MaxSAT Evaluation 2023 : Solver and Benchmark Descriptions

Berg, Jeremias; Järvisalo, Matti; Martins, Ruben; Niskanen, Andreas

2023

http://hdl.handle.net/10138/564026

Berg, J, Järvisalo, M, Martins, R & Niskanen, A (eds) 2023, MaxSAT Evaluation 2023 : Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2023-2, Department of Computer Science, University of Helsinki, Helsinki.

MaxSAT Evaluation 2023

Solver and Benchmark Descriptions

Jeremias Berg, Matti Järvisalo, Ruben Martins, and Andreas Niskanen *(editors)*

University of Helsinki
Department of Computer Science
Series of Publications B
Report B-2023-2

ISSN 1458-4786
Helsinki 2023

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 2023 brought on the 17th edition of the MaxSAT Evaluations, organized as a satellite event of the 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023). 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.

The 2023 evaluation consisted of a total of five tracks: two for exact (earlier referred to as complete) solvers (one for solvers focusing on unweighted and one for solvers focusing on weighted MaxSAT instances), two for in-exact (earlier referred to as incomplete) MaxSAT solvers (using two short per-instance time limits, 60 and 300 seconds, differentiating from the per-instance time limit of 1 hour imposed in the main exact tracks), as well as the incremental track that was introduced for the first time in 2022. (Unfortunately, due to lack of participants the incrementral track was not executed this year.) As in 2017-2022, no distinction was made between "industrial" and "crafted" benchmarks, and no track for purely randomly generated MaxSAT instances was organized.

Adhering to the new rules introduced in 2017, solvers were now required to be open-source. Furthermore, a 1-2 page solver description was expected to accompany 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 2021 are collected together in this compilation.

Finally, we would like to thank everyone who contributed to MaxSAT Evaluation 2023 by submitting their solvers or new benchmarks. We are also grateful for the computational resources provided by the StarExec initiative and the Finnish Computing Competence Infrastructure (FCCI) which enabled running the 2023 evaluation smoothly.

Jeremias Berg, Matti Järvisalo, Ruben Martins, & Andreas Niskanen
MaxSAT Evaluation 2023 Organizers

Contents

Benchmark Descriptions

SOLVERS

CASHWMaxSAT-CorePlus: Solver Description

Shiwei Pan[1,2], Yiyuan Wang[1,2], Zhendong Lei[3,4], Shaowei Cai[3,4,*], Shimao Wang[1,2], Minghao Yin[1,2,*]
[1]School of Computer Science and Information Technology, Northeast Normal University, China
[2]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China
[3]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[4]School of Computer Science and Technology, University of Chinese Academy of Sciences, China
[*]corrsponding author
yiyuanwangjlu@126.com, leizhendong3@huawei.com, caisw@ios.ac.cn,
{pansw779, wangsm928, ymh}@nenu.edu.cn

***Abstract*—This document describes the MaxSAT solver CASHWMaxSAT-CorePlus, submitted to the complete tracks (include unweighted and weighted track) of MaxSAT Evaluation 2023.**

I. Introduction

We developed an improved complete MaxSAT solver called CASHWMaxSAT-CorePlus based on UWrMaxSat [1], SCIP 8.0 [2] and CASHWMaxSAT [3]. In addition, CASHWMaxSAT-CorePlus used an unsatisfiable-core-based OLL procedure [4]–[7]. In this work, we propose two novel ideas to improve the last year's version of CASHWMaxSAT-CorePlus [8].

- For weighted instances, previous works usually divided the soft clauses into different layers based on clauses' weight value. The soft clauses with the same weight value are in the same layer. Different from previous method, we adopt a new layered approach. Assume that the maximum weight value of soft clauses is w_{max}. we will put all the soft clauses whose weight value is in $[w_{max}/2n, w_{max}/n]$ into the same layer where an integer $n \geq 1$.
- For unweighted instances, we call the SAT solver to solve the given instances that only includes all hard clauses. If the SAT solver returns the "sat" state, we will obtain an assignment. We put all true soft clauses based on the assignment into the SAT solver and call the SAT solver. After many iterations, the SAT solver will return the "sat" state. After then, we will put the remaining soft clauses into the SAT solvers and solve them.

II. Future work

First, we could use a simplified version of MaxSAT local search solvers to improve the satisfied solution. Second, we could try to design a novel selection way for selecting an unsatisfiable-core on weighted cases.

References

[1] M. Piotrów, "Uwrmaxsat in maxsat evaluation 2021," *MaxSAT Evaluation 2021*, p. 17, 2021.
[2] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner *et al.*, "The scip optimization suite 8.0," *arXiv preprint arXiv:2112.08872*, 2021.
[3] Z. Lei, S. Cai, D. Wang, Y. Peng, F. Geng, D. Wan, Y. Deng, and P. Lu, "Cashwmaxsat: Solver description," *MaxSAT Evaluation 2021*, p. 8, 2021.
[4] B. Andres, B. Kaufmann, O. Matheis, and T. Schaub, "Unsatisfiability-based optimization in clasp," in *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.
[5] A. Morgado, F. Heras, M. Liffiton, J. Planes, and J. Marques-Silva, "Iterative and core-guided maxsat solving: A survey and assessment," *Constraints*, vol. 18, no. 4, pp. 478–534, 2013.
[6] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided maxsat with soft cardinality constraints," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2014, pp. 564–573.
[7] A. Ignatiev, A. Morgado, and J. Marques-Silva, "Rc2: an efficient maxsat solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, no. 1, pp. 53–64, 2019.
[8] Z. Lei, Y. Wang, S. Pan, S. Cai, and M. Yin, "Cashwmaxsat-coreplus: Solver description," *MaxSAT Evaluation 2022*, p. 1, 2022.

CASHWMaxSAT-CorePlus-m: Solver Description

Yiyuan Wang[1,2], Shiwei Pan[1,2], Zhendong Lei[3,4], Shaowei Cai[3,4,*], Xinyu Wang[1,2], Minghao Yin[1,2,*]
[1]School of Computer Science and Information Technology, Northeast Normal University, China
[2]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China
[3]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[4]School of Computer Science and Technology, University of Chinese Academy of Sciences, China
*corrsponding author
yiyuanwangjlu@126.com, leizhendong3@huawei.com, caisw@ios.ac.cn,
{pansw779, wangxy435, ymh}@nenu.edu.cn

***Abstract*—This document describes the MaxSAT solver CASHWMaxSAT-CorePlus-m, submitted to the complete tracks (include unweighted and weighted track) of MaxSAT Evaluation 2023.**

I. Introduction

We developed an improved complete MaxSAT solver called CASHWMaxSAT-CorePlus based on UWrMaxSat [1], SCIP 8.0 [2] and CASHWMaxSAT [3]. In addition, CASHWMaxSAT-CorePlus used an unsatisfiable-core-based OLL procedure [4]–[7]. In this work, we propose two novel ideas to improve the last year's version of CASHWMaxSAT-CorePlus [8]. Based on CASHWMaxSAT-CorePlus, we optimize the parameter settings of SCIP and develop a new algorithm called CASHWMaxSAT-CorePlus-m.

- For weighted instances, previous works usually divided the soft clauses into different layers based on clauses' weight value. The soft clauses with the same weight value are in the same layer. Different from previous method, we adopt a new layered approach. Assume that the maximum weight value of soft clauses is w_{max}. we will put all the soft clauses whose weight value is in $[w_{max}/2n, w_{max}/n]$ into the same layer where an integer $n \geq 1$.
- For unweighted instances, we call the SAT solver to solve the given instances that only includes all hard clauses. If the SAT solver returns the "sat" state, we will obtain an assignment. We put all true soft clauses based on the assignment into the SAT solver and call the SAT solver. After many iterations, the SAT solver will return the "sat" state. After then, we will put the remaining soft clauses into the SAT solvers and solve them.

II. Future work

First, we could use a simplified version of MaxSAT local search solvers to improve the satisfied solution. Second, we could try to design a novel selection way for selecting an unsatisfiable-core on weighted cases.

References

[1] M. Piotrów, "Uwrmaxsat in maxsat evaluation 2021," *MaxSAT Evaluation 2021*, p. 17, 2021.
[2] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner *et al.*, "The scip optimization suite 8.0," *arXiv preprint arXiv:2112.08872*, 2021.
[3] Z. Lei, S. Cai, D. Wang, Y. Peng, F. Geng, D. Wan, Y. Deng, and P. Lu, "Cashwmaxsat: Solver description," *MaxSAT Evaluation 2021*, p. 8, 2021.
[4] B. Andres, B. Kaufmann, O. Matheis, and T. Schaub, "Unsatisfiability-based optimization in clasp," in *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.
[5] A. Morgado, F. Heras, M. Liffiton, J. Planes, and J. Marques-Silva, "Iterative and core-guided maxsat solving: A survey and assessment," *Constraints*, vol. 18, no. 4, pp. 478–534, 2013.
[6] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided maxsat with soft cardinality constraints," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2014, pp. 564–573.
[7] A. Ignatiev, A. Morgado, and J. Marques-Silva, "Rc2: an efficient maxsat solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, no. 1, pp. 53–64, 2019.
[8] Z. Lei, Y. Wang, S. Pan, S. Cai, and M. Yin, "Cashwmaxsat-coreplus: Solver description," *MaxSAT Evaluation 2022*, p. 1, 2022.

CGSS2 in the 2023 MaxSAT Evaluation

Hannes Ihalainen, Jeremias Berg, Matti Järvisalo
HIIT, Department of Computer Science, University of Helsinki, Finland

I. Introduction

CGSS2 is a MaxSAT solver implementing the core-guided OLL algorithm [11], [1], extended with weight aware core extraction, structure sharing and selective addition of equivalences as described in [10] and [4]. The solver reimplements in C++ the techniques of an older CGSS solver [10]. The solver makes use of stratification, hardening and the so-called core exhaustion, core minimization and intrinsic atmost1 techniques described in [8]. CGSS2 also supports employing internally the mixed integer programming solver SCIP [6] to solve MaxSAT instances with time limit prior invoking OLL.

If you use CGSS2 in your research, we kindly ask you to cite [9] and [10].

II. Preliminaries

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

Without loss of generality we assume that each soft clause is unit, containing the negation of a variable. We say that a variable b is an *objective variable* (of the instance $\mathcal{F}$) if $(\neg b) \in F_s$. As assigning a objective variable to 1 corresponds to falsifying a soft clause, we will in the rest of the text view cores as sets of objective variables and extend the weight function to objective variables via $w(b) = w((\neg b))$.

III. Main Features

We overview the main features of CGSS2. For a more detailed description, we refer the reader to [10] and [9].

OLL. When solving an instance $\mathcal{F}$ the (basic form of the) OLL algorithm [11], [1] iteratively extracts unsatisfiable cores of $\mathcal{F}$ using a SAT-solver, and then reformulates the instance in a way that allows exactly one of the objective variables in the core to be assigned to 1 (corresponding to falsifying a soft clause) in subsequent iterations. This continues until the SAT solver reports the reformulated instance to be satisfiable and returns an optimal solution of the original instance.

Core reformulation. For reformulating a core $\kappa = \{b_1, \ldots, b_n\}$, the CGSS2 solver uses the so called *totalizer* [3] CNF encoding of cardinality constraints. The totalizer encoding can be viewed as a tree structure similar to the ones depicted in Figure 1. The leafs of the tree correspond to the variables in the core. An internal node that is the root of a subtree with the set $S \subseteq \kappa$ as leaves corresponds to $|S| = m$ new variables $b^S_0, \ldots, b^S_{m-1}$ defined with clauses equivalent to $\left(\sum_{b \in S} b \geq k+1\right) \rightarrow b^S_k$. Specifically the root of the full tree corresponds to a set $b^\kappa_0, \ldots b^\kappa_{n-1}$ that count the number of variables of κ set to true by assignments satisfying the totalizer.

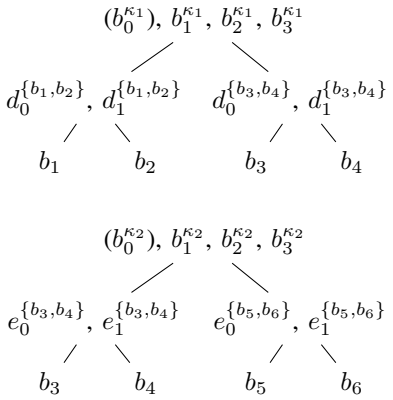

Fig. 1: The structure of totalizers built when relaxing cores $\kappa_1 = \{b_1, b_2, b_3, b_4\}$ (above) and $\kappa_2 = \{b_3, b_4, b_5, b_6\}$ (below).

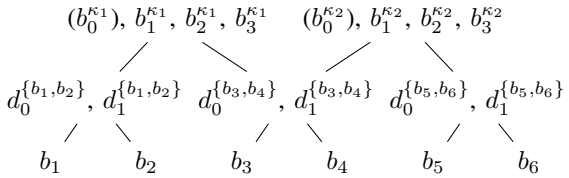

Fig. 2: The structure of totalizers when relaxing the cores κ_1 and κ_2 with structure sharing.

Weight aware core extraction (WCE) [5]. WCE is a heuristic designed to delay the core-reformulation steps performed by a solver implementing OLL for as long as possible. When extracting a new core κ, a solver using WCE will lower the weight of each variable $b \in \kappa$ by $w^\kappa = \min\{w(b) \mid b \in \kappa\}$ (this correspond to the so called clause cloning step). Afterwards, the core is stored and the SAT-solver asked for another core containing variables with positive weight. The stored cores are only reformulated when no new cores can

be found. Note that in the unweighted case (i.e. when the weight of each variable is 1) WCE is equivalent to the so called disjoint core technique that extracts a disjoint set of cores before reformulating.

Structure sharing. Structure sharing is a recently proposed refinement applied together with WCE that attempts to reduce the number of equivalent variables introduced by the core reformulation steps by identifying subtrees that can be shared between several different totalizers. For a concrete example, Figure 1 demonstrates two possible totalizer structures that can be built when relaxing the cores $\kappa_1 = \{b_1, b_2, b_3, b_4\}$ (above) and $\kappa_2 = \{b_3, b_4, b_5, b_6\}$ (below). Both of these structures include a subtree with b_3 and b_4 as leaves. The root of each of these subtrees define separate sets of variables ($\{d_0^{\{b_3,b_4\}}, d_1^{\{b_3,b_4\}}\}$ in the top tree, $\{e_0^{\{b_3,b_4\}} e_1^{\{b_3,b_4\}}\}$ in the bottom) that count the number of variables from the set $\{b_3, b_4\}$ set to true by assignments satisfying the totalizers. These variables will be assigned the same way by all satisfying assignments to the instance. Stated in another way, the two totalizer structures depicted in Figure 1 are equivalent to the smaller single structure depicted in Figure 2.

