodings," in *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016*, 2016, pp. 536–556.

# MSE 2018 Benchmarks: Visibly Pushdown Automata Minimization

Matthias Heizmann
*Computer science institute*
*University of Freiburg*
Freiburg, Germany
heizmann@informatik.uni-freiburg.de

Christian Schilling
*Computer science institute*
*University of Freiburg*
Freiburg, Germany
schillic@informatik.uni-freiburg.de

***Abstract*—We describe a benchmark set consisting of 67 partial Max-SAT problem instances. The benchmarks come from an automata minimization approach. Each benchmark formula encodes constraints on an equivalence relation over automaton states. Any assignment satisfying the constraints corresponds to a relation that allows us to reduce the size of the automaton without changing its language. The more soft clauses are set to true, the coarser the relation and the smaller the resulting automaton.**

## I. Introduction

The benchmarks presented in this article stem from a automata minimization approach. Automata minimization is a crucial component of the software verifier Ultimate Automizer [1]. Our benchmarks were produced while Ultimate Automizer was analyzing computer programs. In this section we briefly explain why automata minimization is important in this approach.

The verification approach [2] of Ultimate Automizer maintains an automaton that represents an abstraction of the program. Typically this automaton is growing in each iteration of a refinement loop. In practice this growth may lead to out-of-memory problems. Ultimate Automizer addresses this problem by using automata minimization. For finite automata there exist many known minimization techniques. However, for its interprocedural analysis [3] Ultimate Automizer uses *visibly pushdown automata* [4], to which these techniques cannot be applied. We have proposed an approach [5] that reduces the minimization problem for visibly pushdown automata to a PMax-SAT problem.

In the following we first explain our approach for finite automata and later extend it to visibly pushdown automata.

## II. Minimization via Quotienting

The most common approach to automata minimization is to identify and merge "equivalent" states (and corresponding transitions), which is also called *quotienting*. The resulting quotient automaton is uniquely determined by defining an equivalence relation over the automaton's states. We give a simple example for a finite automaton in Figure 1.

## III. Algorithm & Encoding

In [5] (see [6] for an extended version) we described sufficient constraints for equivalence relations in order to be suitable for quotienting, meaning that the quotient automaton preserves the language. These constraints can be encoded as a Boolean formula such that any satisfying assignment corresponds to a suitable relation in our sense.

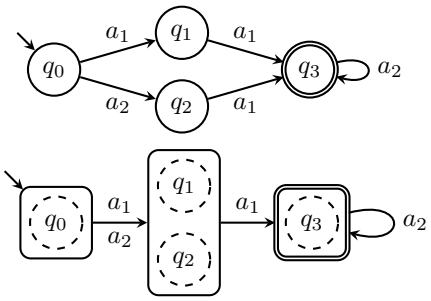



Fig. 1. A finite automaton (top) and a quotient automaton (bottom) with respect to the relation $\{(q_0,q_0),(q_1,q_1),(q_2,q_2),(q_1,q_2),(q_2,q_1),(q_3,q_3)\}$.

Next we outline the encoding for finite automata. For each unordered pair of states $p, q$ we introduce a variable $\mathsf{X}_{\{p,q\}}$ that encodes whether the pair can be merged.

For every two states $p, q$ where exactly one state is accepting we introduce a unit clause.

$$\neg\mathsf{X}_{\{p,q\}} \tag{1}$$

For all transitions $(p, a, p')$ and all states $q$ we add the following clause which states that, if the pair $p, q$ is equivalent, then at least one of $q$'s $a$-successors $q^a_1, \ldots, q^a_{n_a}$ is equivalent to $p'$.

$$\neg\mathsf{X}_{\{p,q\}} \vee (\mathsf{X}_{\{p',q^a_1\}} \vee \cdots \vee \mathsf{X}_{\{p',q^a_{n_a}\}}) \tag{2}$$

We also need to express the equivalence relation constraints. We handle reflexivity internally (by simply ignoring variables $\mathsf{X}_{\{q,q\}}$), and our variables already ensure symmetry. It remains to add transitivity clauses for any distinct states $q_1$, $q_2$, $q_3$.

$$\neg\mathsf{X}_{\{q_1,q_2\}} \vee \neg\mathsf{X}_{\{q_2,q_3\}} \vee \mathsf{X}_{\{q_1,q_3\}} \tag{3}$$

We are interested in satisfying assignments that result in quotient automata with few states. Since a variable encodes whether a pair of states will be merged, the more variables are assigned `true`, the more states can be merged. For instance,

