!!! DRAFT !!! DRAFT !!!

MaxSAT Evaluation 2022

Solver and Benchmark Descriptions

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

University of Helsinki
Department of Computer Science
Series of Publications B
Report B-2022-2

Helsinki 2022

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 2022 brought on the 16th edition of the MaxSAT Evaluations, organized as a satellite event of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022). 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 2022 evaluation consisted of a total of five tracks: two for complete solvers (one for solvers focusing on unweighted and one for solvers focusing on weighted MaxSAT instances), two for incomplete MaxSAT solvers (using two short per-instance time limits, 60 and 300 seconds, differentiating from the per-instance time limit of 1 hour imposed in the main complete tracks), as well as the incremental track as a new development for 2022. As in 2017-2021, 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 2022 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 2022 evaluation smoothly.

Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, Ruben Martins, & Andreas Niskanen
MaxSAT Evaluation 2022 Organizers

Contents

Benchmark Descriptions

SOLVERS

CASHWMaxSAT-CorePlus: Solver Description

Zhendong Lei[1,2], Yiyuan Wang[3,4], Shiwei Pan[3,4], Shaowei Cai[1,2,*], Minghao Yin[3,4]
[1]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]School of Computer Science and Technology, University of Chinese Academy of Sciences, China
[3]School of Computer Science and Information Technology, Northeast Normal University, China
[4]Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China
*corrsponding author
leizhendong3@huawei.com, yiyuanwangjlu@126.com, caisw@ios.ac.cn,
{pansw779, 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 2022.**

I. Introduction

We developed a new complete MaxSAT solver called CASHWMaxSAT-CorePlus based on UWrMaxSat [1] and CASHWMaxSAT [2]. In addition, CASHWMaxSAT-CorePlus used an unsatisfiable-core-based OLL procedure [3]–[6]. In this work, we propose two novel ideas to improve UWrMaxSat and CASHWMaxSAT, resulting in a new solver CASHWMaxSAT-CorePlus.

- First, when the SAT solver returns the "l_Undef" state, we mark all the relax variables in the current assumption as delayed relax variables and then we put them into the delay assumption.
- Second, when the SAT solver returns the "unsat" state, there exist more than one unsatisfiable-cores. Several unsatisfiable-cores may be obtained and we minimize these cores. Afterwards, we choose the best one (i.e., one with the minimum size) from these minimal unsatisfiable-cores. Additional, we don't minimize all obtained unsatisfiable-cores by using the hash structure to decrease the time complexity of the above process.

II. Future work

First, we could use a simplified version of MaxSAT local search solvers such as FPS [7] 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] 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.
[3] 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.
[4] 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.
[5] 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.
[6] 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.
[7] J. Zheng, J. Zhou, and K. He, "Farsighted probabilistic sampling based local search for (weighted) partial maxsat," *arXiv preprint arXiv:2108.09988*, 2021.

CASHWMaxSAT-Plus: Solver Description

Yiyuan Wang[1,2], Shiwei Pan[1,2], Zhendong Lei[3,4], Shaowei Cai[3,4,*], Minghao Yin[1,2], Shuli Hu[1,2] and Yupeng Zhou[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, pansw779@nenu.edu.cn, leizhendong3@huawei.com, caisw@ios.ac.cn,
{ymh, husl903, zhouyp605}@nenu.edu.cn

***Abstract*—This document describes the MaxSAT solver CASHWMaxSAT-Plus, submitted to the complete tracks (include unweighted and weighted track) of MaxSAT Evaluation 2022.**

I. Introduction

We developed a new complete MaxSAT solver called CASHWMaxSAT-Plus based on UWrMaxSat [1] and CASHWMaxSAT [2]. In addition, CASHWMaxSAT-Plus used an unsatisfiable-core-based OLL procedure [3]–[6]. In this work, we propose one novel idea to improve UWrMaxSat and CASHWMaxSAT, resulting in a new solver CASHWMaxSAT-Plus.

- When the SAT solver returns the "l_Undef" state, we mark all the relax variables in the current assumption as delayed relax variables and then we put them into the delay assumption.

II. Future work

First, we could use a simplified version of MaxSAT local search solvers such as FPS [7] 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] 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.
[3] 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.
[4] 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.
[5] 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.
[6] 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.
[7] J. Zheng, J. Zhou, and K. He, "Farsighted probabilistic sampling based local search for (weighted) partial maxsat," *arXiv preprint arXiv:2108.09988*, 2021.

CGSS in the 2022 MaxSAT Evaluation

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

I. Introduction

We overview the CGSS solver as it participated in the 2022 Evaluation. In short, CGSS implements the core-guided OLL algorithm for MaxSAT, extended with weight aware core extraction, structure sharing and selective addition of equivalences as described in [7] and [3]. Additionally, the solver makes use of stratification, hardening and the so-called core-exhaustion and intrinsic atmost1 techniques described in [6].

The solver is implemented in python, on top of PySAT [5] and the RC2 solver [6]. The authors would like to thank the developers of RC2 for their work. If you use CGSS in your research, we kindly ask you cite [7].

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 a *blocking variable* (of the instance $\mathcal{F}$) if $(\neg b) \in F_s$. As assigning a blocking variable to 1 corresponds to falsifying a soft clause, we will in the rest of the text view cores as sets of blocking variables and extend the weight function to blocking variables via $w(b) = w((\neg b))$.

III. Main Features

We overview the main features of CGSS. For a more detailed description, we refer the reader to [7].

OLL: When solving an instance $\mathcal{F}$ the (basic form of the) OLL algorithm [8], [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 blocking 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 CGSS solver uses the so called *totalizer* [2] CNF encoding of cardinality constraints. The totalizer encoding can be viewed as a tree structure similar to those 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 $S \subset \kappa$ as leaves defines $|S| = m$ new variables $b_0^S, \ldots, b_{m-1}^S$ defined with clauses equivalent to $\left(\sum_{b \in S} b \geq k+1\right) \rightarrow b_k^S$. Specificallym the root of the full tree then defines a set $b_0^\kappa, \ldots b_{n-1}^\kappa$ that count the number of variables of κ set to true by assignments satisfying the totalizer.