When relaxing a set of cores obtained via WCE, CGSS2 uses a heuristic set-covering algorithm for identifying maximal sets of literals shared by as many cores as possible and building totalizers that share these sets as subtrees.

Selective addition of equivalences. Consider a count variable b_k^S corresponding to an internal note of a tree that is the root of a subtree with the variables in S as leaves. For correctness of the OLL algorithm, it suffices to add clauses equivalent to the implication $\left(\sum_{b \in S} b \geq k+1\right) \rightarrow b_k^S$. While adding the other direction of the implication (i.e. $b_k^S \rightarrow \left(\sum_{b \in S} b \geq k+1\right)$) could allow the SAT solver to perform more propagation, the large number of clauses required in order to do so for every internal node might instead result in overall decrease in performance.

To balance the potential benefits and overhead (due to extra clauses) of adding both sides of the equivalence defining the variables in a totalizer, CGSS2 attempts to identify nodes for which the equivalence constraints are more likely to lead to further propagation. More specifically, for each leaf and root of a shared subtree, two values are computed: (a) the number of additional clauses needed for defining the full equivalence and (b) how many decisions need to performed by the SAT solver in before the additional constraints result in propagation. If both of these values are below some user provided threshold the equivalence constraints for that particular node are added.

Use of an IP solver. CGSS2 supports internally invoking the integer programming solver SCIP [6] with a user given time limit on a MaxSAT instance before executing the OLL algorithm. In the evaluation setting, 400 seconds is used as the time limit and SCIP is invoked only on instances where the number of variables and clauses are both less than 10^5 and the product of the number of variables and clauses is less than 10^9.

SAT solver. As an underlying SAT-solver, CGSS2 supports Glucose 3 [2] and CaDiCaL [7]. In the evaluation setting, Glucose 3 is used. On each SAT solver call, CGSS2 sorts the assumptions by their weights in decreasing order.

Upper bounds. During search, CGSS2 may find intermediate solutions to the instance. Such solutions give upper bounds on the optimal cost. CGSS2 uses the upper bound for hardening soft clauses and for checking if the search can be terminated (if known lower and upper bounds for the optimal solution match). The bound-based termination may allow termination even without relaxing all found cores.

IV. Compilation and Usage

CGSS2 can be downloaded from https://bitbucket.org/coreo-group/cgss2/src/master/ or the Evaluation website. Instructions for compilation can be found in the README.md file.

In MSE 2023 the solver is invoked with the command line

```
./cgss2 --optimizer scip [instance.wcnf]
```

References

[1] B. Andres, B. Kaufmann, O. Matheis, and T. Schaub, "Unsatisfiability-based optimization in clasp," in *Proc. ICLP Technical Communications*, ser. LIPIcs, vol. 17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, pp. 211–221.

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

[3] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of boolean cardinality constraints," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 2833. Springer, 2003, pp. 108–122.

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

[5] J. Berg and M. Järvisalo, "Weight-aware core extraction in SAT-based MaxSAT solving," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 10416. Springer, 2017, pp. 652–670.

[6] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner *et al.*, "The scip optimization suite 8.0," *arXiv preprint arXiv:2112.08872*, 2021.

[7] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Report Series B, T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2020-1. University of Helsinki, 2020, pp. 51–53.

[8] A. Ignatiev, A. Morgado, and J. Marques-Silva, "RC2: An efficient MaxSAT solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, no. 1, pp. 53–64, 2019.

[9] H. Ihalainen, "Refined core relaxations for core-guided maximum satisfiability algorithms," MSc thesis, University of Helsinki, 2022, http://hdl.handle.net/10138/351207.

[10] H. Ihalainen, J. Berg, and M. Järvisalo, "Refined core relaxation for core-guided maxsat solving," in *CP*, ser. LIPIcs, vol. 210. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 28:1–28:19.

[11] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided MaxSAT with soft cardinality constraints," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 8656. Springer, 2014, pp. 564–573.

EvalMaxSAT 2023

Florent Avellaneda
Université du Québec à Montréal
Montréal, Canada
Email: avellaneda.florent@uqam.ca

Centre de Recherche de l'IUGM
Montréal, Canada
Email: florent.avellaneda@gmail.com

INTRODUCTION

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

Here, the significant modifications made in this new version of EvalMaxSAT are presented. For a general description of how EvalMaxSAT functions, please refer to [6].

A. Code Simplification

The first major modification is the simplification of the code and the replacement of the inheritance hierarchy with the use of templates. It is important to note that to further simplify the code, this new version is single-threaded.

B. SCIP Support

During the MSE 2022 competition, the solvers UWrMaxSat [7] and Cashwmaxsat [8] showed good performance on a number of small problems by using a mixed-integer programming solver to solve them. Therefore, in this new version of EvalMaxSAT, the option to use the SCIP solver [9] to solve a formula has been added. In the version submitted to the competition, EvalMaxSAT with the SCIP solver allocates 500 seconds for instances with fewer than 100,000 variables.

C. Precalculation of UB with Incomplete Solver

A crucial technique employed by many MaxSAT solvers to tackle weighted MaxSAT formulas is the utilization of the hardening technique [10]. This technique involves deriving an upper bound on the cost of the optimal solution to deduce that certain high-weight soft clauses can be transformed into hard clauses. While the hardening technique is typically used in conjunction with a stratification technique [11], which allows for the determination of upper bounds from admissible solutions at different stratification levels, the process of obtaining high-quality upper bounds can be slow.

To address this drawback and improve the efficiency of EvalMaxSAT, the latest version introduces a precalculation phase using an incomplete solver. Specifically, EvalMaxSAT incorporates the Loandra [12] and Nuwls [13] solvers, running each for 2 minutes and 30 seconds before proceeding with the main solving process. These incomplete solvers aim to provide a higher-quality upper bound by exploring the search space and obtaining a partial solution within the given time limit.

D. Adaptive Research Core Times

The resolution time of MaxSAT solvers based on the OLL algorithm is heavily influenced by the quality of the considered unsat cores. In this regard, EvalMaxSAT distinguishes itself by dedicating significant time to optimizing the quality of unsat cores. It achieves this by making multiple calls to a SAT solver on the same formula, generating multiple unsat cores, and selecting the best one to retain [6]. Although the additional time spent searching for improved unsat cores is compensated in the long run by generating smaller cardinality cores, there needs to be a balance regarding the time allocated to optimizing unsat cores.

In this latest version of EvalMaxSAT, several criteria have been introduced to determine the time devoted to unsat core optimization, including:

- **Early stop:** After the 20th iteration, if the last improvement occurred before the $n/3$ iteration (where n represents the current iteration), the search process is halted. This mechanism prevents excessive time from being spent on optimizing unsat cores when the improvement rate becomes significantly slow. By stopping the search at this point, EvalMaxSAT avoids diminishing returns and focuses on other aspects of the solving process.
- **Diminishing optimization factor:** Since the expected benefit from discovering higher-quality unsat cores is proportional to the remaining computation time, a diminishing factor is applied to the time dedicated to unsat core optimization as time progresses.

CONCLUSION

In conclusion, this paper presented significant modifications made in the new version of EvalMaxSAT, including code simplification, SCIP solver support, precalculation of upper bounds, and adaptive research core times. These enhancements improve the efficiency and effectiveness of EvalMaxSAT in solving MaxSAT problems.

REFERENCES

[1] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Report Series B, T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2020-1. University of Helsinki, 2020, pp. 51–53.

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

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

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

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

[6] F. Avellaneda, "A short description of the solver evalmaxsat," *MaxSAT Evaluation*, vol. 8, 2020.

[7] M. Piotrów, "Uwrmaxsat entering the maxsat evaluation 2022," *MaxSAT Evaluation 2022*, p. 21.

[8] Z. Lei, S. Cai, D. Wang, Y. Peng, F. Geng, D. Wan, Y. Deng, and P. Lu, "Cashwmaxsat: Solver description," *MaxSAT Evaluation*, vol. 2021, p. 8, 2021.

[9] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 8.0," Optimization Online, Technical Report, December 2021. [Online]. Available: http://www.optimization-online.org/DB_HTML/2021/12/8728.html

[10] A. Morgado, F. Heras, and J. Marques-Silva, "Improvements to core-guided binary search for maxsat," in *Theory and Applications of Satisfiability Testing–SAT 2012: 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings 15*. Springer, 2012, pp. 284–297.

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

[12] J. Berg, "Loandra in the 2020 maxsat evaluation," *MaxSAT Evaluation 2022*, vol. 2022, 2022.

[13] Y. Chu, S. Cai, Z. Lei, and X. He, "Nuwls-c: Solver description," *MaxSAT Evaluation 2022*, p. 28.

MaxCDCL in MaxSAT Evaluation 2023

1st Chu-Min Li
Université de Picardie Jules Verne,
MIS, Amiens, France
Aix Marseille Univ, *Université de Toulon*
CNRS, LIS, Marseille, France
chu-min.li@u-picardie.fr

2rd Jordi Coll
Artificial Intelligence Research Institute
CSIC, Bellaterra, Spain
jcoll@iiia.csic.es

3rd Shuolin Li, 4th Djamal Habet
Aix Marseille Univ, *Université de Toulon*
CNRS, LIS, Marseille, France
shuolin.li@etu.univ-amu.fr,
Djamal.Habet@univ-amu.fr

5th Felip Manyà
Artificial Intelligence Research Institute
CSIC, Bellaterra, Spain
felip@iiia.csic.es

6th Kun He
Huazhong University of Science and Technology
Wuhan,China
brooklet60@hust.edu.cn

I. Introduction

MaxCDCL is an extension of CDCL (Conflict Driven Clause Learning) based on the Branch-and-Bound (BnB) scheme for unweighted (partial) MaxSAT [1]. Its main distinguishing feature w.r.t. a CDCL SAT solver is a lookahead procedure to underestimate a lower bound (LB) of the number of soft clauses that will be falsified if search continues. If LB reaches the current upper bound (UB), i.e., if LB$\geq$UB, then any solution better than the best solution found so far does not exist. This situation is called a soft conflict, from which an analysis is carried out to derive a hard clause explaining the soft conflict and to guide the backtracking, so that the same soft conflict will not be produced again in the future. When a hard clause is falsified, the conflict is said to be hard, and MaxCDCL analyzes it as in a CDCL SAT solver to learn a hard clause. The soft conflicts are together with the hard conflicts to drive the entire search, given UB.

MaxCDCL already participated in MaxSAT evaluation 2022 and was ranked 6th over 11 competitors in the unweighted partial MaxSAT track [2].

II. New techniques in MaxCDCL 2023

A. Bounded Variable Elimination and Equivalent Literal Substitution

MaxCDCL 2022 already implemented Bounded Variable Elimination (BVE) and Equivalent Literal Substitution (ELS). However, BVE was only performed in preprocessing, and ELS was only performed after the first feasible solution is found. This was a conservative approach.

In fact, MaxCDCL begins search by setting UB=1. While it does not find any feasible solution satisfying all hard clauses and falsifying less than UB soft clauses, it multiplies UB by 2 and restarts search. However, all changes made for UB are not valid for 2$\times$UB. In MaxCDCL 2022, when UB is doubled, we simply removed all learnt clauses and all variables assignments. When a feasible solution satisfying all hard clauses and falsifying less than UB soft clauses is found, let k be the number of falsified soft clauses, MaxCDCL sets k as the new UB. All changes made for UB, including ELS, are also valid for UB$\leftarrow k$, because $k <$UB. So, it is valid to do preprocessing BVE by setting UB$= \infty$ and ELS after finding the first feasible solution.

In 2023, MaxCDCL saves all original clauses and the relevant information for variables, which are restored each time before UB is increased. In this way, inprocessing BVE and ELS are applied for all UBs, because all clauses, including original clauses, possibly modified in search for smaller UB, are simply removed before increasing UB. On the contrary, when UB is decreased, the modified clauses are kept.

Note that variables occurring in soft clauses are never removed. If a soft clause s is equivalent to a literal h such that h or $\neg h$ does not occur in any soft clause, then h is replaced by s in all hard clauses. If two soft clauses s_1 and s_2 are complementary, i.e., s_1 is satisfied if and only if s_2 is falsified, then the constant cost is increased by 1, and both s_1 and s_2 are removed. If s_1 and s_2 are equivalent, nothing is done, which is different from weighted partial MaxSAT for which s_1 and s_2 could be unified by removing one of them and the sum of their weights becomed the weight of the remaining one.

Inprocessing BVE is called when the total number of literals in original hard clauses is decreased by 1% or 1% variables are assigned at level 0. The variables are checked and eliminated in increasing order of their activity, as proposed in [3]. Since MaxCDCL switches between LRB [4] and VSIDS [5] to define variable activity, the activity of a variable is the current LRB (VSIDS) score used for selecting decision variables.

B. Learnt clause deletion based on variable activity

Similar to a CDCL SAT solver, MaxCDCL learns a new clause from each soft conflict and each hard conflict. These learnt clauses are organized into three subsets as in its base SAT solver Maple_CM [6]: CORE, TIER2 and LOCAL according to their LBD as defined in [7], the learnt clauses in LOCAL having large LBD and being useful only locally.

In MaxCDCL 2022, as well as in its base SAT solver Maple_CM, the clauses in LOCAL are sorted periodically in the increasing order of their activity and the first half is removed, the activity of a clause roughly corresponding to the number of times the clause is used in the recent conflicts.

MaxCDCL 2023 re-defines the activity of each clause c in LOCAL as follows. Let minAct (minAct2) be the (second) smallest variable activity in c. The activity of c is defined to be (1000×minAct+minAct2)/(LBD×LBD). This idea is inspired from [3] in which variables are eliminated in the increasing order of their activity in inprocessing BVE, so that variables with high activity are protected from being eliminated. The intuition is that a clause not frequently used in recent conflicts can be useful (i.e., can easily become unit or falsified) in the future and should not be removed if the activities of its variables are high. On the contrary, a clause containing two small activity variables will be unlikely useful and can be removed without hurting search.

C. Learnt clause vivification based on variable activity

The authors of [6], [8] asked and answered three questions when vivifying clauses: (1) when should the clause vivification be performed? (2) what are the clauses to be vivified in a clause vivification? (3) What is the order to propagate the literals when vivifying a clause? However, there is a question that is never asked nor answered in our knowledge: in a set of clauses, which clause should be vivified first?