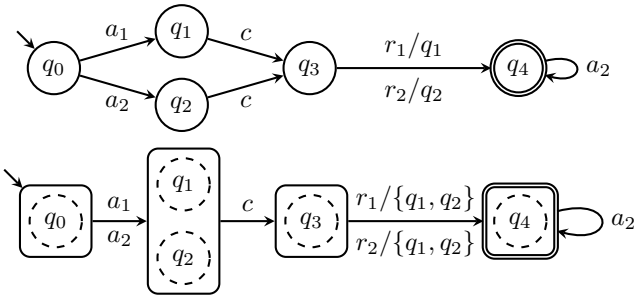



Fig. 2. A visibly pushdown automaton (top) and a quotient automaton (bottom) with respect to the relation $\{(q_0, q_0), (q_1, q_1), (q_2, q_2), (q_1, q_2), (q_2, q_1), (q_3, q_3), (q_4, q_4)\}$.

the constraints are always trivially satisfied by assigning `false` to all variables, which is hence completely useless for our goal. We thus want to maximize the number of variables that are assigned the value `true`. This optimization can be added by introducing for each variable a *soft* clause

$$\mathsf{X}_{\{q_i, q_j\}} \tag{4}$$

of weight 1 and making all previous clauses *hard*, thereby obtaining a partial MaxSAT problem.

## IV. Extension to Visibly pushdown automata

A visibly pushdown automaton extends the finite automaton model by adding a stack. Unlike an ordinary pushdown automaton, the stack access is determined by the type of the input symbol. A symbol can have one of three types: *internal*, in which case the stack is neither read from nor written to; *call*, in which case the stack is written to ("push"), where in our case the written symbol is always the current state; and *return*, in which case the stack is read from ("pop"). We give a simple example in Figure 2.

To update the encoding, we need to reflect the new transition types. The internal transitions correspond to the finite automaton transitions above. For call transitions $(p, c, p')$ and return transitions $(p, r, \hat{p}, p')$ and all states $q$ and $\hat{q}$ we respectively construct one of the following clauses.

$$\neg\mathsf{X}_{\{p,q\}} \vee (\mathsf{X}_{\{p',q_1^c\}} \vee \cdots \vee \mathsf{X}_{\{p',q_{n_c}^c\}}) \tag{5}$$

$$\neg\mathsf{X}_{\{p,q\}} \vee \neg\mathsf{X}_{\{\hat{p},\hat{q}\}} \vee (\mathsf{X}_{\{p',q_1^r\}} \vee \cdots \vee \mathsf{X}_{\{p',q_{n_r}^r\}}) \tag{6}$$

Again, the states $q_i^c$ are the respective $c$-successors of $q$, and the states $q_i^r$ are the $r$-successors of $q$ with stack symbol $\hat{q}$.

## V. Discussion

We note that for deterministic finite automata, quotienting is sufficient to obtain the unique minimal automaton. This property neither holds for nondeterministic finite automata nor for visibly pushdown automata, and moreover, the minimal automaton is not unique. For finite automata, the above algorithm at least always results in a unique maximal relation, which is also known as direct bisimulation and can be computed more efficiently [7]. For visibly pushdown automata, the maximal relation is not unique anymore. That makes the problem a perfect candidate for a reduction to PMax-SAT, where also no unique solution exists in general.

## VI. Benchmark file description

We have implemented the above algorithm in the automata library of the Ultimate framework[1]. The automata library is also available via a web interface[2].

To construct the benchmarks, we applied Ultimate Automizer to C programs from the SV-COMP 2016 [8] to produce 67 automata, which our implementation then encoded in a `.wcnf` file. The file name convention is `UAutomizer_`**`P`**`_Abstraction`**`K`**`.wcnf`, where **`P`** is the name of the program that was analyzed and **`K`** is the iteration in which the automaton was produced.

The benchmark size grows cubically with the number of states. The input automaton sizes range between 11 and 104 states, which are considered small in the practical application (the automata easily grow to tens of thousands of states).

The above-mentioned constraints are added in the following order to each file.

1) acceptance (unit clauses of type (1))
2) transitions (clauses of types (2), (5), and (6))
3) transitivity (ternary clauses of type (3))
4) soft clauses (unit clauses of type (4))

The arity of the transition clauses depends on the nondeterminism in the automaton. Typically the transitivity clauses dominate a benchmark file because their number is cubic in the number of variables.