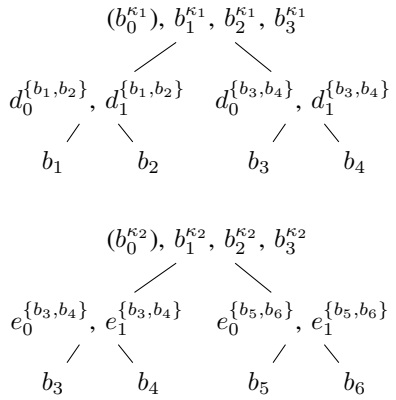

Fig. 1: The structure of totalizers built when relaxing cores $\kappa_1 = \{b_1, b_2, b_3, b_4\}$ (above) and $\kappa_2 = \{b_3, b_4, b_5, b_6\}$ (below).

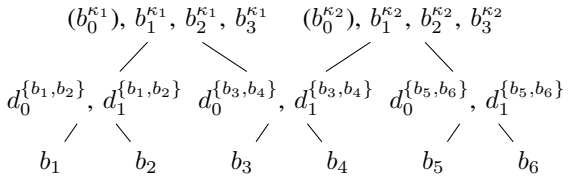

Fig. 2: The structure of totalizers when relaxing the cores κ_1 and κ_2 with structure sharing.

Weight aware core extraction (WCE) [4] 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 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 having b_3 and b_4 as leaves. The root of each of these subtrees define separate variables ($d_i^{\{b_3,b_4\}}$ for the top tree, $e_i^{\{b_3,b_4\}}$ for 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 equivalent in all solutions 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, the CGSS solver 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 seeks to identify counting variables of the (shared) totalizers to which it would be useful to add equivalence constraints that facilitate more propagation.

More precisely, 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 do perform more propagation, the large number of clauses required in order to do so for every internal node might instead result in overall deterioration of performance.

In order to balance the potential benefits and overhead (in the form of extra clauses) of adding both sides of the equivalence defining the variables in a totalizer, CGSS 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.

Bounds: The use of WCE and stratification leads to CGSS obtaining intermediate solutions to the instance during search. The cost of any such solution is an upper bound on the optimal cost of the instance. CGSS stores these solutions, effectively turning it into an any-time MaxSAT solver. At the same time, the cores extracted during search can be used to compute a lower bound on the optimal cost by summing the minimum weight of variables appearing in each extracted core. The use of both an upper and a lower bound can allow the solver to terminate as soon as the bounds match, sometimes even before reformulating all of the extracted cores.

IV. Compilation and Usage

CGSS is implemented on top of RC2 in the PySAT framework [5], [6] in a mixture of Python and C++ and can be found at https://bitbucket.org/coreo-group/cgss/src/master/ or the Evaluation website. Installing and running CGSS resembles installing and running RC2, please follow the readme of the repository for more details. The readme also details the command line parameters, the evaluation version of CGSS is invoked by running:

```
python rc2.py -lamWnP [instance.wcnf.gz]
```

from the examples subfolder of the repository base folder.

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] 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.

[3] 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.

[4] 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.

[5] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, pp. 428–437. [Online]. Available: https://doi.org/10.1007/978-3-319-94144-8_26

[6] ——, "RC2: An efficient MaxSAT solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 11, no. 1, pp. 53–64, 2019.

[7] 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.

[8] 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.

Weighted version of EvalMaxSAT 2022

Florent Avellaneda*, Carl-Elliott Bilodeau-Savaria†, Lancelot Normand‡
Computer Science Departement, Université du Québec à Montréal
QC, Canada
Email: *avellaneda.florent@uqam.ca, †bilodeau-savaria.carl-elliott@courrier.uqam.ca, ‡normand.lancelot@courrier.uqam.ca,

I. Introduction

EvalMaxSAT[1] is a MaxSAT solver written in modern C++ language mainly using the Standard Template Library (STL). The solver is built on top of the SAT solver 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]. This new version of the solver includes support for weighted formulas.

II. Description

The main strategy of the solver to solve weighted formula is to rapidly find non-optimal solutions that allow some soft variables to be transformed to hard variables. Specifically, if an assignment is obtained such that the sum of the weights of the unsatisfied soft variables is smaller than the weight of a soft variable v, then we can deduce that this variable v must be satisfied, and can therefore be considered a hard variable.

To find non-optimal solutions, there is one existing approach, known in the literature as the stratification strategy [6], involves considering only the variables with a weight higher than a certain threshold as soft, then reducing the threshold until all the soft variables are considered.

In this new version of EvalMaxSAT, a second strategy is added in addition to the stratification strategy. The second strategy does not immediately add new constraints when a new core is considered, but accumulates constraints until the formula becomes satisfiable. The accumulated constraints are considered only when the formula becomes satisfiable and a search for soft variables to transform to hard will be performed

Algorithm 1 presents a general view of how the solver functions. Note, that the function ChooseNextMinimumWeight represents a heuristic used to select a threshold value necessary to select a subset of soft variables. The heuristic implemented in the tool consists of reducing the threshold by a minimum step initially, then increasing this step when the computation time of the second loop (line 4) increases.