This question is very important, because a clause simplified, i.e., one or more literals are removed by vivification, can help vivify other clauses. The intuition is that the clauses having greater probability to be simplified should be vivified first. The issue then becomes how to estimate the probability with which a clause can be simplified. Nevertheless, we do not need the real value of this probability. What we need in clause vivification is the order of this probability w.r.t. other clauses.

We use the minimum variable activity in a clause to compare the probability with which the clause can be simplified. Intuitively, if the variables of a clause all have high activities, the clause might have great probability to be simplified in a clause vivification. Consequently, MaxCDCL 2023 vivifies the clauses in a set in the decreasing order of their minimum variable activity. This idea is also inspired from [3].

D. Failed literal detection based on variable activity

Periodically, failed literal detection is performed before a slected restart at level 0. Since this detection is time-consuming, only a small subset of literals is detected. A question is then how to select this subset of literals to detect. MaxCDCL 2023 selects the 500 variables with higher activity and for each variable x detects two literals x and $\neg x$. Intuitively, these literals have high probability to be failed. In addition, even if x, as well as $\neg x$, is not failed, we can discover easily those literals l such that $x \rightarrow l$ and $\neg x \rightarrow l$, or $x \rightarrow l$ and $\neg x \leftarrow \neg l$. In the former case, l can be satisfied immediately, and in the second case, x and l are equivalent literals to be treated by equivalent literal substitution.

III. Versions of MaxCDCL in MaxSAT evaluation 2023

We submit three versions of MaxCDCL to MaxSAT evaluation 2023. The first version is pure MaxCDCL without using any third-party solver. The second version first calls SCIP [9] for 10 minutes, then MaxHS [10] for 15 minutes, and finally pure MaxCDCL for 35 minutes. If the instance is not solved by SCIP in 10 minutes, nor by MaxHS in 15 minutes, MaxCDCL reads the best UB obtained by MaxHS (if any) and uses it as the initial UB (initUB). Recall that MaxCDCL begins search by setting UB=1, then while UB is not feasible, sets UB$\leftarrow min(2 \times UB, initUB + 1)$, and then while UB is feasible and k is the number of falsified soft clauses under UB, sets UB$\leftarrow k$. The last infeasible UB is the optimal solution. The third version is the same as the second one but giving 20 minutes to MaxHS and 30 to pure MaxCDCL.

Acknowledgments

This work has been partially supported by AI CHAIR reference ANR-19-CHIA-0013-01 (MASSAL'IA) and project ANR-20-ASTR-0011 (POSTCRYPTUM) funded by the French Agence Nationale de la Recherche, and projects PID2019-111544GB-C21 and TED2021-129319B-I00 funded by MCIN/AEI/10.13039/501100011033, and partially supported by Archimedes Institute, Aix-Marseille University. We thank the Université de Picardie Jules Verne for providing the Matrics Platform.

References

[1] C.-M. Li, Z. Xu, J. Coll, F. Manyà, D. Habet, and K. He, "Combining clause learning and branch and bound for maxsat," in *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, 2021.

[2] J. Coll, S. Li, C.-M. Li, F. Manya, D. Habet, and K. He, "Maxcdcl and wmaxcdcl in maxsat evaluation 2022," in *MaxSAT Evaluation 2022 : Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2022, pp. 15–16.

[3] S. Li, C.-M. Li, M. Luo, J. Coll, D. Habet, and F. Manya, "A new variable ordering for in-processing bounded variable elimination in sat solvers," in *Proceedings of the 32nd International Joint Conference on Artificial Intelligence*, 2017, pp. 703–711.

[4] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for sat solvers," in *Theory and Applications of Satisfiability Testing–SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings 19*. Springer, 2016, pp. 123–140.

[5] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik, "Efficient conflict driven learning in a Boolean satisfiability solver," in *Proccedings of the IEEE/ACM International Conference on Computer Aided Design, ICCAD*, 2001, pp. 279–285.

[6] C.-M. Li, F. Xiao, M. Luo, F. Manya, Z. Lu, and Y. Li, "Clause vivification by unit propagation in cdcl sat solvers,," *Artificial Intelligence, volume 279*, p. 103197, 2020.

[7] L. Simon and G. Audemard, "Predicting learnt clauses quality in modern sat solvers," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009, pp. 399–404.

[8] M. Luo, C.-M. Li, F. Xiao, F. Manya, and Z. Lü, "An effective learnt clause minimization approach for cdcl sat solvers," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 2017, pp. 703–711.

[9] K. Bestuzheva et al., "The SCIP Optimization Suite 8.0," Optimization Online, Technical Report, December 2021.

[10] F. Bacchus, "MaxHS in the 2021 MaxSAT Evaluation," *MaxSAT Evaluation 2021*, p. 14, 2021.

WMaxCDCL in MaxSAT Evaluation 2023

1st Jordi Coll
Artificial Intelligence Research Institute
CSIC, Bellaterra, Spain
jcoll@iiia.csic.es

2nd Shuolin Li
Aix Marseille Univ,Université de Toulon
CNRS, LIS, Marseille, France
shuolin.li@etu.univ-amu.fr

3rd Chu-Min Li
Université de Picardie Jules Verne,
Amiens, France
Aix Marseille Univ,Université de Toulon
CNRS, LIS, Marseille, France
chu-min.li@u-picardie.fr

4th Felip Manyà
Artificial Intelligence Research Institute
CSIC, Bellaterra, Spain
felip@iiia.csic.es

5th Djamal Habet, 6th Mohamed Sami Cherif
Aix Marseille Univ,Université de Toulon
CNRS, LIS, Marseille, France
Djamal.Habet@univ-amu.fr, mohamed-sami.cherif@univ-amu.fr

7th Kun He
Huazhong University of Science and Technology
Wuhan,China
brooklet60@hust.edu.cn

I. Introduction

WMaxCDCL participated for first time in MaxSAT Evaluation 2022. Its main algorithm is MaxCDCL [1], an algorithm that combines Branch and Bound and clause learning. WMaxCDCL solves Weighted Partial MaxSAT, and its first version was developed as a modification of the MaxCDCL solver for unweighted Partial MaxSAT from [1]. Both MaxCDCL and WMaxCDCL have been independently updated since the version in [1], though some common features have been implemented in both solvers.

II. WMaxCDCL Algorithm

The MaxCDCL algorithm is an extension for MaxSAT of the CDCL algorithm which combines Branch and Bound and clause learning. Similarly as done in CDCL, the MaxCDCL algorithm roughly alternates decisions and unit propagation with conflict analysis and clause learning. Moreover, at some selected nodes of the search tree, MaxCDCL computes a lower bound (LB) of the number of soft clauses that will be falsified in any solution that satisfies the hard clauses. If the bounding procedure detects that the current assignment cannot be extended to a satisfying assignment that improves the best solution found so far, i.e. $LB \geq UB$, a *soft conflict* is detected. Similarly to (hard) conflicts in CDCL, which can also occur in MaxCDCL, and where a hard clause is falsified, MaxCDCL detects an implicit clause that is falsified when a soft conflict occurs. Both after hard and soft conflicts, conflict analysis is used to find the first unique implication point and backtrack. In addition, when the lower bounding procedure does not detect a soft conflict but $LB + \omega(c) \geq UB$, for some soft clause c with weight $\omega(c)$, such soft clause can be hardened. This hardening is done by unit propagation after introducing new clauses explaining the reason of the hardening.

The computation of the lower bound is based on the detection of local unsatisfiable cores, i.e. cores that depend on the current partial assignment. Roughly, the detection of a local core is done by assuming soft clauses to be true and applying unit propagation until some conflict is found [2]–[4]. For every detected local core (set of soft clauses), the lower bound can be increased by the minimum weight of its soft clauses.

III. Implementation Details

The solver first tries to find an initial feasible cost, by solving the problem with a sequence of increasing upper bound values UB starting at 1, and increased by $UB \leftarrow min(1.5 \cdot UB, initUB)$ until a feasible solution is found or *initUB* is reached. Here, *initUB* is either a trivial upper bound (sum of weights of soft clauses) or an upper bound computed externally and received as input. Then, the optimization procedure continues by decreasing UB until the optimal solution is found and proven.

Before starting the search we find incompatible subsets of soft clauses by unit propagation, i.e. sets of soft clauses such that at most one of them can be satisfied according to hard clauses. Then, for every set of weighted clauses $(c_1, \omega(c_1)), \ldots, (c_n, \omega(c_n))$, we derive a fixed cost of $m(n-1)$, where m is the minimum of the weights $\omega(c_1), \ldots, \omega(c_m)$.

When the number of free soft clauses n and the upper bound UB are small, we add as implied constraints a CNF encoding of pseudo-Boolean constraints, expressing that the cost of the solution must be smaller than the best found upper bound. In particular, we add the MDD encoding [5] when $n \leq 50$ and $n \cdot K \leq 10^5$, and otherwise the GGPW encoding [6] when $n \leq 500$ or $n \leq 5000$ and $n \cdot K \leq 10^5$.

We also implement a number of inprocessing algorithms, namely failed literal detection, equivalent literal substitution,

clause simplification, and solution improvement with local search by means of a custom implementation of the local search method described in [7].

IV. WMaxCDCL in MaxSAT Evaluation 2023

Three versions of WMaxCDCL are submitted in MaxSAT Evaluation 2023.

The first one is the pure WMaxCDCL solver without using any third-party solver.

The second one precedes the execution of WMaxCDCL with, first, 10 minutes of the SCIP solver [8]. If no solution is found, then the MaxHS solver [9] is run for 20 more minutes. Finally, if the instance is not solved, WMaxCDCL is run for the remaining time, taking the last solution found by MaxHS as *initUB*. In order to run the SCIP solver, we use the UWrMaxSAT-SCIP implementation [10], which includes instance parsing and preprocessing. We specify that the 900 seconds must used by SCIP, unless UWrMaxSAT-SCIP internally decides to stop or avoid the SCIP computation and continue with other solving techniques.

The third version is the same as the second one but giving 15 minutes to SCIP and 15 minutes to MaxHS.

Acknowledgments

This work has been partially supported by AI CHAIR reference ANR-19-CHIA-0013-01 (MASSAL'IA) and project ANR-20-ASTR-0011 (POSTCRYPTUM) funded by the French Agence Nationale de la Recherche, and projects PID2019-111544GB-C21 and TED2021-129319B-I00 funded by MCIN/AEI/10.13039/501100011033, and partially supported by Archimedes Institute, Aix-Marseille University. We thank the Université de Picardie Jules Verne for providing the Matrics Platform.

References

[1] C.-M. Li, Z. Xu, J. Coll, F. Manyà, D. Habet, and K. He, "Combining clause learning and branch and bound for maxsat," in *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, 2021.

[2] C. M. Li, F. Manyà, N. O. Mohamedou, and J. Planes, "Exploiting cycle structures in Max-SAT," in *In Proceedings of SAT 2009*, ser. LNCS, vol. 5584. Springer, 2009, pp. 467–480.

[3] C. M. Li, F. Manyà, and J. Planes, "Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers," in *Proceedings of CP 2005*, ser. LNCS, vol. 3709. Springer, 2005, pp. 403–414.

[4] ——, "Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT," in *Proceedings of AAAI 2006*, 2006, pp. 86–91.

[5] M. Bofill, J. Coll, J. Suy, and M. Villaret, "An mdd-based SAT encoding for pseudo-boolean constraints with at-most-one relations," *Artificial Intelligence Review*, vol. 53, no. 7, pp. 5157–5188, 2020.

[6] M. Bofill, J. Coll, P. Nightingale, J. Suy, F. Ulrich-Oltean, and M. Villaret, "SAT encodings for pseudo-boolean constraints together with at-most-one constraints," *Artificial Intelligence*, vol. 302, p. 103604, 2022.

[7] J. Zheng, K. He, J. Zhou, Y. Jin, C.-M. Li, and F. Manya, "Bandmaxsat: A local search MaxSAT solver with multi-armed bandit," in *Proceedings of 31st International Joint Conference on Artificial Intelligence (To appear)*, 2022.

[8] K. Bestuzheva et al., "The SCIP Optimization Suite 8.0," Optimization Online, Technical Report, December 2021.

[9] F. Bacchus, "Maxhs in the 2022 maxsat evaluation," *MaxSAT Evaluation 2022*, pp. 17–18, 2022.

[10] M. Piotrów, "Uwrmaxsat entering the maxsat evaluation 2022," *MaxSAT Evaluation 2022*, pp. 21–22, 2022.

Open-WBO @ MaxSAT Evaluation 2023

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

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

Miguel Terra-Neves[1], Vasco Manquinho[2], Inês Lynce[2]
{neves,vmm,ines}@inesc-id.pt
OutSystems[1], INESC-ID/IST[2], Portugal

I. Introduction

Open-WBO [1] is an open source MaxSAT solver that supports several MaxSAT algorithms [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] and SAT solvers [13], [14], [15]. Open-WBO is particularly efficient for unweighted MaxSAT, and although it is not as competitive in the last years, it was one of the best solvers in the MaxSAT Evaluations from 2014 to 2017. Two versions of Open-WBO were submitted to the unweighted track at MaxSAT Evaluation 2023: open-wbo-oll [7] and open-wbo-res [7], [4], [10]. Both versions use the modified parser with support to the new format where hard clauses are marked with 'h' and the 'p-line' is removed. The remainder of this document describes the differences between these versions.

II. SAT solvers

Open-WBO is based on the data structures of MiniSat 2.2 [13], [16]. Therefore, solvers based on MiniSat 2.2 can be used as a potential back-end solvers. For the MaxSAT Evaluation 2023, we use MergeSAT as the back-end SAT solver of both versions open-wbo-oll and open-wbo-res.

MergeSAT is a new CDCL solver developed by Norbert Manthey and it is based on the SAT competition winner of 2018, MapleLCMDistChronoBT [17], and adds several known techniques. For restarts, only partial backtracking is used, learned clause minimization is implemented more efficiently, and also applies simplification again in case the first swipe resulted in a simplification. Finally, the time-based decision heuristic switch is made deterministic by using solving steps. To support being used inside MaxSAT solvers, the incremental search feature had to been enabled again.

III. MaxSAT Algorithms

In this section we briefly describe the algorithms used for the complete and incomplete tracks at the MaxSAT Evaluation 2023.

A. Complete Unweighted Track

Two versions were submitted to the complete unweighted track: open-wbo-oll [7] and open-wbo-res [7], [4], [10].

open-wbo-oll uses the OLL algorithm [7], whereas open-wbo-res [7], [4], [10] uses a variant of the unsatisfiability-based algorithm MSU3 [3], [4], [10]. These algorithms work by iteratively refining a lower bound λ 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].