## References


[1] M. Heizmann, Y. Chen, D. Dietsch, M. Greitschus, J. Hoenicke, Y. Li, A. Nutz, B. Musa, C. Schilling, T. Schindler, and A. Podelski, "Ultimate Automizer and the search for perfect interpolants - (competition contribution)," in *TACAS II*, ser. LNCS, vol. 10806. Springer, 2018, pp. 447–451. [Online]. Available: https://doi.org/10.1007/978-3-319-89963-3_30

[2] M. Heizmann, J. Hoenicke, and A. Podelski, "Software model checking for people who love automata," in *CAV*, ser. LNCS, vol. 8044. Springer, 2013, pp. 36–52. [Online]. Available: https://doi.org/10.1007/978-3-642-39799-8_2

[3] ——, "Nested interpolants," in *POPL*. ACM, 2010, pp. 471–482. [Online]. Available: http://doi.acm.org/10.1145/1706299.1706353

[4] R. Alur and P. Madhusudan, "Visibly pushdown languages," in *STOC*. ACM, 2004, pp. 202–211. [Online]. Available: http://doi.acm.org/10.1145/1007352.1007390

[5] M. Heizmann, C. Schilling, and D. Tischner, "Minimization of visibly pushdown automata using partial Max-SAT," in *TACAS (1)*, ser. LNCS, vol. 10205. Springer, 2017, pp. 461–478. [Online]. Available: https://doi.org/10.1007/978-3-662-54577-5_27

[6] ——, "Minimization of visibly pushdown automata using partial Max-SAT," *CoRR*, vol. abs/1701.05160, 2017. [Online]. Available: http://arxiv.org/abs/1701.05160

[7] K. Etessami, T. Wilke, and R. A. Schuller, "Fair simulation relations, parity games, and state space reduction for Büchi automata," *SIAM J. Comput.*, vol. 34, no. 5, pp. 1159–1175, 2005. [Online]. Available: https://doi.org/10.1137/S0097539703420675

[8] D. Beyer, "Reliable and reproducible competition results with benchexec and witnesses (report on SV-COMP 2016)," in *TACAS*, ser. LNCS, vol. 9636. Springer, 2016, pp. 887–904. [Online]. Available: https://doi.org/10.1007/978-3-662-49674-9_55


[1] https://github.com/ultimate-pa/ultimate

[2] https://ultimate.informatik.uni-freiburg.de/automata_library

*MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*, volume B-2018-2 of Department of Computer Science Series of Publications B, University of Helsinki 2018.

# RBAC User Query Authorization Problem: 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

***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

$$DMER(\{R_1, ..., R_n\}, k)$$

stating that $n$ or more roles from the set of (conflicting) roles $\{R_1, ..., R_n\}$ 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 2018

The benchmark set `uaq` consists of 97 instances obtained by encoding different families of synthetically generated UAQ problems:

- `nr` (number of roles): these problems are parametric in the number of roles ($|R|$); the number of permissions grows linearly with the number of roles; permissions are

assigned randomly to roles under the proviso that each permission is assigned to at least four roles and, dually, that each role has assigned at least two permissions; $|P_{lb}|$ is set to 50.

- `plb`: these problems are parametric in $|P_{lb}|$. Permissions are assigned randomly to roles under the proviso that each permission is assigned to three roles and five permissions are assigned to each role.
- `rpp` (roles per permission): these problems are parametric in the number of roles to which each permission is assigned. Permissions are assigned randomly to roles under the proviso that two permissions are assigned to each role; the number of roles $|R|$ is set to 100 and $|P_{lb}|$ is set to 20.
- `ppr` (permissions per role): these problems are parametric in the number of permissions assigned to each role (*ppr*), while the number of roles to which each permission is assigned is set to either 4 or 6. The number of roles, i.e. $|R|$, is set to 200 while $|P_{lb}|$ is set to 100.

Each problem instance has one DMER constraint every 3 roles, i.e. the total number of DMER constraints is equal to $|R|/3$. Thus, for example, instances with 60 roles have 20 constraints. All DMER constraints have $k = 2$; the value of $n$ is 3 for `nr` and `rpp`, 5 for `plb` and `ppr`.

The instances are named using following convention: uaq-*family*-nr*NR*-*nc*NC-*nN*-*kK*-*rppRPP*-ppr*PPR*-plb$P_{lb}$.dimacs, where:

- *family* is the family of the problem instance (i.e. `nr`, `plb`, `rpp` or `ppr`),
- *NR* is the number of roles,
- *NC* is the number of DMER constraints,
- *n* and *k* are the parameters of the DMER constraints,
- *rpp* is the number of roles assigned to each permission,
- *ppr* is the number of permissions assigned to each role and
- $P_{lb}$ is the number of permissions whose activation is requested.