Algorithm 1

Input: A formula φ

1: **while** $true$ **do**
2: $w_{min} \leftarrow ChooseNextMinimumWeight()$
3: $\varphi' \leftarrow \{cl \mid cl \in \varphi$ and $weight(cl) \geq w_{min}\}$
4: **while** $true$ **do**
5: $(result, \varphi_{core}) \leftarrow SATSolver(\varphi')$
6: **if** $result$ is a satisfying assignment **then**
7: $curCost \leftarrow CalculateCost(result, \varphi)$
8: $\varphi \leftarrow Harden(\varphi, curCost - cost)$
9: **if** φ_{tmp} is empty **then**
10: **break**
11: **end if**
12: $\varphi' \leftarrow \varphi' \cup \varphi_{tmp}$
13: **continue**
14: **end if**
15: $\varphi_{core} \leftarrow minimize(\varphi', \varphi_{core})$
16: $cost \leftarrow cost + min_{v \in \varphi_{core}}(weight(v))$
17: $\varphi' \leftarrow relax(\varphi', \varphi_{core})$
18: $\varphi_{tmp} \leftarrow \varphi_{tmp} \cup createSum(\varphi_{core}, k)$
19: **end while**
20: **if** $w_{min} = 1$ **then**
21: **return** $cost$
22: **end if**
23: **end while**

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] 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.

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

Exact: evaluating a pseudo-Boolean solver on MaxSAT problems

Jo Devriendt
KU Leuven
Leuven, Belgium
jo.devriendt@kuleuven.be

***Abstract*—Weighted MaxSAT solving is a special case of pseudo-Boolean optimization, also known as binary linear programming. This submission aims to investigate whether Exact, a conflict-driven cutting planes learning pseudo-Boolean solver, is competitive on MaxSAT problems.**

***Index Terms*—binary linear programming, pseudo-Boolean solving, cutting planes, core-guided optimization**

I. Introduction

It is well-known[1] that a weighted MaxSAT formula can be written as a binary linear program (BLP):

$$\begin{aligned} \text{Minimize} \quad & \sum_{c \in C} w_c z_c \\ \text{s.t.} \quad & z_c + \sum_{x \in c^+} x + \sum_{y \in c^-} (1 - y) \geqslant 1 \qquad \forall c \in C \end{aligned}$$

where C is a set of clauses, c^+ and c^- are the set of positive and negative literals in a clause $c \in C$ respectively, w_c is the cost of not satisfying c, and all variables x, y and z are binary.

Even though this BLP formulation is natural, the state-of-the-art in previous MaxSAT evaluations employs repeated calls to Boolean satisfiability (SAT) solvers instead of one straightforward call to an integer linear programming (ILP) solver. Most likely, the reason for this is that ILP solvers rely heavily on exploiting the linear relaxation of a BLP, while all constraints in the above BLP are clauses, which have a particularly weak linear relaxation.

A third technology that could natively handle the above BLP however is pseudo-Boolean (PB) solving. Similar to ILP technology, PB technology natively takes linear constraints over binary variables as input. However, in contrast to ILP solvers, a PB solver does not chiefly depend on reasoning on the linear relaxation of a BLP. Instead, so-called *conflict-driven cutting-planes learning* (CDCPL) PB solvers derive (*learn*) from each conflict in the search tree an implied linear constraint that, if it had been derived previously, would have prevented the conflict through unit propagation. In this way, CDCPL PB solvers are a generalization of *conflict-driven clause learning* (CDCL) SAT solvers, where a CDCPL solver can learn stronger constraints than clauses.

[1]See, e.g., https://en.wikipedia.org/wiki/Maximum_satisfiability_problem#(1-1/e)-approximation

II. Submission

We submit the CDCPL solver Exact[2] to the MaxSAT evaluation. Exact is a fork of the CDCPL solver RoundingSat[3] [1]. For this submission, we do not employ RoundingSat's linear programming integration [2], as we expect the linear relaxations of the instances to be too weak. We do make use of its optimized propagation routines [3] and its hybrid core-guided optimization technique [4].

Exact improves upon its predecessor through a myriad of refactorings, extensions and improvements. We highlight three important ones for this MaxSAT evaluation submission.

A first one is the *stratification* routine of Exact's core-guided optimization. Instead of core-guided stratification based on [5], Exact uses a simple routine that ignores all soft clauses with a cost lower than some τ, which initially is set to the highest clause cost (the highest weight in the objective of the BLP representation). If Exact does not find a core with this τ (i.e., it finds a solution where all hard and non-ignored soft clauses are satisfied, or timeouts in the core-guided search) τ is halved, to consider more soft clauses. This process is repeated until the maximum cost is halved to 1, at which point all soft clauses are taken into account.

A second improvement is the exploitation of the observation that a single *PB core* may yield multiple *cardinality cores*, which can be used during the core-guided lower bound derivation and objective reformulation process [4]. For instance, given an objective function $4x+3y+2z+w$ to be minimized, and a PB core $2x + 2y + z + w \geqslant 4$, Exact constructs an initial implied cardinality core $x + y + z \geqslant 2$, reformulating the objective to $2x + y + w + 2a + 4$ through the *extension constraint* $x + y + z = 2 + a$. But as $2x + 2y + z + w \geqslant 4$ also implies $x + y + w \geqslant 2$, Exact can further reformulate the objective to $x + 2a + b + 6$ with the extension constraint $x + y + w = 2 + b$, increasing the objective lower bound from 4 to 6 without any new core-guided solver call.

A third improvement is meant to address the fact that, given a search conflict implied by only clausal constraints, CDCPL solvers can only learn a clause, which is identical to regular CDCL SAT solving (which has a more efficient implementation). For CDCPL to work well, non-clausal constraints need to appear in the conflict implication graph, so that

[2]https://gitlab.com/JoD/exact
[3]https://gitlab.com/miao_research/roundingsat

strong non-clausal constraints can be learned [6]. On MaxSAT instances, Exact introduces non-clausal constraints in three ways. Firstly, a derived upper or lower bound on the objective function is typically a non-clausal constraint. Secondly, a core-guided extension constraint also typically is equivalent to a conjunction of non-clausal constraints. Thirdly, implied cardinality constraints can be detected from a conjunction of clauses. Work on cardinality detection in RoundingSat exists [7], where an investigation of the implication graph during conflict analysis yields the right information to construct cardinality constraints. Exact uses a different approach, where repeated *probing* (deciding a single variable and running unit propagation) yields the necessary edges in the implication graph to derive *at-most-one* cardinality constraints.