More specifically, the open-wbo-res version uses the resolution-based partitioning techniques [10] by default. We represent a MaxSAT formula using a resolution-based graph representation and iteratively join partitions by using a proximity measure extracted from the graph representation of the formula. The algorithm ends when only one partition remains, and the optimal solution is found. Since the partitioning of some MaxSAT formulas may be unfeasible or not significant, we heuristically choose to run either MSU3 with partitions or the OLL algorithm. In particular, we do not use partition-based techniques when one of the following criteria is met: (i) the formula is too large ($>$ 1,000,000 clauses), (ii) the ratio between the number of partitions and soft clauses is too high (> 0.8), (iii) the sparsity of the graph is too small (< 0.04), or (iv) there exist some at-most-one relations between soft clauses (> 10), i.e., if one soft clause is satisfied it implies that some other soft clauses will be unsatisfied.

B. Preprocessing

We perform identification of unit cores and at-most-one relations between soft clauses by using unit propagation. A similar technique is done in RC2 [18], the winner of the MaxSAT Evaluation 2018.

C. Other

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

IV. Availability

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

Acknowledgments

We would 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. Finally, we thank David Chen for his study on the impact of disjoint cores, unit cores, and at-most-one relations between soft clauses that were done in the scope of Independent Studies at CMU.

REFERENCES

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

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

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

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

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

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

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

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

[9] R. Martins, V. Manquinho, and I. Lynce, "Improving linear search algorithms with model-based approaches for MaxSAT solving," *J. Exp. Theor. Artif. Intell.*, vol. 27, no. 5, pp. 673–701, 2015.

[10] 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.

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

[12] S. Joshi, P. Kumar, S. Rao, and R. Martins, "Open-WBO-Inc: Approximation Strategies for Incomplete Weighted MaxSAT," in *JSAT*. IOS Press, 2019.

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

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

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

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

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

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

Pacose: An Iterative SAT-based MaxSAT Solver

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

I. Overview

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

II. Pre- / Inprocessing

The 2023 version of Pacose contains the preprocessing tool maxpre version 2 [5] and performs our own preprocessing routines Generalized Boolean Multilevel Optimization (GBMO) and TrimMaxSAT [6].

MaxPre version 1 [7], has already been utilized by several MaxSAT solvers in past competitions. Preprocessing has gained increasing importance in recent years, with many top solvers from previous years adopting MaxPre and other techniques to achieve favorable results. The successful outcomes reported in [5] have further persuaded us to incorporate version 2 of this preprocessor into our own implementation.

We generalized the plain variant of Boolean Multilevel Optimization to work with arbitrary weights and split additional instances with that. Further we implemented a greedy algorithm TrimMaxSAT to remove never satisfiable instances and getting first upper / lower bounds.

III. Encoding and Algorithm

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

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

IV. Heuristics

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

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

References

[1] J. P. Warners, "A linear-time transformation of linear inequalities into conjunctive normal form," *Information Processing Letters*, vol. 68, no. 2, pp. 63–69, 1998.

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

[3] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A partial Max-SAT solver system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 8, pp. 95–100, 2012.

[4] G. Audemard and L. Simon, "On the glucose SAT solver," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 01, p. 1840001, 2018.

[5] H. Ihalainen, J. Berg, and M. Järvisalo, "Clause redundancy and preprocessing in maximum satisfiability," in *Automated Reasoning: 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8–10, 2022, Proceedings*. Springer, 2022, pp. 75–94.

[6] T. Paxian, P. Raiola, and B. Becker, "On preprocessing for weighted MaxSAT," in *Verification, Model Checking, and Abstract Interpretation: 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17–19, 2021, Proceedings 22*. Springer, 2021, pp. 556–577.

[7] T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo, "Maxpre: an extended maxsat preprocessor," in *Theory and Applications of Satisfiability Testing–SAT 2017: 20th International Conference, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 20*. Springer, 2017, pp. 449–456.

[8] O. Bailleux, Y. Boufkhad, and O. Roussel, "New encodings of pseudo-boolean constraints into CNF," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2009, pp. 181–194.

[9] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of Boolean cardinality constraints," in *Principles and Practice of Constraint Programming–CP 2003*. Springer, 2003, pp. 108–122.

Loandra in the 2022 (and 2023) MaxSAT Evaluation

Jeremias Berg
HIIT, Department of Computer Science, University of Helsinki, Finland

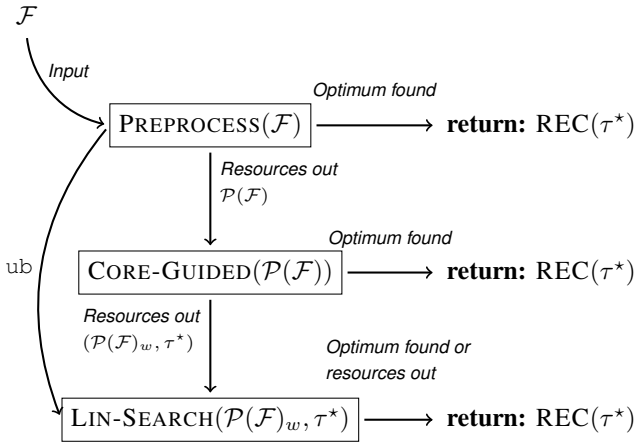

Fig. 1: The structure of Loandra.

I. Preliminaries

We briefly overview the any-time Loandra MaxSAT-solver as it participated in the incomplete track of the 2022 MaxSAT Evaluation, focusing especially on the differences to the 2019 and 2020 versions. All of the new changes to Loandra relate to the preprocessing phase of the algorithm. In particular, the solver now employs a recent extension of MaxPRE (named MaxPRE 2.0) capable of stronger reasoning as well as outputting an upper bound $\texttt{ub}$ on the optimal cost. More detailed descriptions can be found in [4], [11], [10].

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

Loandra is implemented on top of Open-WBO [12]. We thank the developers of Open-WBO for their work.

II. Structure of Loandra

Figure 1 overviews the structure of Loandra. The solver implements core-boosted linear search [4] augmented with tightly integrated MaxSAT preprocessing [3], [10], [11], [2]. More specifically, Loandra consists of three main components: a) *Preprocessing*, b) *Core-guided search* and c) *Linear search*.

a) Preprocessing: On input $\mathcal{F}$, the execution starts by invoking the MaxPre 2.0 [10] preprocessor on $\mathcal{F}$. MaxPre 2.0 is run with the technique string `[u]#[uvsrgVGc]`, enforcing a 30s time-limit on and a skip technique value of 20. In more detail, the preprocessor runs the same "base" techniques as in previous years (unit propagation, bounded variable elimination, subsumption elimination, self-subsuming resolution, group subsumed label elimination and binary core removal) as well as the so called intrinsic at-most-one and TrimMaxSAT techniques [8], [15]. The TrimMaxSAT technique is extended to all literals rather than only literals appearing in soft clauses.

In addition to the more expressive preprocessing rules, another novelty of applying MaxPRE 2.0 is the possibility of obtaining and upper bound $\texttt{ub}$ on $\text{COST}(\mathcal{F})$. The bound is supplied to the linear search phase. Unless MaxPre can compute an optimal solution to $\mathcal{F}$, the preprocessed instance $\mathcal{P}(\mathcal{F})$ is then handed to the core guided phase, reusing the assumption variables introduced during preprocessing [3].

b) Core-guided search: CORE-GUIDED, the core-guided phase is unchanged from previous versions of Loandra. As the instantiation of the core-guided algorithm, we use a reimplementation of PMRES [14] extended with weight aware core extraction (WCE) [5] and clause hardening. If CORE-GUIDED is able to find an optimal solution τ to $\mathcal{P}(\mathcal{F})$, an optimal solution $\text{REC}(\tau)$ to $\mathcal{F}$ is reconstructed and returned. Otherwise the final working instance $\mathcal{P}(\mathcal{F})_w$ and the best found solution τ^* are handed to the linear search component.

c) Linear search: LIN-SEARCH, the linear search phase of Loandra is an implementation of the SAT/UNSAT linear search algorithm [6], extended with solution guided phase saving and varying resolution in the style of LinSBPS [7]. The component is for the most part the same as in the 2019 version. As the pseudo-Boolean encoding, we use the so called generalized totalizer [9]. The initial bound $B = \min\{\texttt{ub}, \text{COST}(\mathcal{F}, \tau^*)\}$ on PB-encoding is set to the minimum of the upper bound found by the preprocessor and the cost of the best solution found by the core-guided phase. Note that the linear search phase operates on the working instance of the core-guided search. As such, the range over which it searches is $[\texttt{lb}, B]$ where $\texttt{lb}$ is the lower bound obtained by the core-guided phase. The lower bound is implicitly maintained in the transformed formula, meaning that in practice, the PB constraint is built over the range $[0, B - \texttt{lb}]$.

In the beginning of each resolution (i.e. invocation of linear

search on a subset of the soft clauses), the best known solution τ^* is minimized in order to alleviate the missinterpretation of costs that might happen due to preprocessing in the context of incomplete solving [11]. The minimization procedure resembles ideas proposed in MaxSAT solving algorithms based on bit-vector optimization [13]. In short, the procedure loops over all literals in the objective funciton, attempting to assign an increasing number of them to false (i.e. to not incur cost).

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

III. Implementation details

All algorithms are implemented on top of the publicly available Open-WBO system [12] using Glucose 4.1 [1] as the back-end SAT solver. In order to minimize I/O overhead, we make direct use of the preprocessor interface offered by MaxPre. The linear search algorithm uses the generalized totalizer encoding [9] to convert the PB constraints needed in linear search to CNF. In the evaluation, we set a 30s time limit for the preprocessing phase and a 30 second time limit for the core-guided phase. These limits were chosen based on preliminary experiments. On weighted instances, the core-guided phase is also terminated when the stratification bound would be lowered to 1. On unweighted instances the phase is terminated at the latest after extracting one set of disjoint cores.

IV. Compilation and Usage

Building and using Loandra resembles building and using Open-WBO. A statically linked version of Loandra in release mode can be built by running MAKE RS in the base folder.

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

V. Acknowledgments

The algorithmic techniques underlying Loandra have been developed in collaboration with a number of different people. The initial work on the solver was done together with Emir Demirović and Peter Stuckey [4]. Other contributors to Loandra include Matti Järvisalo, Marcus Leivo, Tuukka Korhonen, and Hannes Ihalainen [11], [10]. The primary developer is supported by the Academy of Finland under grant 342145. My sincerest thanks to everyone who has contributed to the algorithmic ideas underlying Loandra.

References

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

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

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

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

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

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

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

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

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

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

[11] M. Leivo, J. Berg, and M. Järvisalo, "Preprocessing in incomplete maxsat solving," in *ECAI*, ser. Frontiers in Artificial Intelligence and Applications, vol. 325. IOS Press, 2020, pp. 347–354.

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

[13] A. Nadel, "Anytime weighted maxsat with improved polarity selection and bit-vector optimization," in *FMCAD*. IEEE, 2019, pp. 193–202.

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

[15] T. Paxian, P. Raiola, and B. Becker, "On preprocessing for weighted maxsat," in *VMCAI*, ser. Lecture Notes in Computer Science, vol. 12597. Springer, 2021, pp. 556–577.

NuWLS-c-2023: Solver Description

Yi Chu[1], Shaowei Cai[2,3], and Chuan Luo[4]

[1] *Institute of Software, Chinese Academy of Sciences, Beijing, China*
[2]*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China*
[3] *School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China*
[4] *School of Software, Beihang University, Beijing, China*

***Abstract*—This document serves as a description of our solver, NuWLS-c-2023, which has been submitted to all four incomplete tracks of the MaxSAT Evaluation 2023.**

I. Introduction

Our NuWLS-c-2023 solver is an improved version of the NuWLS-c solver that participated in the MaxSAT Evaluation 2022. Like SATLike-c [1], [2], NuWLS-c-2023 includes two engines - one is our proposed stochastic local search (SLS) algorithm NuWLS-2.0, which introduces a new weighting scheme for soft and hard constraints; the other is the SAT-based algorithm TT-Open-WBO-Inc [3].

II. The SLS algorithm: NuWLS-2.0

A. Preliminaries

For each clause c, we use $w(c)$ to denote the weight of clause c. For each soft clause c, we use $w_{org}(c)$ to denote the original weight of c, which is given in the instance. avg_{softw} is used to denote the average original soft clause weights. It is evident that for unweighted instances, the values of $w_{org}(c)$ and avg_{softw} are always equal to 1.

B. The New Weighting Scheme

In the literature [4], it is noted that developing a weighting scheme for problems with hard constraints is challenging as it involves weighing unsatisfied constraints while preserving the differentiation between hard and soft constraints. To balance the relationships between hard and soft clause weights, we designed the following weighting scheme, named **New-Weighting**.

At the beginning of each round of the local search process, the New-Weighting scheme initializes each clause weight as follows:

- For each hard clause c, we set $w(c) := 1$.
- For each soft clause c, we set $w(c) := n \times 1$ for unweighted instances, and $w(c) := n \times \frac{w_{org}(c)}{avg_{softw}}$ for weighted instances, where n is initialized to 0.

When the local search encounters a local optimum, the clause weights are updated as follows:

- For hard clauses: for each falsified hard clause c, $w(c) := w(c) + h\_inc$.
- For soft clauses: if the current assignment α is a feasible solution, then $n := n + s\_inc$.

n is a coefficient dynamically adjusted in the weighting scheme to control the weights of soft clauses. h_inc is the increment for hard clause weight. s_inc is the increment for the coefficient n.

Algorithm 1: NuWLS-2.0

```
Input: (W)PMS instance F, cutoff.
Output: The best solution found and its cost, or "No
        solution found".
α* := ∅; cost* := +∞;
while elapsed time<cutoff do
    α := an initial complete assignment;
    Initialize clause weights by New-Weighting;
    L = 10000000;
    for step = 0; step < L; step++ do
        if α is feasible and cost* > cost(α) then
            α*:=α; cost*:=cost(α); L=step+10000000;
            if cost*==0 then
                return α* and cost*;
        if (D = {x|score(x) > 0}) ≠ ∅ then
            v := a variable in D selected by BMS strategy;
        else
            update clause weights by New-Weighting;
            if ∃ falsified hard clauses then
                c := a random falsified hard clause;
            else
                c := a random falsified soft clause;
            v := the variable with highest score in c;
        α := α with v flipped;
if α* ≠ ∅ then return α* and cost*;
else return No solution found;
```

Our weighting scheme treats all soft clauses as a single constraint, and it exhibits several novel features. First, previous weighting schemes for (weighted) partial MaxSAT ((W)PMS) problem either did not change the weights of soft clauses, or treated them the same as hard clauses by increasing the weight of unsatisfied soft clauses or decreasing the weight of satisfied soft clauses. Our weighting scheme increases the weights of all soft clauses equally, regardless of whether they are satisfied or not. Second, our weighting scheme eliminates the need for setting upper bounds on the weights of soft clauses (Weighting-PMS [5]). Instead, it limits the increase of soft clause weights by imposing certain conditions (under the condition that the current assignment is a feasible solution).