## 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.

# XAI-MinDSet2: Explainable AI with MaxSAT

Alexey Ignatiev and Joao Marques-Silva
Faculty of Sciences, University of Lisbon, Portugal
{aignatiev,jpms}@ciencias.ulisboa.pt

## I. Introduction

Machine learning (ML) has witnessed remarkable progress and important successes in recent years. In some settings, predictions made by machine learning algorithms should provide explanations, preferably explanations that can be interpreted (or understood) by human decision makers. Concrete examples include safety-critical situations, but also when transparency of decisions is paramount. An often used approach to provide explanations for ML predictions is to resort to some sort of logic-related model, including rule/decision lists [1], rule/decision sets [2], and decision trees [3]. These logic-related models can in most cases associate explanations with predictions, represented as conjunctions of literals, that follow from the actual model representation. Clearly, the smaller the model representation is, the simpler the explanations are likely to be, and so easier to understand by human decision makers.

A novel SAT-based approach to computing smallest size interpretable decision sets has been recently proposed [4]. As shown in [4], the proposed iterative procedure exploiting a SAT oracle to compute the smallest number of rules in a decision set significantly pushes the state of the art in decision set learning [2] and, thus, in explainable machine learning. The XAI-MinDSet2 benchmark set briefly[1] described below comprises (unweighted) partial MaxSAT instances encoding the problem of literal minimization given a target (smallest size) set of rules.

## II. Description

The classification problem can be roughly formulated in the following way. Given a training dataset (e.g. set of example *itemsets* each being marked by a *label*, or *class*), one needs to find a set of rules covering all the itemsets in the training dataset and generalizing well on the *unseen* data, i.e. itemsets not presented in the training dataset.

Depending of the policy for handling the rule overlap in the target decision sets, several problem formulations were presented and studied in [4]. One of the formulations, which is referred to as *MinDSet2* (or *MinDS2*), consists in computing a minimum size set of rules that has no overlap on the training data and covers all the example itemsets, i.e. it classifies all of them correctly.

The MinDS2 problem encoding into SAT is omitted here but can be found in [4]. It should be noted that the approach of [4] is an iterative procedure that (1) encodes the MinDS2 problem into SAT, given a training dataset and an integer parameter $k$, and (2) invokes a SAT oracle to decide whether there is a decision set with $k$ rules that is a solution for the MinDS2 problem. The process starts with $k$ being equal to the number of distinct classes and increases $k$ at every iteration. As soon as the SAT oracle decides that the input CNF formula $\mathcal{F}_k$ is satisfiable, the smallest size decision set (i.e. a decision set containing $k$ rules) can be extracted from the assignment satisfying $\mathcal{F}_k$.

Although value $k$ computed this way is typically very small, the rules of the resulting decision sets may be large; in other words, the computed decision sets can be too expensive in terms of the total number of literals used. This is where MaxSAT can be of help. As suggested by [4], when smallest $k$ is computed, an additional optimization problem can be devised to compute a decision set of size $k$ minimizing the total number of literals used. This approach follows the idea of Boolean lexicographic optimization [5]. One can easily formulate such MaxSAT problem by (1) considering the CNF formula encoding the MinDS2 problem of size $k$ as the hard part and (2) adding unit size soft clauses, which prefer to deselect every literal in the target decision set. The number of clauses falsified by a MaxSAT model is equal to the minimum number of literals that have to be used in the smallest size decision set for a given dataset. Such optimization problem was proposed and studied in [4] for a subset of the PMLB benchmark repository [6], [7]. As in [4], the total number of used datasets is 45 and so is the number of MaxSAT benchmarks. Each benchmark of the XAI-MinDSet2 family is named after the corresponding training dataset.

[1]The reader is referred to [4] for details.

## References

[1] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin, "Learning certifiably optimal rule lists," in *KDD*, 2017, pp. 35–44.
[2] H. Lakkaraju, S. H. Bach, and J. Leskovec, "Interpretable decision sets: A joint framework for description and prediction," in *KDD*, 2016, pp. 1675–1684.
[3] C. Bessiere, E. Hebrard, and B. O'Sullivan, "Minimising decision tree size as combinatorial optimisation," in *CP*, 2009, pp. 173–187.
[4] A. Ignatiev, F. Pereira, N. Narodytska, and J. Marques-Silva, "A SAT-based approach to learn explainable decision sets," in *IJCAR*, 2018, to appear.
[5] 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.
[6] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, "PMLB: a large benchmark suite for machine learning evaluation and comparison," *BioData Mining*, vol. 10, no. 1, p. 36, Dec 2017. [Online]. Available: https://doi.org/10.1186/s13040-017-0154-4
[7] "Penn machine learning benchmarks." [Online]. Available: https://github.com/EpistasisLab/penn-ml-benchmarks/

## Solver Index

# Benchmark Index

# Author Index