III. Conclusion

By combining the effectiveness of CDCLP and core-guided optimization, PB solving technology may have become competitive to SAT-based approaches on MaxSAT problems. Exact's submission to 2021's MaxSAT evaluation will provide experimental data to support or reject this hypothesis.

IV. Acknowledgment

Exact is a fork of RoundingSat, which in turn uses code from MiniSat [8]. The author of Exact is Jo Devriendt (KU Leuven). The authors of RoundingSat are Jan Elffers (formerly Lund University), Jo Devriendt (KU Leuven), Stephan Gocht (Lund University) and Jakob Nordström (University of Copenhagen, Lund University). The authors of MiniSat are Niklas Eén (formerly University of California) and Niklas Sörensson (formerly Chalmers University of Technology).

References

[1] J. Elffers and J. Nordström, "Divide and conquer: Towards faster pseudo-Boolean solving," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, Jul. 2018, pp. 1291–1299.

[2] J. Devriendt, A. Gleixner, and J. Nordström, "Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search," in *Proceedings of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '20)*, ser. Lecture Notes in Computer Science, vol. 12296. Springer, Sep. 2020, pp. xxiv–xxv.

[3] J. Devriendt, "Watched propagation of 0-1 integer linear constraints," in *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, ser. Lecture Notes in Computer Science, vol. 12333. Springer, Sep. 2020, pp. 160–176.

[4] J. Devriendt, S. Gocht, E. Demirović, J. Nordström, and P. Stuckey, "Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning," vol. 35, no. 5, May 2021, pp. 3750–3758. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16492

[5] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy, "Improving SAT-based weighted MaxSAT solvers," in *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, ser. Lecture Notes in Computer Science, vol. 7514. Springer, Oct. 2012, pp. 86–101.

[6] S. R. Buss and J. Nordström, "Proof complexity and SAT solving," in *Handbook of Satisfiability*, 2nd ed., ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, Feb. 2021, vol. 336, ch. 7, pp. 233–350.

[7] J. Elffers and J. Nordström, "A cardinal improvement to pseudo-Boolean solving," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, Feb. 2020, pp. 1495–1503.

[8] N. Eén and N. Sörensson, "An extensible SAT-solver," in *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03), Selected Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2919. Springer, 2004, pp. 502–518.

MaxCDCL and WMaxCDCL in MaxSAT Evaluation 2022

1st Jordi Coll, 2nd Shuolin Li
Aix Marseille Univ,*Université de Toulon*
CNRS, LIS, Marseille, France
jordi.coll@lis-lab.fr, 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
Aix Marseille Univ,
Université de Toulon
CNRS, LIS, Marseille, France
Djamal.Habet@univ-amu.fr

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

I. Introduction

MaxCDCL and WMaxCDCL are two new unweighted and weighted partial MaxSAT solvers respectively. The main solving algorithm is MaxCDCL [1] for the MaxCDCL solver, and a weighted extension of MaxCDCL for the WMaxCDCL solver.

II. MaxCDCL 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 $LB = UB - 1$, all non-falsified soft clauses 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, the lower bound can be increased by one. A detailed description of the unweighted MaxCDCL algorithm can be found in [1], [5].

III. WMaxCDCL Algorithm

There are some adaptations that we do on the MaxCDCL algorithm to deal with weights. Here we describe the main ones. These particularities require the use many more data structures in WMaxCDCL solver. Therefore, although WMaxCDCL can solve unweighted instances, the source codes of MaxCDCL and WMaxCDCL are different.

The contribution to the lower bound for each core is not always one but it is the minimum weight among the soft clauses of the core. Hence, we make virtual copies of the soft clauses by splitting their weights so that every clause can belong to multiple local cores. More precisely, we dynamically decrease the weights of the soft clauses as they appear in new cores.

There are also relevant changes regarding hardening, since the fact that different soft clauses can have different weights implies that hardening can happen more frequently and at different decision levels. Moreover, after hardening soft clauses, usually more unit propagations can be done, which can falsify new soft clauses causing an increase the lower bound of the cost, which at turn can enable more hardening. Therefore, in WMaxCDCL, the unit propagation phase of CDCL is replaced by a fix point propagation loop that alternates unit propagation and hardening.

IV. Implementation details

Both MaxCDCL and WMaxCDCL solvers are implemented on top of MapleCOMSPS_LRB [6]. They include a number of preprocessing, inprocessing, and additional techniques to enhance their performance that we list in this section.

We compute a first upper bound *initUB* and lower bound *initLB* of the optimal cost, in order to limit the search, with a combination of methods. First, MaxHS [7] is run for 5 minutes to find initial bounds. The MaxHS version from [7] has been slightly adapted to deal with the new instance format and set time limits. The binary files submitted to the competition are compiled with IBM ILOG CPLEX version 22.1.

MaxHS in the 2022 MaxSat Evaluation

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

1. MaxHS

MaxHS originated in the work of Davies [4] who developed the first MaxSat solver based on the Implicit Hitting Set approach (IHS). The core components of MaxHS are described in [5], [6], [7]. The PhD thesis of Saikko [9] also provides an excellent overview of the IHS approach along with a number of additional insights. In addition to various algorithmic and code improvements over the years, MaxHS also employs the techniques of reduced cost fixing [1] and abstract cores [2]. Both of these techniques go beyond the basic IHS approach.

2. 2022

The 2022 version of MaxHS is built on top of the 2021 version. As noted in last years entry, as MaxSat instances have become larger the relative advantages of using a more powerful Sat solver have increased. Hence, MaxHS now uses the Cadical solver for Sat solving [3] (the potentially more efficient Kissat does not fully support all of the Sat solver features, e.g., assumptions, needed by MaxHS). It also uses IBM's commercial Mixed Integer Programming Solver (IBM CPLEX version 20.1.0.0) under IBM's Academic Initiative licencing program.