Third, this weighting scheme preserves the relative importance of weights among soft clauses in weighted instances.

C. Our NuWLS-2.0 Algorithm

Based on the **New-Weighting** scheme and the dynamic local search (DLS) framework, we develop a new SLS algorithm named NuWLS-2.0. The pseudo-code of NuWLS-2.0 is outlined in algorithm 1.

III. The Hybrid Solver: NuWLS-c-2023

We combine NuWLS-2.0 with the state-of-the-art SAT-based solvers TT-Open-WBO-Inc [3], leading to the hybrid solver NuWLS-c-2023. The framework of NuWLS-c-2023 is similar to SATLike-c [1], [2].

References

[1] Zhendong Lei, Shaowei Cai, SATLike-c: Solver description. MSE'18

[2] Zhendong Lei, Shaowei Cai, Fei Geng, et al., SATLike-c: Solver description. MSE'21

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

[4] Thornton John, and Abdul Sattar. "Dynamic constraint weighting for over-constrained problems." PRICAI. Vol. 98. 1998.

[5] Shaowei Cai, Zhendong Lei, "Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability". Artif. Intell. 287: 103354 (2020)

Combining BandMaxSAT and FPS with NuWLS-c

Jiongzhi Zheng[1,2] Kun He[1,2] Mingming Jin[1,2] Zhuo Chen[1,2] Jinghui Xue[1,2]
[1]School of Computer Science and Technology, Huazhong University of Science and Technology, China
[2]Hopcroft Center on Computing Science, Huazhong University of Science and Technology, China
{jzzheng,brooklet60,mingmingk,ciaozer,jh_xue}@hust.edu.cn

***Abstract*—This document describes our MaxSAT solvers NuWLS-c-Band and NuWLS-c-FPS submitted to the MSE 2023. NuWLS-c-Band and NuWLS-c-FPS are submitted to both the weighted and unweighted incomplete tracks.**

I. Introduction

Recently, many efficient local search algorithms for the MaxSAT have been proposed, such as SATLike [1] and NuWLS [2]. SATLike and NuWLS mainly benefit from their clause weighting schemes. The anytime MaxSAT solvers based on them, i.e., SATLike-c and NuWLS-c, combine them with a famous SAT-based solver, TT-Open-WBO-Inc [3]. Both SATLike-c and NuWLS-c show excellent performance in recent MaxSAT Evaluations, indicating that combining effective local search strategies with SAT-based solver can result in better performance.

Different from SATLike and NuWLS, we propose some new and effective variable selection strategies for local search MaxSAT methods and develop two new local search algorithms, BandMaxSAT [4] and FPS [5]. BandMaxSAT associates a multi-armed bandit with the soft clauses and tries to help the algorithm learn to select to satisfy appropriate soft clauses. FPS combines probabilistic sampling and two-level look-ahead flipping methods. Both of them are generic variable selection strategies and can be used to improve various local search MaxSAT solvers, including SATLike and NuWLS.

We combine BandMaxSAT and FPS with the NuWLS-c solver, resulting in NuWLS-c-Band and NuWLS-c-FPS, respectively.

II. BandMaxSAT

BandMaxSAT [4] associates a multi-armed bandit to the soft clauses. Each arm corresponds to a soft clause. BandMaxSAT uses the bandit model to select the search direction to escape from feasible local optima. Note that a local optimum indicates that there is no variable with positive *score*, i.e., flipping any variable cannot improve the current solution. A feasible local optimum indicates there is no falsified hard clause.

The procedure of BandMaxSAT is shown in Algorithm 1. The function $cost(A)$ equals the total weight of soft clauses falsified by A if A is feasible, otherwise equals $+\infty$. When BandMaxSAT does not fall into local optima, the algorithm selects to flip a variable with positive *score* by the sampling method called Best from Multiple Selections (BMS), which chooses k (15 by default) random variables with positive *score* (with replacement) and returns one with the highest *score* (lines 5-6). When an infeasible local optimum is reached, the algorithm first randomly samples a falsified hard clause, then selects to flip the variable with the highest *score* in the clause (lines 9-10).

When a feasible local optimum is reached, the algorithm selects to pull an arm by the PickArm() function (line 14). Since the number of arms in the bandit model equals to the number of soft clauses, which is usually very large, selecting the best arm among all the arms is inefficient. Thus, we apply the sampling strategy to reduce the selection scope. Specifically, the PickArm() function first randomly samples $ArmNum$ (20 by default) arms which are all corresponding to falsified soft clauses, then selects the sampled arm with the largest value of Upper Confidence Bound (UCB). After that, BandMaxSAT selects to flip the variable with the highest *score* in the soft clause corresponding to the selected arm. The UCB of each arm i is represented by U_i, which can be calculated as follows.

$$U_i = V_i + \lambda \cdot \sqrt{\frac{ln(N)}{t(i)+1}}, \tag{1}$$

where N indicates the number of times fallen into a feasible local optimum, V_i is the estimated value of arm i, $t(i)$ is the number of times that arm i has been selected, and λ (1 by default) is the exploration bias parameter.

The estimated value of each arm is initialized to 1 at the beginning of the algorithm. The estimated values are updated by the function update_estimated_value() function in line 12. Since the arms (i.e., soft clauses) are connected by the variables, we assume that the arms in our bandit model are not independent of each other. We also believe that the improvement (or deterioration) of A over A' may not only be due to the last action, but also due to earlier actions. Hence, we apply the delayed reward method to update the estimated value of the last d (20 by default) pulled arms once a reward is obtained. Specifically, suppose that A' and A are the last and current feasible local optimal solutions respectively, A^* is the best solution found so far, and $\{a_1, ..., a_d\}$ is the set of the latest d pulled arms (a_d is the most recent one). Then, the estimated values of the d arms are updated as follows:

$$V_{a_i} = V_{a_i} + \gamma^{d-i} \cdot \frac{cost(A') - cost(A)}{cost(A') - cost(A^*) + 1}, i \in \{1, ..., d\}, \tag{2}$$

where γ is the reward discount factor. Suppose in Eq. 2 $cost(A') - cost(A)$ is constant, then the closer $cost(A')$ and

Algorithm 1: BandMaxSAT

Input: A (W)PMS instance $\mathcal{F}$, an initial complete assignment A of $\mathcal{F}$, cut-off time *cutoff*, BMS parameter k, reward delay steps d, reward discount factor γ, number of sampled arms $ArmNum$, exploration bias parameter λ

Output: A feasible assignment A of $\mathcal{F}$, or *no feasible assignment found*

```
A* := A, cost(A') := +∞, N := 0;
while running time < cutoff do
    if A is feasible & cost(A) < cost(A*) then
        A* := A;
    if D := {x|score(x) > 0} ≠ ∅ then
        v := a variable in D picked by BMS(k);
    else
        update_clause_weights();
        if ∃ falsified hard clauses then
            c := a random falsified hard clause;
        else
            update_estimated_value(A, A', A*, d, γ);
            N := N + 1, A' := A;
            c := PickArm(ArmNum, N, λ);
            t(c) := t(c) + 1
        v := the variable with the highest score in c;
    A := A with v flipped;
if A* is feasible then return A*;
else return no feasible assignment found;
```

$cost(A^*)$, the more rewards the action of pulling the last arm can yield, which is reasonable and intuitive.

III. FPS

The Farsighted Probabilistic Sampling (FPS) method [5] combines the look-ahead strategy with the probabilistic sampling strategy in an effective way. When a local optimum is reached, FPS first randomly samples sc_num (10 by default) falsified clauses, then tries to look-ahead from a random variable of each sampled clause, to check whether flipping a pair of variables can improve the current solution. If FPS fails to improve the current solution by flipping a pair of variables, it will select to flip the best among the best sampled single variable and the best sampled pair of variables.

With the help of the look-ahead strategy, FPS can improve the local optima for the single flipping mechanism, so as to find higher-quality solutions. While the probabilistic sampling strategy can help the algorithm improve its efficiency.

REFERENCES

[1] S. Cai, Z. Lei, "Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability," Artificial Intelligence, 287: 103354, 2020.

[2] Y. Chu, S. Cai, Z. Lei, X. He, "NuWLS-c: Solver Description," MaxSAT Evaluation 2022, 2022: 28.

[3] N. Alexander, "Anytime weighted maxsat withimproved polarity selection and bit-vector optimization," FMCAD 2019: 193–202.

[4] J. Zheng, K. He, J. Zhou, Y. Jin, C. M. Li, F. Manyà, "BandMaxSAT: A Local Search MaxSAT Solver with Multi-armed Bandit," IJCAI 2022: 1901-1907.

[5] J. Zheng, J. Zhou, K. He, "Farsighted Probabilistic Sampling based Local Search for (Weighted) Partial MaxSAT," AAAI 2023.

noSAT-MaxSATv2

Ole Lübke Sibylle Schupp
Institute for Software Systems
Hamburg University of Technology (TUHH)
Hamburg, Germany
{ole.luebke, schupp}@tuhh.de

***Abstract*—Using a SAT solver to solve MaxSAT is a well-established and efficient method. However, in resource-constrained computing environments, e.g., embedded systems, such algorithm designs can be hard to realize. With *noSAT-MaxSAT*, we explore alternatives that do not rely on an external, dedicated SAT solver, by executing an anytime local-search MaxSAT solving algorithm only on the hard clauses of the formula instead.**

In many real-world problems a significant portion of the hard clauses feature not more than two literals. Therefore, in *noSAT-MaxSATv2* we integrated a linear-time algorithm for 2-SAT as a preprocessing step to find a good initial assignment before proceeding to solve the remaining hard clauses with local search. Additionally, compared to the previous version of *noSAT-MaxSAT*, we updated the local-search algorithm from SATLike to NuWLS, the algorithm which won all incomplete tracks of the MaxSAT Evaluation 2022.

I. Introduction

In recent years the solvers in the incomplete/anytime track of the MaxSAT Evaluation (MSE) have converged to employ algorithms that rely on complete SAT solvers. A call to the SAT solver is often executed to obtain a valid initial assignment for the hard clauses in partial MaxSAT problems, or iteratively in linear search-based MaxSAT solvers [1, 2, 3, 4].

Our goal is to develop an efficient MaxSAT solver that is suitable for deployment in resource-constrained computing environments. Here, careful analysis of the resource requirements of the software, especially regarding runtime and memory, is usually required to verify the system against its specification. Each additional software module complicates this process, potentially to the point of infeasibility. Therefore, one of the key requirements for our solver is that it must not rely on a SAT solver.

noSAT-MaxSATv2 is based on the NuWLS local search algorithm, which won all incomplete tracks of the MSE 2022. It entered the competition as NuWLS-c and uses a dedicated SAT solver to solve the hard clauses in partial MaxSAT problems [5]. Instead of employing a SAT solver, *noSAT-MaxSATv2* first executes a linear-time 2-SAT algorithm on all hard clauses that have two or fewer literals. We refer to such clauses as *2-clauses*. For solving the remaining hard clauses, it uses NuWLS to solve the remaining hard clauses.

In the following, we elaborate on the employed 2-SAT solving algorithm and describe the requirements and architecture of *noSAT-MaxSAT*.

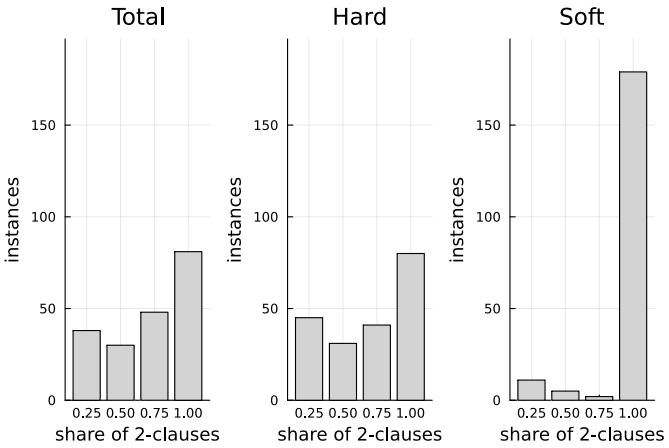

Fig. 1. Histograms showing the share of 2-clauses in the MSE 2022 benchmark instances

II. 2-SAT Solving

Many real-world MaxSAT problems have a large portion of 2-clauses. This is evident in the benchmark instances used in the MSE 2022, as shown by the histograms in Figure 1.

While MaxSAT is NP-complete even when restricted to only 2-clauses [6], 2-SAT can be solved in linear time [7, 8].

Because of its simplicity, we decided to use the linear-time algorithm by Even et al. [7] to solve hard 2-clauses. When at least 50% of the hard clauses are 2-clauses, we apply the algorithm. Otherwise, we preprocess using decimation [9] (as in the previous version of *noSAT-MaxSAT*). The resulting variable assignment is then used as the initial assignment for the local search algorithm that is employed to solve any remaining hard clauses.

III. Requirements & Architecture

noSAT-MaxSAT is developed under the following requirements, derived from common constraints found in programming embedded systems. The software

1) is programmed in C;
2) is self-contained, i.e., it has no external dependencies (other than the C standard library);
3) does not allocate memory dynamically;
4) does not use floating-point operations;
5) does not contain unbounded loops, i.e., it only contains `for` loops where the loop variable `i` is an integer that

is monotonically increased (decreased) until it reaches a certain maximum (minimum) `n`. However, `n` is not required to be a compile-time constant (yet it must be constant upon entering the loop), and the loop condition may be extended by conjunctively adding any number of boolean expressions (i.e., the loop may terminate before reaching `n`).

A solver which fulfills all of these requirements could not be expected to perform well in the MSE, because the sizes of the benchmarks are unknown beforehand, which conflicts with requirements 3) and 5). To circumvent this, *noSAT-MaxSAT* is split into a library that fulfills the requirements, and an application that uses the library but is not bound by the above-mentioned restrictions.

The interface of the library essentially consists of two functions: `nsms_solve` and `nsms_calcMemoryRequirements`. Given the number of variables and clauses of a formula, the latter function computes (an upper bound on) the number of bytes of memory the solver will need. The former function takes the formula, a pointer to a sufficiently-sized memory block, and the algorithm configuration. The application code takes care of parsing the input file and allocating memory to construct the formula that is then passed to the library functions.

In a constrained computing environment this splitting is not an option. For a particular application, however, it can be expected that the problem domain is more homogenous in such environments than in the MSE, so upper bounds on the number of variables and clauses are known a priori and memory can be pre-allocated statically.

References

[1] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. *MaxSAT Evaluation 2019 : Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2019. URL: https://helda.helsinki.fi/handle/10138/308068.

[2] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins. *MaxSAT Evaluation 2020 : Solver and Benchmark Descriptions*. University of Helsinki, Department of Computer Science, 2020. URL: https://helda.helsinki.fi/handle/10138/318451.

[3] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins. *MaxSAT Evaluation 2021 : Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2021. URL: https://helda.helsinki.fi/handle/10138/333649.

[4] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, Ruben Martins, and Andreas Niskanen. *MaxSAT Evaluation 2022 : Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2022. URL: https://helda.helsinki.fi/handle/10138/347396.

[5] Yi Chu, Shaowai Cai, Zhendong Lei, and Xiang He. "NuWLS-c: Solver Description". In: *MaxSAT Evaluation 2022 : Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2022, p. 28.

[6] M. R. Garey, D. S. Johnson, and L. Stockmeyer. "Some Simplified NP-complete Graph Problems". In: *Theoretical Computer Science* 1.3 (1976), pp. 237–267. DOI: 10.1016/0304-3975(76)90059-1.

[7] S. Even, A. Itai, and A. Shamir. "On the Complexity of Time Table and Multi-Commodity Flow Problems". In: *16th Annual Symposium on Foundations of Computer Science*. 1975, pp. 184–193. DOI: 10.1109/SFCS.1975.21.

[8] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. "A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas". In: *Information Processing Letters* 8.3 (1979). DOI: 10.1016/0020-0190(79)90002-4.

[9] Shaowei Cai, Chuan Luo, and Haochen Zhang. "From Decimation to Local Search and Back: A New Approach to MaxSAT". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. IJCAI-17. Melbourne, Australia, Aug. 2017, pp. 571–577. DOI: 10.24963/ijcai.2017/80.

TT-Open-WBO-Inc-23: an Anytime MaxSAT Solver Entering MSE'23

Alexander Nadel
Data and Decision Sciences, Technion, Haifa, Israel
Email: alexander.nadel@cs.tau.ac.il

***Abstract*—This document describes the solver TT-Open-WBO-Inc-23, submitted to the four incomplete tracks of MaxSAT Evaluation 2023. TT-Open-WBO-Inc-23 is the 2023 version of our solver TT-Open-WBO-Inc [9], itself based on Open-WBO-Inc [5]. The main innovation in TT-Open-WBO-Inc-23 is the integration of the local search component from NuWLS-c [4].**

I. Introduction

TT-Open-WBO-Inc [9] is our anytime MaxSAT solver, based on Open-WBO-Inc [5]. Mostly similarly to the previous year's version [11], TT-Open-WBO-Inc-23 combines the following algorithms:

1) NuWLS-c local search [4] for preprocessing (the only significant change from the previous year's version, which used SATLike [3]).
2) The unweighted component uses Mrs. Beaver [7], enhanced by the following two heuristics from Sect. 4.1 in [6]: global stopping condition for OBV-BS and size-based switching to complete part.
3) The weighted component uses BMO-based clustering [5].
4) The Polosat SAT-based local search algorithm [8] replaces the regular SAT invocations in both the unweighted and weighted components.

We adjusted some of the low-level parameters of the aforementioned algorithms to the benchmarks from the latest MaxSAT Evaluation.

We submitted two versions of TT-Open-WBO-Inc-23, the difference being the underlying SAT solver:

1) TT-Open-WBO-Inc-23(I): with IntelSAT [10].
2) TT-Open-WBO-Inc-23(G): with Glucose 4.1 [1].

References

[1] G. Audemard and L. Simon. On the Glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018.
[2] F. Bacchus, J. Berg, M. Järvisalo, R. Martins, and A. Niskanen, editors. *MaxSAT Evaluation 2022: Solver and Benchmark Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2022.
[3] S. Cai and Z. Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artif. Intell.*, 287:103354, 2020.
[4] Y. Chu, S. Cai, Z. Lei, and X. He. Nuwls-c: Solver description. In Bacchus et al. [2].
[5] S. Joshi, P. Kumar, S. Rao, and R. Martins. Open-wbo-inc: Approximation strategies for incomplete weighted maxsat. *J. Satisf. Boolean Model. Comput.*, 11(1):73–97, 2019.
[6] A. Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *FMCAD 2019*, pages 193–202.
[7] A. Nadel. Solving MaxSAT with bit-vector optimization. In *SAT 2018*, pages 54–72, 2018.
[8] A. Nadel. On optimizing a generic function in SAT. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 205–213. IEEE, 2020.
[9] A. Nadel. Polarity and variable selection heuristics for SAT-based anytime MaxSAT. *J. Satisf. Boolean Model. Comput.*, 12(1):17–22, 2020.
[10] A. Nadel. Introducing Intel(R) SAT solver. In K. S. Meel and O. Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
[11] A. Nadel. TT-Open-WBO-Inc-22: an Anytime MaxSAT Solver Entering MSE'22. In Bacchus et al. [2].

BENCHMARKS

MaxSAT Evaluation 2023: Solver and Benchmark Descriptions, volume B-2023-2 of Department of Computer Science Series of Publications B, University of Helsinki 2023.

MaxSAT Encodings for Inconsistency Measurement

Andreas Niskanen
University of Helsinki
Finland

Isabelle Kuhlmann
University of Hagen
Germany

Matthias Thimm
University of Hagen
Germany

Matti Järvisalo
University of Helsinki
Finland

***Abstract*—Inconsistency measurement aims at developing and analyzing methods for obtaining a quantitative understanding of the level of inconsistency in knowledge bases. Determining the level of inconsistency for propositional knowledge bases is a computationally challenging task. Here we briefly outline benchmarks generated via recently proposed MaxSAT encodings of four different inconsistency measures.**

***Index Terms*—inconsistency measurement**

I. Problem Setting

A knowledge base $\mathcal{K}$ is a finite set of propositional formulas. If there is an assignment satisfying every formula in $\mathcal{K}$, we say that $\mathcal{K}$ is consistent, and otherwise $\mathcal{K}$ is inconsistent. An *inconsistency measure* is a function $\mathcal{I}$ mapping a knowledge base $\mathcal{K}$ to a number $\mathcal{I}(\mathcal{K}) \geq 0$, for which $\mathcal{I}(\mathcal{K}) = 0$ iff $\mathcal{K}$ is consistent [1], [2]. We consider the following instantiations of this general notion.

- The *contension measure* $\mathcal{I}_{\mathrm{c}}$ [3] returns the smallest number of atoms in $\mathcal{K}$ which need to be set to *both* (true and false) in order to satisfy $\bigwedge \mathcal{K}$ according to Priest's three-valued logic [4].
- The *forgetting-based measure* $\mathcal{I}_{\mathrm{f}}$ [5] returns the smallest number of atom occurrences in $\mathcal{K}$ which need to be replaced by truth constants (*true* or *false*) to make $\mathcal{K}$ consistent.
- The *hit-distance measure* $\mathcal{I}_{\mathrm{hit}}$ [6] searches for an assignment satisfying the maximum number of formulas in $\mathcal{K}$, and returns the number of formulas not satisfied.
- The *sum-distance measure* $\mathcal{I}_{\mathrm{sum}}$ [6] searches for an assignment minimizing the sum of distances to the models of each formula in $\mathcal{K}$, and returns this sum.

II. MaxSAT Encodings

We briefly outline MaxSAT encodings of the inconsistency measures considered in this work [7]. Our encodings are based on first modifying the input knowledge base $\mathcal{K}$ into $\mathcal{K}'$, and then transforming $\mathcal{K}'$ into an equisatisfiable CNF formula via the Tseitin transformation. The CNF encoding thus includes auxiliary variables corresponding to subformulas in $\mathcal{K}'$.

The MaxSAT encodings for the contension and forgetting-based measures are similar. We replace the ith occurrence of atom x in $\mathcal{K}$ by a fresh variable x_i, which simulates replacing this occurrence by a truth constant. This modified knowledge base is transformed to CNF and included as hard clauses. For the contension measure, we include for each atom x a soft constraint $\bigwedge_{i=1}^{n_x}(x \leftrightarrow x_i)$. For the forgetting-based measure, we include for each atom x and its occurrence i a soft constraint $(x \leftrightarrow x_i)$. These soft constraints are encoded as unit-weighted soft clauses using auxiliary variables and hard clauses.

The MaxSAT encoding for the hit-distance measure simply includes $\mathcal{K}$ as hard clauses and every $\phi \in \mathcal{K}$ as a soft constraint. The latter is achieved by including the auxiliary variable corresponding to the formula as a soft clause with unit weight. Towards a MaxSAT encoding of the sum-distance measure, we introduce a copy of each $\phi \in \mathcal{K}$ where each atom x has been replaced by the variable x_ϕ. These copies are included as hard clauses. For each $\phi \in \mathcal{K}$ and each atom x, we include a soft constraint $(x \leftrightarrow x_\phi)$, which is encoded as a soft clause with unit weight.

III. Benchmarks

The benchmark instances were generated using the ARG dataset from previous work on inconsistency measurement [8]. This dataset consists of 326 knowledge bases with a standard SAT encoding of stable semantics in abstract argumentation [9], using argumentation frameworks from the ICCMA 2019 competition (http://argumentationcompetition.org/2019/), with an additional constraint enforcing that $20\,\%$ of arguments are included in the stable extension.

The instances are named according to the format `im-<measure>_<af>.tgf.pl.wcnf`, where `<measure>` is the inconsistency measure (contension, forgetting, hit, or sum) and `<af>` is the original argumentation framework. We selected instances with file size at most 100MB, and for which UWrMaxSat [10] (version 1.4) did not produce an optimal solution in 10 minutes in an earlier evaluation [7].

This work is supported by Academy of Finland (grants 322869, 347588, 356046) and Deutsche Forschungsgemeinschaft (grant 506604007.

References

[1] J. Grant and M. V. Martinez, Eds., *Measuring Inconsistency in Information*, ser. Studies in Logic. College Publications, 2018, vol. 73.
[2] M. Thimm and J. P. Wallner, "On the complexity of inconsistency measurement," *Artificial Intelligence*, vol. 275, pp. 411–456, 2019.
[3] J. Grant and A. Hunter, "Measuring consistency gain and information loss in stepwise inconsistency resolution," in *Proc. ECSQARU 2011*, ser. Lecture Notes in Computer Science, W. Liu, Ed., vol. 6717. Springer, 2011, pp. 362–373.
[4] G. Priest, "The logic of paradox," *Journal of Philosophical Logic*, vol. 8, no. 1, pp. 219–241, 1979.
[5] P. Besnard, "Forgetting-based inconsistency measure," in *Proc. SUM 2016*, ser. Lecture Notes in Computer Science, S. Schockaert and P. Senellart, Eds., vol. 9858. Springer, 2016, pp. 331–337.

[6] J. Grant and A. Hunter, "Analysing inconsistent information using distance-based measures," *International Journal of Approximate Reasoning*, vol. 89, pp. 3–26, 2017.
[7] A. Niskanen, I. Kuhlmann, M. Thimm, and M. Järvisalo, "MaxSAT-Based inconsistency measurement," in *Proc. ECAI 2023*. IOS Press, 2023.
[8] I. Kuhlmann, A. Gessler, V. Laszlo, and M. Thimm, "Comparison of SAT-based and ASP-based algorithms for inconsistency measurement," *arXiv*, p. 2304.14832, 2023, preprint.
[9] P. Besnard, S. Doutre, and A. Herzig, "Encoding argument graphs in logic," in *Proc. IPMU 2014*, ser. Communications in Computer and Information Science, A. Laurent, O. Strauss, B. Bouchon-Meunier, and R. R. Yager, Eds., vol. 443. Springer, 2014, pp. 345–354.
[10] M. Piotrów, "UWrMaxSat: Efficient solver for MaxSAT and pseudo-boolean problems," in *Proc. ICTAI 2020*. IEEE, 2020, pp. 132–136. [Online]. Available: https://doi.org/10.1109/ICTAI50040.2020.00031

MaxSAT Encodings for Judgment Aggregation

Ari Conati
HIIT, Department of Computer Science
University of Helsinki, Finland

Andreas Niskanen
HIIT, Department of Computer Science
University of Helsinki, Finland

Matti Järvisalo
HIIT, Department of Computer Science
University of Helsinki, Finland

I. Problem Overview

In preference aggregation, voters specify preferences over a set of candidates, and the task is to identify a suitable collective ranking which reflects the preferences of the group [1], [2]. Such a ranking is chosen according to a *voting rule* which, for a given voter profile, specifies a set of optimal preference rankings. Various voting rules have been specified in the literature, and under many proposed rules, computing a single optimal ranking is NP-hard.

Formally, for a set of m candidates $C = \{1, \ldots, m\}$ and a profile of n voters $V = (\succ_1, \ldots, \succ_n)$ where each $\succ_i$ is a preference relation over C, a voting rule R maps V to a set of collective rankings $R(V)$. In particular, we consider voting rules which define a set of strict total orders over the candidates (rather than, e.g., a set of "winners"). Each MaxSAT instance in this benchmark set corresponds to a specific profile V and voting rule R. The optimal solutions of each MaxSAT instance correspond exactly to $R(V)$.

II. MaxSAT Encodings

To obtain MaxSAT encodings for preference aggregation, we make use of recently proposed MaxSAT encodings for judgment aggregation as detailed in [3]. Judgment aggregation is a generic framework which considers the aggregation of individual judgments regarding the truth or falsehood of logical statements [4], [5], and captures various settings involving aggregation of information by social choice mechanisms, including preference aggregation [6]. A voter's preferences $\succ_i$ are represented as a truth assignment over propositional variables $p_{x>y}$ for each pair of candidates $x, y \in C$, with $p_{x>y}$ set to true iff $x \succ_i y$. That is, a voter specifies their preference between each pair of candidates. These truth assignments are called *judgment sets*.

For the preference aggregation instantiation, each encoding includes hard clauses which enforce that solutions correspond to strict total orders. The remaining clauses enforce constraints associated with the given judgment aggregation rule. In particular, the rules considered in the benchmark instances specify a cost function across candidate judgment sets. The encodings include soft clauses which enforce minimization of this cost. Each benchmark instance is generated according to one of the following rules [7] (using the corresponding encoding [3]).