In 2022 the input parser and option processing subsystems were rewritten, and the preprocessing phase was redesigned and reimplemented. The MaxSat preprocessor MaxPre [8] had been tried before without positive results. This arises from a different processing of soft clause "labels" (also known as the relaxation or blocking variables) done in MaxPre which often produces far more blocking variables than MaxHS. In particular, MaxPre will introduce a new "label" even for unit soft clauses (when the literal of that soft clause appears in both polarities in the rest of the formula). MaxHS on the other hand always reuses the literal of unit softs as the "label", irrespective of its purity in the rest of the formula. Since much of the MaxSat processing is dependent the number of blocking variables, we have found that MaxHS can be more effective without MaxPre. Nevertheless, sometimes MaxPre is able to simplify the problem more profoundly. In the 2022 code, MaxPre is run, but the simplified formula it produces is only used if it contains fewer labels than the original formula. In future work we plan to identify and integrate only those parts of MaxPre that prove to be useful (which mainly seem to be the technique "generalized subsumed label elimination").

The second change introduced in 2022 was to identify a wider range of special input cases and then configure the solver more specifically for solving these special cases. The special cases identified in the 2022 code are (a) when the input is a hitting set problem, (b) when the input has a small number of soft clauses, (c) when every variable of the formula forms a unit soft clause, and (d) when the input consists of a small number of variables. Cases (a) and (b) are newly identified in 2022. For case (a) we have found that using the MIP solver tended to produce the best results, and for case (b) using a simple time bounded Linear Sat Unsat algorithm which can improve results in the weighted case. Case (c) and (d) were handled in the 2021 code, and are processed by a mixture of first trying the MIP solver for a short period of time, then trying the technique of abstract cores, and then utilizing the information gathered so far for a final round of MIP solving. In future work, we plan on monitoring the progress of these "trial" phases more carefully so that better decisions can be made about when to abort these phases.

References

[1] Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in MaxSAT. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2017.

[2] Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set maxsat solving. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2020.

[3] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger. Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020. In *Proceedings of the SAT Competition 2020–Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, page 5153. University of Helsinki, 2020.

[4] Jessica Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. PhD thesis, University of Toronto, 2013. https://tspace.library.utoronto.ca/bitstream/1807/43539/6/Davies_Jessica_E_201311_PhD_thesis.pdf.

[5] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011.

[6] Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MaxSAT. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013.

[7] Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013.

[8] Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An extended MaxSAT preprocessor. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017.

[9] Paul Saikko. *Implicit hitting set algorithms for constraint optimization*. PhD thesis, University of Helsink, 2019. https://helda.helsinki.fi/bitstream/handle/10138/306951/implicit.pdf?sequence=1&isAllowed=y.

Open-WBO @ MaxSAT Evaluation 2022

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

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

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

I. Introduction

OPEN-WBO [1] is an open source MaxSAT solver that supports several MaxSAT algorithms [2], [3], [4], [5], [6], [7], [8] and SAT solvers [9], [10], [11]. OPEN-WBO is particularly efficient for unweighted MaxSAT and has been one of the best solvers in the MaxSAT Evaluations from 2014 to 2017. Two versions of OPEN-WBO were submitted to the unweighted track at MaxSAT Evaluation 2022: open-wbo-res-mergesat and open-wbo-res-glucose. The only difference between OPEN-WBO 2022 and the 2021 version is a 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 [9], [12]. Therefore, solvers based on MINISAT 2.2 can be used as a potential back-end solver. For the MaxSAT Evaluation 2021, we use GLUCOSE 4.1 [10], [13], [14] as the back-end SAT solver of the version that ends in glucose and MERGESAT [11] as the back-end SAT solver of the version that ends in mergesat.

MERGESAT [11] is a new CDCL solver developed by Norbert Manthey and it is based on the SAT competition winner of 2018, MAPLELCMDISTCHRONOBT [15], and adds several known techniques. For restarts, only partial backtracking is used, learned clause minimization is implemented more efficiently, and also applies simplification again in case the first swipe resulted in a simplification. The time-based decision heuristic switch is made deterministic by using solving steps. Assumption literals are set before search, and the CCNR SLS engine, as well as polarity selection during decision with rephasing is used. To support being used inside MaxSAT solvers, the incremental search feature had to be enabled again.

III. MaxSAT Algorithms

In this section, we briefly describe the algorithms used for the complete track at the MSE2021.

A. Complete Unweighted Track

Two versions were submitted to the complete unweighted track: open-wbo-res-mergesat and open-wbo-res-glucose.

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

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

REFERENCES

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

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

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

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

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

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

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

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

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

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

[11] N. Manthey, "The SAT solver MergeSat," in *SAT*. Springer, 2021.

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

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

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

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

[16] 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.

UWrMaxSat Entering the MaxSAT Evaluation 2022

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

***Abstract*—UWrMaxSat is a complete solver for partial weighted MaxSAT instances and pseudo-Boolean ones. It can be also characterized as an anytime solver, since it outputs the best known solution, whenever its run is interrupted. It needs a SAT solver as an oracle and can be used with a few modern solvers, from which COMiniSatPS by Chanseok Oh (2016) has been selected as a default one. Several solving strategies have been implemented in it (selected by parameters) but the default one is a core-guided OLL procedure, where its own sorter-based pseudo-Boolean constraint encoding is applied to translate cardinality constraints into CNF. This paper describes new elements in UWrMaxSat version 1.4, which is submitted to the MaxSAT Evaluation 2022. They include (1) the IPAMIR interface that standardizes an access to a MaxSAT solver as a library, and (2) an integration with SCIP solver for mixed integer programming.**

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

I. Introduction

An example of optimization problems can be represented as a partial weighted MaxSAT instance or, equivalently, as a set of linear inequation over Boolean variables, called pseudo-Boolean (PB), with a linear goal function, which value should be minimized. A MaxSAT instance consists of two sets containing hard clauses and weighted soft ones, respectively. Hard ones must be satisfied by any solution and the goal of MaxSAT optimization is to find a model that minimizes also the sum of weights of unsatisfied soft ones. It is clear that clauses can be easily converted into linear inequations over Boolean variables. Inverse translations are used in many PB solvers, but one needs an efficient encoder to convert PB constraints into clauses.