This work is supported by Academy of Finland (grants 322869, 347588, 356046).

1) The *Kemeny* rule minimizes the average Hamming distance between the collective judgment set and the judgment sets of the voters.
2) The *Slater* rule minimizes the Hamming distance between the collective judgment set and the majoritarian judgment set.
3) The *MaxHamming* rule minimizes the maximum Hamming distance between the collective judgment set and the judgment sets of the voters.

III. Benchmark Data

The benchmark set consists of 90 MaxSAT instances selected from benchmarks in an earlier evaluation [3]. The file names follow the convention `ja-<rule>-preflib-<id>.wcnf` where `<rule>` is the judgment aggregation rule (`kemeny`, `slater`, or `maxham`), and `<id>` is the name of the voter profile from the PrefLib database [8]. Specifically, the MaxSAT instance is obtained by generating the encoding using the file `<id>.soc` in https://www.preflib.org/static/data/types/soc.zip, and an optimal solution to the MaxSAT instance maps back to an optimal collective preference ranking under the specified rule.

References

[1] F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, Eds., *Handbook of Computational Social Choice*. Cambridge University Press, 2016.

[2] D. Baumeister and J. Rothe, "Preference aggregation by voting," in *Economics and Computation: An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, ser. Springer texts in business and economics. Springer, 2016, pp. 197–325.

[3] A. Conati, A. Niskanen, and M. Järvisalo, "Sat-based judgment aggregation," in *AAMAS*. ACM, 2023, pp. 1412–1420.

[4] U. Endriss, "Judgment aggregation," in *Handbook of Computational Social Choice*. Cambridge University Press, 2016, pp. 399–426.

[5] D. Baumeister, G. Erdélyi, and J. Rothe, "Judgment aggregation," in *Economics and Computation, An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, ser. Springer texts in business and economics. Springer, 2016, pp. 361–391.

[6] J. Lang, G. Pigozzi, M. Slavkovik, L. van der Torre, and S. Vesic, "A partial taxonomy of judgment aggregation rules and their properties," *Soc. Choice Welf.*, vol. 48, no. 2, pp. 327–356, 2017.

[7] U. Endriss, R. de Haan, J. Lang, and M. Slavkovik, "The complexity landscape of outcome determination in judgment aggregation," *J. Artif. Intell. Res.*, vol. 69, pp. 687–731, 2020.

[8] N. Mattei and T. Walsh, "Preflib: A library for preferences http://www.preflib.org," in *ADT*, ser. LNCS, vol. 8176. Springer, 2013, pp. 259–270.

Synthesizing Pareto-Optimal Interpretations for Black-Box Models: A MaxSAT Encoding*

*Benchmarks for the Main Track of the MaxSat Evaluation 2023

1st Hazem Torfah
University of California at Berkeley
Berkeley, CA, USA
torfah@berkeley.edu

2nd Shetal Shah
Indian Institute of Technology Bombay
Mumbai, India
shetals@cse.iitb.ac.in

3rd Supratik Chakraborty
Indian Institute of Technology Bombay
Mumbai, India
supratik@cse.iitb.ac.in

4th S. Akshay
Indian Institute of Technology Bombay
Mumbai, India
akshayss@cse.iitb.ac.in

5th Sanjit A. Seshia
University of California at Berkeley
Berkeley, CA, USA
sseshia@berkeley.edu

***Abstract*—We present a set of benchmarks comprising MaxSAT encodings for the problem of synthesizing Pareto-optimal interpretations for black-box models. In Pareto-optimal interpretation synthesis, a set of Pareto-optimal interpretations is synthesized with respect to correctness and explainability measures. Users can choose the class of syntactic templates from which an interpretation should be synthesized, and the quantitative measures on correctness and explainability of an interpretation. This multi-objective optimization problem can be solved via a reduction to quantitative constraint solving, such as weighted maximum satisfiability. Our set of benchmarks includes encodings of the Pareto-optimal synthesis problem for three different black-box models, a decision module for predicting the performance of a perception module in an airplane, a bank loan predictor, and a predictor for the solvability of first-order formulas by a theorem prover.**

***Index Terms*—Explainability, Multi-objective optimization**

I. Introduction

Synthesizing a "good" human-understandable interpretation of a black-box ML component often requires striking the right balance between the correctness or accuracy of the interpretation (measured in terms of fidelity, misclassification rate of predictions, etc.) and its explainability or understandability (approximated by the size/depth of decision tree/list/diagram, number and nature of predicates used, etc.) [1], [2]. In most cases, the correctness and explainability measures are in direct conflict with each other. Thus, a simple interpretation that is easily understood by humans may disagree in its predictions with the output of a black-box ML component for many input instances, whereas an interpretation that correctly predicts the output for most input instances may be too large and unwieldy for human comprehension. This is not surprising since components like DNNs are often used to learn highly non-trivial functions for which simple models aren't available. Therefore, the synthesis of interpretations for black-box ML components is inherently a multi-objective optimization problem with conflicting objectives, and Pareto optimality is the best we can hope for when synthesizing such interpretations.

For a finite class of Interpretations, exploring the set of Pareto-optimal interpretations can be done by solving a series of weighted maximum satisfiability problems, each encoding the synthesis problem for certain regions of the solution space bounded by a threshold on the explainability measure [4]. The set of benchmarks presented in this paper comprises the MaxSAT encodings of instances of the Pareto-optimal interpretation synthesis problem for three different black-box models. In the next sections we formally define the Pareto-optimal interpretation synthesis problem, give a brief description of the MaxSAT formalization of the problem, and conclude with a description of the benchmarks.

This work was partially supported by NSF grants 1545126 (VeHICaL), 1646208 and 1837132, by the DARPA contracts FA8750-18-C-0101 (AA) and FA8750-20-C-0156 (SDCPS), by Berkeley Deep Drive, and by Toyota under the iCyPhy center.

II. Pareto-optimal Interpretation Synthesis

An interpretation is simply a syntactic structure, such as a decision tree, decision diagram, linear model, etc. We will fix a class of interpretations $\mathcal{E}$ over an input domain $\mathcal{I}$ and output domain $\mathcal{O}$. For an interpretation $E \in \mathcal{E}$, we define $f_E \in (\mathcal{I} \to \mathcal{O})$ to be the semantic function that is computed by E. Note that different interpretations may compute the same semantic function.

Every interpretation $E \in \mathcal{E}$ is associated with a pair of real-valued measures (c, e), where c is the correctness measure and e is the explainability measure of E. We define a partial order $\preceq$ on such pairs as: $(c, e) \preceq (c', e')$ iff $c \leq c'$ and $e \leq e'$. Given a set X of (c, e) pairs, we define $\max^{\preceq} X$ to be the set of $\preceq$-maximal pairs in X. An interpretation E with the pair of measures (c, e) is said to be *Pareto-optimal* if (c, e) is maximal over pairs of measures of all interpretations.

Definition 1 (Pareto-optimal interpretation synthesis): Let $\mathcal{E}$ be a syntactic class of interpretations over inputs $\mathcal{I}$ and outputs $\mathcal{O}$. Further, let $\mathcal{S} \subseteq \mathcal{I} \times \mathcal{O}$ be a set of samples, $\Delta_{\mathcal{C}} \colon (\mathcal{I} \to \mathcal{O}) \times 2^{(\mathcal{I} \times \mathcal{O})} \to \mathbb{R}^{\geq 0}$ be a correctness measure, and $\Delta_{\mathcal{E}} \colon \mathcal{E} \to \mathbb{R}^{\geq 0}$ an explainability measure. The Pareto-optimal interpretation synthesis problem $\langle \mathcal{E}, \mathcal{S}, \Delta_{\mathcal{C}}, \Delta_{\mathcal{E}} \rangle$ is the multi-objective problem of finding a Pareto-optimal interpretation

$$E \in \arg \operatorname*{\overrightarrow{max}}_{E' \in \mathcal{E}} \left(\Delta_{\mathcal{C}}(f_{E'}, \mathcal{S}), \Delta_{\mathcal{E}}(E')\right)$$

.

We interpret $\Delta_{\mathcal{C}}(f_E, \mathcal{S})$ as a measure of "closeness" between an interpretation and the semantic constraints defined by $\mathcal{S}$, in which case, the optimization problem is one of maximization. An example of such a measure is the prediction accuracy. Similarly, for $\Delta_{\mathcal{E}}(\cdot)$, we choose to define it as a reward function that we want to maximize. For each $\preceq$-maximal pair of measures, there can be multiple corresponding interpretations realizing the measures. We don't distinguish between them for purposes of this paper. Our goal can therefore be stated as one of finding a minimal representative set of interpretations for all Pareto-optimal pairs of measures.

III. Weighted MaxSAT Encoding

We start by giving a high-level description of how to encode an instance of Pareto-optimal interpretation synthesis as a weighted maximum satisfiability problem.

For an instance $\langle \mathcal{E}, \mathcal{S}, \Delta_{\mathcal{C}}, \Delta_{\mathcal{E}} \rangle$ of the Pareto-optimal interpretation synthesis problem, we define its encoding as a conjunction of four formulae. Specifically, $\phi_{\langle \mathcal{E}, \mathcal{S}, \Delta_{\mathcal{C}}, \Delta_{\mathcal{E}} \rangle} = \phi_{\mathcal{E}} \wedge \phi_{\mathcal{S}} \wedge \phi_{\Delta_{\mathcal{C}}} \wedge \phi_{\Delta_{\mathcal{E}}}$ where, (i) $\phi_{\mathcal{E}}$ encodes the syntactic restrictions fixed by the class of interpretations, e.g., bounded multi-valued decision diagrams with the permitted predicates (features and branchings) and labels; (ii) $\phi_{\mathcal{S}}$ encodes the semantic constraints, i.e., the relation between the samples in $\mathcal{S}$ and an interpretation satisfying $\phi_{\mathcal{E}}$; (iii) $\phi_{\Delta_{\mathcal{C}}}$ encodes the correctness measure, e.g., in case of prediction accuracy it encodes whether an interpretation agrees on a sample; and finally (iv) $\phi_{\Delta_{\mathcal{E}}}$ defines constraints that encode certain structural aspects of an interpretation, e.g., the predicates chosen and whether a node was used. We discuss some details of these formulas below. The full encoding is presented in [3].

a) Encoding of the interpretation class ($\phi_{\mathcal{E}}$): We discuss the encoding for our interpretation class of bounded multi-valued decision diagrams over inputs $\mathcal{I}$ and outputs $\mathcal{O}$. These diagrams are restricted by a finite set of decision predicates, denoted by P. Let L be a set of output labels, e.g., the classes of a classifier . An *interpretation* $E \in \mathcal{E}$ is a multi-valued decision diagram over a finite set of nodes $\mathcal{N}$, where each internal node corresponds to a decision predicate $p \in P$ and each leaf to an output label $\ell \in L$. Outgoing transitions of a node are labelled according to the branchings of the predicate corresponding to the node. Furthermore, the same predicate may appear on different nodes in the decision diagram, but not more than once along a path. For a given P, L, and a bound n on the number of nodes $\mathcal{N}$ in the decision diagram, the formula $\phi_{\mathcal{E}}$ encodes an acyclic decision diagram of at most n-nodes over a set P of predicates, with leaves labeled by elements of L.

b) Encoding of the samples: The formula $\phi_{\mathcal{S}}$ encodes the relation between the samples and the interpretation $\phi_{\mathcal{E}}$. It uses an auxiliary variable m_s for each sample $s = (i, o)$ in the set $\mathcal{S}$. Logically, m_s is set to true iff the interpretation given by a satisfying assignment of $\phi_{\mathcal{E}}$ produces the output label o when fed the input i. For decision diagrams, this is encoded by symbolically matching the input i to a decision path in the diagram, and by comparing the value of o with that of the label reached at the end of the decision path. Note that the number of these auxiliary variables grows linearly with the size of the sample set.

c) Encoding the correctness measure ($\phi_{\Delta_{\mathcal{C}}}$): To encode $\Delta_{\mathcal{C}}$, we add a unit soft clause m_s for each sample $s \in \mathcal{S}$. By assigning appropriate weights to these unit clauses and by maximizing the sum of weights of satisfied clauses, we obtain an interpretation that maximizes $\Delta_{\mathcal{C}}$ with respect to the sample set $\mathcal{S}$. E.g., if $\Delta_{\mathcal{C}}$ represents the prediction accuracy, then assigning a weight of 1 to each unit clause m_s gives us an interpretation that agrees on a maximal number of samples in $\mathcal{S}$. If the user is interested in interpretations that agree on certain types of samples, then higher weights should be given to these samples.

d) Encoding the explainability measure ($\phi_{\Delta_{\mathcal{E}}}$): To encode $\Delta_{\mathcal{E}}$, we add a unit clause u_γ for each syntactic structure γ of an interpretation in $\mathcal{E}$ and give it a weight according to how important γ is. For example, in the case of decision diagrams, using some predicates may be more favorable than others. To encode this, we add unit clauses $\lambda_{i,p}$ that are set to true iff predicate p is used in node i, and assign higher weights for clauses representing favorable predicates. To further reward the synthesis of decision diagrams with fewer nodes, we can also add unit soft clauses u_i for each node i that is set to true iff node i is not reachable from the root node in an interpretation satisfying $\phi_{\mathcal{E}}$, and give them positive weights. In this case, by maximizing the satisfaction of these clauses, we reward the synthesis of small decision diagrams.

e) Encoding the thresholds on the explainability measure: The exploration algorithm [4], first starts by synthesizing a Pareto-optimal interpretation with a maximized sum of correctness and explainability. It then iteratively, continues the search for Pareto-optimal interpretations in regions restricted by thresholds on the explainability measure [1]. To restrict the space of interpretations to ones that have an explainability measure between two thresholds $\delta^l_{\mathcal{E}}$ and $\delta^u_{\mathcal{E}}$ we add an additional constraint that sums up the weights of satisfied soft clauses $\lambda_{i,p}$ and u_i and compares the result to $\delta^l_{\mathcal{E}}$ and $\delta^u_{\mathcal{E}}$. This is done by adding an encoding for binary representations of the weights and encoding a binary adder that sums them up.

In our weighted MaxSAT formulation, we require that all clauses resulting from a Tseitin encoding (i.e., a transformation into CNF) of the formula $\phi_{\langle \mathcal{E}, \mathcal{S}, \Delta_{\mathcal{C}}, \Delta_{\mathcal{E}} \rangle}$, except the unit soft clauses mentioned above, be hard clauses.