UWrMaxSat was created as an extension of the MiniSat+ 1.1 solver by Eén and Sörensson (2012) [5] in two basic steps: firstly, this PB solver was extended with a new sorter-based encoder by Michał Karpiński and Marek Piotrów to, so called, KP-MiniSat+ [7], [8], and then, a MaxSAT parser and the corresponding solving techniques were added to it by Piotrów and the solver was renamed to UWrMaxSat. Therefore, it is able to solve both PB and MaxSAT instances. Moreover, it appeared that the implemented translation of an PB instance into an equivalent MaxSAT one and the selected MaxSAT algorithm become an efficient way to solve many PB examples of optimization problems [9].

But, first of all, UWrMaxSat is a competitive complete MaxSAT solver, as it was shown by results od MaxSAT Evaluations (MSE) in the last three years. Furthermore, outcomes of the complete tracks of MSE 2021 indicated that translating clauses into linear inequation over 0-1 integer variables and using the mixed integer programming solver SCIP [4] to solve them can be a successful way to find the optimal solution of several small MaxSAT instances. Therefore, this method has been added as an option to UWrMaxSat.

Furthermore, this year's version is submitted with two new features: (1) the IPAMIR interface that has been defined to standardize the way of using a solver as a library, and (2) an improved greedy algorithm of the encoder module to better check if some of previously encoded sorting networks can be reused in a new encoding.

II. Description

A new version of UWrMaxSat is denoted as 1.4 and it is a fourth time when the solver takes part in MaxSAT Evaluations. For the main features of the previous versions see [14], [15], [17]. A more detailed description of UWrMaxSat ver. 1.1 can be found in [16]. In this year's version, we continue to use incrementally the SAT solver COMiniSatPS by Chanseok Oh (2016) [13] as an oracle. The default search strategy for the optimal solution is also the same as in previous years, that is, a core-guided linear one, where unsatisfiability cores are processed by the OLL procedure [1], [6], [11] and cardinality constraints generated by it are encoded with the help of 4-Way Merge Selection Networks [8] and Direct Networks [3]. The general description of search strategies used by MaxSAT solvers can be found, for example, in [12].

UWrMaxSat can be compiled with or without two additional libraries: the extended MaxSAT preprocessor MaxPre created at the University of Helsinki and implemented by Tuukka Korhonen [10], and the mixed integer programming solver SCIP created in cooperation of several organizations [4]. The submitted UWrMaxSat 1.4 uses only the second library in version 8.0.0. The default configuration is to run the SCIP solver on small MaxSAT instances in a separate thread without given timeout. Using the corresponding options one can force it to be run in the same thread and for a defined number of seconds before the start of the MaxSAT solver (these options are set in the competition). An input instance is preprocessed before sending it to SCIP by the UWrMaxSat's implementation of an algorithm to detect unit clauses and at-most-one cardinality constraints and, then, by simplifications done by the SAT solver.

The recently-defined IPAMIR interface has been added to UWrMaxSat to standardize the API of the corresponding

library. All elements of IPAMIR have been implemented. The only restriction is that the library allows an application to use a single instance of the solver at a time. In version 1.4 of the library, MaxPre is not used and should be compiled without it. On the other hand, it can be compiled to be used with the SCIP library and the corresponding options can be set in an environment variable UWRFLAGS (as well as some other useful ones of UWrMaxSat).

In the IPAMIR implementation, a general assumption is to have a single MaxSat-solver object and a corresponding SAT-solver one for a whole solving process between the initial call to `ipamir_init` and the final call to `ipamir_release`. That means that the SAT solver keeps and used all learnt pieces of information. To have a consistent state of it, the MaxSat solver replaces all self-generated unit clauses by assumptions that are discarded after the end of a call to `ipamir_solve`. In particular, such replacements are done in the hardening procedure of soft literals.

Hard clauses added through the IPAMIR interface are sent directly to the SAT solver and kept only there. Sets of weighted soft literals and assumptions can be changed between calls to `ipamir_solve`, so they are analysed from scratch each time the MaxSat solver is used. Assumptions from IPAMIR are merged with ones from the solver and are used as assumptions to each SAT solving. A stratification technique [2] is applied to the set of weighted literals. Cores obtained from the SAT solver are encoded as cardinality constrains with a method described shortly below.

A sorter-based encoding of PB constraints was implemented in the original MiniSat+ solver. Next, it was replaced by a much more complicated one in KP-MiniSat+, but still each constraint was encoded separately into clauses. Recently, Michał Karpiński and me have proposed an improvement of such technique and implemented it in UWrMaxSat. By using a variation of a greedy set cover algorithm, when adding constraints to our encoding, the encoder reuses parts of the already encoded PB-instance in order to decrease the size (the number of variables and clauses) of the resulting SAT instance [9].

Finally, the parser of UWrMaxSat was modified to accept the new proposed input format of partial weighted MaxSAT instances. Recall that there is no p-line in it and hard clauses are preceded by the letter 'h' instead of the "very big" weight given in a p-line.

REFERENCES

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

[2] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving sat-based weighted maxsat solvers. In Michela Milano, editor, *Proc. CP*, pages 86–101. Springer, 2012.

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

[4] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 8.0. ZIB-Report 21-41, Zuse Institute Berlin, December 2021.

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

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

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

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

[9] Michał Karpiński and Marek Piotrów. Reusing comparator networks in pseudo-boolean encodings. *arXiv preprint arXiv:2205.04129*, 2022.

[10] Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An extended MaxSAT preprocessor. In Serge Gaspers and Toby Walsh, editors, *Proc. SAT*, volume 10491 of *LNCS*, pages 449–456. Springer, 2017.

[11] Antonio Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In Barry O'Sullivan, editor, *Proc. CP*, LNCS, pages 564–573. Springer, 2014.

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

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

[14] Marek Piotrów. Uwrmaxsat - a new minisat+-based solver in maxsat evaluation 2019. In Fahiem Bacchus, Matti Järvisalo, and Ruben Martins, editors, *MaxSAT Evaluation 2019 : Solver and Benchmark Descriptions*, Department of Computer Science Report Series B-2019-2, pages 11–12, 2019.

[15] Marek Piotrów. Uwrmaxsat - an efficient solver in maxsat evaluation 2020. In Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins, editors, *MaxSAT Evaluation 2020 : Solver and Benchmark Descriptions*, Department of Computer Science Report Series B-2020-2, pages 34–35, 2020.

[16] Marek Piotrów. Uwrmaxsat: Efficient solver for maxsat and pseudo-boolean problems. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 132–136, 2020.

[17] Marek Piotrów. Uwrmaxsat in maxsat evaluation 2021. In Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins, editors, *MaxSAT Evaluation 2021 : Solver and Benchmark Descriptions*, Department of Computer Science Report Series B-2021-2, pages 17–18, 2021.

WMaxCDCL-BandAll in MaxSAT Evaluation 2022

1st Jordi Coll, 2nd Shuolin Li
Aix Marseille Univ,*Université de Toulon*
CNRS, LIS, Marseille, France
jordi.coll@lis-lab.fr, 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
Aix Marseille Univ,
Université de Toulon
CNRS, LIS, Marseille, France
Djamal.Habet@univ-amu.fr

6th Jiongzhi Zheng, 7th Kun He
Huazhong University of Science and Technology
Wuhan,China
jzzheng@hust.edu.cn,brooklet60@hust.edu.cn

I. WMaxCDCL-BandAll

WMaxCDCL-BandAll is a modification of the WMaxCDCL solver, which also participates in MaxSAT Evaluation 2022 (see *MaxCDCL and WMaxCDCL in MaxSAT Evaluation 2022*). The basic algorithm is the same: a an addaptation to weighted MaxSAT of the MaxCDCL algorithm [1], [2]. WMaxCDCL-BandAll includes the same preprocessing and inprocessing techniques described for WMaxCDCL. The difference is in WMaxCDCL-BandAll is in the local search technique used to improve the found suboptimal solutions. This procedure has been replaced by the BandAll local search method implemented in the DT-HyWalk solver, which participates in the incomplete track of MaxSAT Evaluation 2022 (see *Decision Tree based Hybrid Walking Strategies*). This method is a variant of BandMaxSAT [3].

Acknowledgments

This work has been partially funded by the French Agence Nationale de la Recherche, reference ANR-19-CHIA-0013-01, and project PID2019-111544GB-C21 funded by MCIN/AEI/10.13039/501100011033, and partially supported by Archimedes Institute, Aix-Marseille University. F. Manyà was supported by mobility grant PRX21/00488 of the *Ministerio de Universidades*. 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] ——, "Boosting branch-and-bound maxsat solvers with clause learning," *AI Communications*, no. Preprint, pp. 1–21, 2021.

[3] 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.

Decision Tree based Hybrid Walking Strategies

Jiongzhi Zheng[1], Kun He[1], Zhuo Chen[1], Jianrong Zhou[1], Chu-Min Li[2]
[1]School of Computer Science and Technology, Huazhong University of Science and Technology, China
[2]MIS, Université de Picardie Jules Verne, France
{jzzheng, brooklet60}@hust.edu.cn

***Abstract*—This document describes our MaxSAT solver DT-HyWalk submitted to the MSE 2022. DT-HyWalk is submitted to both the weighted and unweighted incomplete tracks.**

I. Introduction

DT-HyWalk contains two components, the local search component and the SAT-based component. The SAT-based component is the TT-Open-WBO-Inc solver [1]. The combination of the local search component and TT-Open-WBO-Inc is the same as that of SATLike-c [2]. That is, first using an SAT solver to obtain a feasible solution on all the hard clauses, then using the local search component to improve the solution until there is no improvement over 10^7 steps, finally using TT-Open-WBO-Inc to calculate in the rest time with the upper bound equals to the best solution found by local search.

In the local search component of DT-HyWalk, we implement several local search algorithms including BandMaxSAT [3], FPS [4], SimpleWalk (another proposed local search method), as well as their variants and some combinations of them. These local search algorithms all follow the same searching process as SATLike [5] does when the local optima for SATLike are not reached (i.e., there is at least one variable with a positive *score*). When they reach a local optimum for SATLike, they perform different walking strategies to escape from the local optimum.

We first introduce the three basic algorithms, BandMaxSAT, FPS, and SimpleWalk, then introduce some of their variants and combinations, finally introduce the decision tree based hybrid method of DT-HyWalk.

II. BandMaxSAT

BandMaxSAT [3] 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*

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*

```
1  A* := A, cost(A') := +∞, N := 0;
2  while running time < cutoff do
3      if A is feasible & cost(A) < cost(A*) then
4          A* := A;
5      if D := {x|score(x) > 0} ≠ ∅ then
6          v := a variable in D picked by BMS(k);
7      else
8          update_clause_weights();
9          if ∃ falsified hard clauses then
10             c := a random falsified hard clause;
11         else
12             update_estimated_value(A, A', A*, d, γ);
13             N := N + 1, A' := A;
14             c := PickArm(ArmNum, N, λ);
15             t(c) := t(c) + 1
16         v := the variable with the highest score in c;
17     A := A with v flipped;
18 if A* is feasible then return A*;
19 else return no feasible assignment found;
```

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 $cost(A^*)$, 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 [4] combines the look-ahead strategy with the probabilistic sampling strategy in an effective way. FPS applies the same method as SATLike and BandMaxSAT do when the algorithm does not reach a local optimum for SATLike. When a local optimum is reached, FPS first randomly samples 10 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 SATLike, so as to find higher-quality solutions. While the probabilistic sampling strategy can help the algorithm improve its efficiency.