[1]For more details on why it suffices to restrict the explainability measure, we refer the reader to [4]

IV. BENCHMARKS

Our benchmarks comprise of the weighted MaxSAT encodings created while solving the Pareto-optimal interpretation synthesis problem for three different black-box models: a decision module monitoring the performance of a perception module in an airplane, a deep neural network for predicting bank loans, and a neural network for predicting the solvability of first-order formulas by a theorem prover.

For each model, we solved the Pareto-optimal interpretation synthesis problem for different sizes of the decision diagram and different numbers of samples (determined based on the theory of probably approximately correct learning in terms of the size of the class of decision diagrams, a confidence measure, and an error margin). The MaxSAT encodings created for each model are sorted into different folders named after the model. Each of these folders includes several runs of the synthesis problem sorted into folders named after the number of samples, the number of predicates, and the size of diagrams used. Each of these folders has a config file stating the properties of the decision diagrams and defining the explainability weights. The encodings are available in another folder and are named after the explainability region for which the synthesis was solved, and a lower bound on the correctness. Specifically, an encoding `ms_el_eu_c` defines the encoding for the region bounded by a lower bound `el` and upper bound `eu` on the explainability measure. The value `c` defines a lower bound on the correctness and is relevant for the exploration algorithm but not for the encoding.

In the following, we give a few more details on the different models and parameters of the synthesis instances:

a) Decision module for predicting the performance of a perception module in an airplane (AP): The decision module is an implementation of a decision tree that was trained on data collected from 200 simulations, using the XPlane[2] simulator [6]. The decision module predicts, based on the time of day, the cloud types, and the initial positioning of an airplane on a runway, whether a perception module used by the plane can be trusted to keep the plane on the centerline of a Taxi way.

The encodings define decision diagrams that accept three inputs (t, c, p) which determine the time of day, the cloud type, and the initial position of the plane on the runway. The diagrams have two outputs, namely, *alert* and *no alert*. The encodings are those for five configurations of the problem, (5,3,277), (5,5,325), (7,3,100), (7,3,333), and (7,3,555), the first component being the bound on the size of the decision diagram, the second determining the number of predicates used, and the third defining the number of randomly generated samples over which an interpretation is synthesized.

b) Bank loan predictor (BL).: The bank loan predictor is a deep neural network that was trained on synthetic data that we created. The network was trained on the following features: age, monthly income, credit score, and the number of dependents. The training set included 100000 entries chosen such that the majority of people with age between 18 to 29 years, and those with age between 30 and 49 years but with income less than $6000, were denied the loan. The values of the remaining features were chosen randomly.

The encodings define decision diagrams that accept three inputs (a, i, c, d) which define the age, income, credit score, and the number of dependents. The diagrams had two outputs, namely, *approve* and *deny*. The encodings are sorted according to three configurations of the problem, (7,4,277), (7,4,365), and (7,4,608), the first component being the bound on the size of the decision diagram, the second determining the number of predicates used, and the third defining the number of randomly generated samples over which an interpretation is synthesized.

c) Theorem prover (TP): The neural network predicts the solvability of first-order formulas by a theorem prover with respect to the percentage of unit clauses and the average clause length in a formula. The data used to train the neural network can be found on the UCI machine learning repository under the following link https://archive.ics.uci.edu/ml/datasets/First-order+theorem+proving. The network was trained on the following features: F10, is a feature determining the average clause length in the formula, and F1, is the percentage of unit clauses in the formula. For more details on the attributes, we refer the reader to [5]. The authors of [5] included data for five different heuristics H1-H5. We used the data for H1, thus predicting the solvability for H1.

The encodings define decision diagrams that accept the two inputs (F1, F10). The diagrams had two outputs, namely, *solvable* and *not solvable*. The encodings are sorted according to three configurations of the problem, (5,6,338), (7,6,422), and (7,6,703), the first component being the bound on the size of the decision diagram, the second determining the number of predicates used, and the third defining the number of randomly generated samples over which an interpretation is synthesized.

[2]x-plane.org

REFERENCES

[1] A. Adadi and M. Berrada. "Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI)". IEEE Access, 2018.

[2] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, Franco, F. Giannotti, and D. Pedreschi. "A Survey of Methods for Explaining Black Box Models". ACM Comput. Surv., 2018.

[3] H. Torfah, S. Shah, S. Chakraborty, S. Akshay and S. A. Seshia, "Synthesizing Pareto-Optimal Interpretations for Black-Box Models", CoRR arXiv, abs/2108.07307, http://arxiv.org/abs/2108.07307, 2021.

[4] H. Torfah, S. Shah, S. Chakraborty, S. Akshay and S. A. Seshia, "Synthesizing Pareto-Optimal Interpretations for Black-Box Models". Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, October 19-22, https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_24, 2021.

[5] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. "Learning for First-Order Theorem Proving - Learning to Select a Good Heuristic". J. Autom. Reasoning, 53(2), https://archive.ics.uci.edu/ml/datasets/First-order+theorem+proving, 2014.

[6] H. Torfah, S Junges, D. J. Fremont, and Sanjit A. Seshia "Formal Analysis of AI-Based Autonomy: From Modeling to Runtime Assurance".

Description of Benchmarks on Optimizing Binary Decision Diagrams

Hao Hu, Marie-José Huguet, Mohamed Siala
LAAS-CNRS, Université de Toulouse, INSA, Toulouse, France
{hhu, huguet, siala}@laas.fr

I. Introduction

By providing compact representations for Boolean functions, Binary Decision Diagrams (BDDs) are interpretable in the context of binary classification. Compared to Decision Trees, Binary Decision Diagrams could avoid the general *replication* problem and *fragementation* problem [6], [8], which indicates that a Binary Decision Diagram is *smaller* in size than the corresponding Decision Tree generally in practice.

Recently, several *exact methods* have been proposed to provide optimal decision trees or optimal binary decision diagrams via Boolean Satisfiability (SAT) [1], [7] or its variant MaxSAT [4], [5]. Those exact methods have gained increasing interest from the MaxSAT community. Especially, benchmarks from the MaxSAT-based approach to learn optimal decision trees [5] have been selected by *MaxSAT Evaluation 2021* in all tracks [2], [3]. Although the use of incomplete solvers could find high-quality solutions within a limited time, this MaxSAT-based approach suffers from general scalability issues, which somehow motivates the need of exact methods for optimal interpretable machine learning models with lighter encodings.

Therefore, we propose a new MaxSAT-based exact method for optimal binary decision diagram in [4]. The objective is to optimize binary decision diagrams of limited depths that minimize classification error. Briefly, it is realized in two steps: an SAT approach to learning optimal binary decision diagrams with the *smallest* depths with perfect empirical accuracy, and a lifted MaxSAT approach for the objective. We assume that all binary decision diagrams are *ordered* and *reduced*[1].

II. SAT Approach of Optimizing Binary Decision Diagrams

A. Problem Definition

The proposed SAT encoding aims to solve the following decision problem:

- $P(\mathcal{E}, H)$: *Given a set of examples $\mathcal{E}$, is there a BDD of depth H that classifies correctly all examples in $\mathcal{E}$?*

Notice that the SAT approach uses a linear search with an initial depth to find the optimal binary decision diagram with the smallest depth classifying all examples correctly, by progressively increasing (respectively, decreasing) the depth if the answer to the corresponding decision problem is *False* (respectively, *True*), and terminates when the answer is *True* (respectively, *False*).

[1]The restriction *ordered* indicates the existence of a global sequence of features in BDD, the restriction *reduced* indicates the elimination of all isomorphic sub-graphs. Detailed definitions are described in [4].

B. SAT Encoding

We can generate an ordered reduced binary decision diagram of depth H by the combination of a sequence of Boolean variables of size $H : [x_1, \ldots, x_H]$, and a truth table of length 2^H associated to a Boolean function. To solve the classification problem $P(\mathcal{E}, H)$, one needs to find a sequence of binary features of size H that maps one-to-one the sequence of Boolean variables, and a truth table associated to a Boolean function classifying all examples correctly. The sequence of binary features found is noted as *feature ordering*. Therefore, the SAT encoding contains two parts of constraints:

- **Part 1**: Constraints for selecting features of the dataset into the feature ordering of size H.
- **Part 2**: Constraints for generating a truth table that classifies all examples in $\mathcal{E}$ correctly with the selected feature ordering.

We propose two SAT encodings, the difference is the constraints of **Part 2**. Here, we consider the second model, called the *improved model* in [4], as it benefits from a clear theoretical advantage in terms of encoding size. In brief, the core idea is to let every positive (respectively, negative) example lead to a positive (respectively, negative) value of the truth table, with the selected feature ordering. For instance, there are 2^H constraints to classify a positive example e_q correctly:

$$\begin{aligned} &\neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge \neg d_H^q \rightarrow c_1 \\ &\neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge d_H^q \rightarrow c_2 \\ &\qquad \ldots \\ &d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge \neg d_H^q \rightarrow c_{2^H-1} \\ &d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge d_H^q \rightarrow c_{2^H} \end{aligned} \tag{1}$$

where Boolean variable c_j is 1 iff the j-th value of the truth table is 1, and Boolean variable d_i^q is 1 iff for example e_q the value of the i-th feature selected in the feature ordering is 1. The antecedent of each constraint corresponds to an identical decision path from the root of BDD to a value of the truth table. To classify a negative example correctly, there are also 2^H similar constraints like Constraints 1, just change the consequent of each constraint as the negation.

III. MaxSAT Approach of Optimizing Binary Decision Diagrams

A. Problem Definition

The proposed MaxSAT approach aims to solve the following optimization problem:

- $P^*(\mathcal{E}, H)$*: Given a set of examples* $\mathcal{E}$*, find a BDD of depth* H *that maximises the number of examples in* $\mathcal{E}$ *that are correctly classified.*

Compared to the SAT approach requiring perfect empirical accuracy, the MaxSAT approach could reduce the risk of overfitting.

B. MaxSAT Encoding

The MaxSAT encoding is obtained from the SAT encoding following a simple technique. The idea is to keep the *structural* constraints (**Part 1**) as *hard clauses*, and the *classification* constraints (*Constraints 1* of **Part 2**) as *soft clauses*.

As explained before, for any example e_q, the antecedent of each constraint of classification corresponds to an identical decision path from the root of BDD to a value of truth table. Considering there is only a unique decision path for e_q, the number of satisfied soft clauses only corresponds to the two possible values:

- $\mathbf{2^H}$: example e_q is correctly classified.
- $\mathbf{2^H - 1}$: example e_q is wrongly classified.

Therefore, considering $\mathcal{E}$ contains M examples, when solving the proposed MaxSAT encoding, the number of *satisfied* soft clauses is in the interval of $[M \times (2^H - 1), M \times 2^H]$. Equivalently speaking, the number of *unsatisfied* soft clauses of the solution provided by a MaxSAT solver is equal to the number of examples *wrongly* classified.

IV. Benchmark Instances

The zip archive contains 90 WCNF files in 15 folders, which correspond to the names of machine learning datasets. The datasets used to generate WCNF are from CP4IM[2]. More precisely, they are binarized with the *one-hot-encoding*. Table I shows the detailed information of those datasets, where M indicates the number of examples, K_{orig} indicates the number of features, K indicates the number of features after the binarized, and pos indicates the percentage of positive examples.

In each folder, there are 8 WCNF files that correspond to optimizing binary decision diagrams with 8 different depths from 2 to 9. These benchmarks are all ***unweighted***. Considering the encoding size, all WCNF files corresponding to depth 2 are recommended for ***unweighted complete track***, and the rest are suitable for ***unweighted incomplete track***.

The name of each WCNF file follows the format: wcnf_incomplete_improved_*ratio_seed_depth*.WCNF.

- *ratio*: The sample ratio used when generating a training set using the hold-out method. By default, the whole dataset is considered as the training set.

[2]https://dtai.cs.kuleuven.be/CP4IM/datasets/

Dataset	M	K_{orig}	K	pos
anneal	812	42	89	0.77
audiology	216	67	146	0.26
australian	653	51	124	0.55
cancer	683	9	89	0.35
car	1728	6	21	0.30
cleveland	296	45	95	0.54
hypothyroid	3247	43	86	0.91
kr-vs-kp	3196	36	73	0.52
lymph	148	27	68	0.55
mushroom	8124	21	112	0.52
tumor	336	15	31	0.24
soybean	630	16	50	0.15
splice-1	3190	60	287	0.52
tic-tac-toe	958	9	27	0.65
vote	435	16	48	0.61

TABLE I
DETAILED INFORMATION OF DATASETS FROM CP4IM.

- *seed*: The seed used to make the stratified sampling, 2019 is used as default.
- *depth*: The depth of the optimal binary decision diagram aimed to learn.

References

[1] F. Avellaneda, "Efficient inference of optimal decision trees," in *Proceedings of the Thirty-Fourth Conference on Artificial Intelligence (AAAI)*, New York, USA, February 2020.

[2] F. Bacchus, J. Berg, M. Järvisalo, and R. Martins, Eds., *MaxSAT Evaluation 2021: Solver and Benchmark Descriptions*, ser. Department of Computer Science Report Series B. Finland: Department of Computer Science, University of Helsinki, 2021.

[3] H. Hu, E. Hebrard, M.-J. Huguet, and M. Siala, "Description of benchmarks on learning optimal decision trees and boosted trees," *MaxSAT Evaluation 2021*, p. 39, 2021.

[4] H. Hu, M. Huguet, and M. Siala, "Optimizing binary decision diagrams with maxsat for classification," in *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*. AAAI Press, 2022, pp. 3767–3775. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/20291

[5] H. Hu, M. Siala, E. Hebrard, and M. Huguet, "Learning optimal decision trees with maxsat and its integration in adaboost," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, C. Bessiere, Ed. ijcai.org, 2020, pp. 1170–1176. [Online]. Available: https://doi.org/10.24963/ijcai.2020/163

[6] R. Kohavi, "Bottom-up induction of oblivious read-once decision graphs," in *Machine Learning: ECML-94, European Conference on Machine Learning, Catania, Italy, April 6-8, 1994, Proceedings*, ser. Lecture Notes in Computer Science, F. Bergadano and L. D. Raedt, Eds., vol. 784. Springer, 1994, pp. 154–169. [Online]. Available: https://doi.org/10.1007/3-540-57868-4_56

[7] N. Narodytska, A. Ignatiev, F. Pereira, and J. Marques-Silva, "Learning optimal decision trees with SAT," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence - IJCAI*, July 13-19, 2018, pp. 1362–1368.

[8] J. Oliver, *Decision graphs: an extension of decision trees*. Citeseer, 1992.

Solver Index

Benchmark Index

Author Index