IV. SimpleWalk and Others

SimpleWalk is a local search algorithm with a simple walking strategy. When the algorithm falls into local optima, it first randomly samples 10 falsified clauses, then randomly samples 5 variables in each sampled clause, finally selects to flip the sampled variable with the highest *score*. Such a method has a wider walking scope than SATLike does, thus can find better search directions to escape from local optima.

There are also some variants and combinations of BandMaxSAT, FPS, and SimpleWalk in DT-HyWalk, including:

- FPS+SimpleWalk (FS): first randomly sample clauses and variables as SimpleWalk does, then look-ahead from the sampled variable with the highest *score* in each sampled clause.
- BandMaxSAT+FPS (BF): first select the falsified clause to be satisfied as BandMaxSAT does, then randomly sample 10 variables in the selected clause to look-ahead.
- BandAll (BA): an extension of BandMaxSAT that associates a multi-armed bandit to all the clauses including both hard and soft ones. When the algorithm falls into a local optimum (feasible or infeasible), it calls the bandit model to select to pull an arm (satisfy a falsified clause). The updating of the estimated values as well as the arm selection process for hard clauses and soft clauses are independent.
- ...

V. Decision Tree based Hybrid Walking

Different walking strategies such as those in BandMaxSAT and FPS are suitable for different kinds of instances. Therefore, to help the solver decide to select an appropriate walking strategy to explore the solution space, we use a decision tree that trains on all the instances from the incomplete tracks of the last four years of MSE. The features include the number of variables $NVARS$, the number of clauses $NCLS$, the number of hard clauses $NHARDS$, the number of soft clauses $NSOFTS$, the proportion of soft clauses $SOFT\_PERCENT$, the minimum, average, maximum length of hard clauses $MIN\_HARD, AVG\_HARD, MAX\_HARD$, the minimum, average, maximum length of soft clauses $MIN\_SOFT, AVG\_SOFT, MAX\_SOFT$, the average length of all clauses AVG_LEN, the minimum, average, maximum weight of soft clauses $MIN\_WTS, AVG\_WTS, MAX\_WTS$, a total of 15.

DT-HyWalk combines 6 walking strategies for weighted instances ($MAX\_WTS > 1$), and 5 walking strategies for unweighted instances. When solving an instance, the solver uses the decision tree to select an appropriate walking strategy.

References

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

[2] Z. Lei, S. Cai, F. Geng, et al., "SATLike-c: Solver Description," MaxSAT Evaluation 2021, 2021: 19-20.

[3] J. Zheng, J. Zhou, K. He, "Farsighted Probabilistic Sampling based Local Search for (Weighted) Partial MaxSAT," arXiv preprint arXiv:2108.09988, 2021.

[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.

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

Loandra in the 2022 MaxSAT Evaluation

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

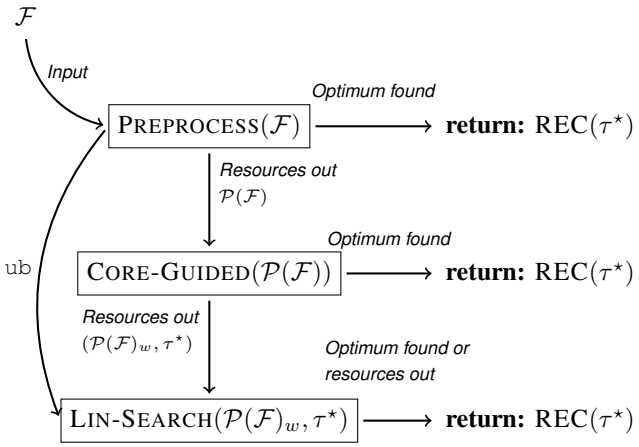

Fig. 1: The structure of Loandra.

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 re-

sembles 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: Solver Description

Yi Chu[1], Shaowei Cai[1,2], Zhendong Lei[1,2], and Xiang He[1,2]

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

***Abstract*—This document describes the solver NuWLS-c, submitted to the four incomplete tracks of MaxSAT Evaluation 2022.**

I. Introduction

NuWLS-c is based on SATLike-c [3]. NuWLS-c has two main engines, one is the local search solver NuWLS and the other is the SAT-based solver TT-Open-WBO-inc [2].

II. Local Search Algorithm: NuWLS

Our NuWLS algorithm utilizes the framework of the local search algorithm SATLike [1], which is a dynamic local search framework for SAT and exploits the distinction of hard and soft clauses by a clause weighting scheme.

We propose a new clause weighting scheme for updating clause weights during the local search, which makes a deeper distinction between hard and soft clauses. The weighting scheme used in NuWLS is named **Dist-Weighting** (Distinguished Weighting).

First, usually, SLS algorithms update clause weights when encountering local optima. Nevertheless, previous clause weighting schemes either update only the hard clause weights, or update both hard and soft clause weights under the same conditions. Our Dist-Weighting scheme updates hard and soft clause weights according to different activation conditions.

Second, the framework of the SATLike algorithm uses a scoring function (the score of the variable, $score(x)$) to guide the search. $score(x)$ is the increment of the total weight of satisfied clauses (either hard clauses or soft clauses) caused by flipping x. For the weighted instances, when the weights of soft clauses are updated, the importance between soft clauses could be destroyed, thus leading the search to a region where there is no chance of finding better solutions. The issue is more pronounced when the average weight of soft clauses is small. In fact, SATLike does not increase the weights of soft clauses when solving weighted instances where the average weight of soft clauses ($ws_{avg}(I)$) is less than 10000. Additionally, too large average weight of soft clauses disrupts the balance between the hard and soft clauses. To address this issue, we propose a weighting scheme where the weight of each soft clause c ($w(c)$) is initially set to 1, and the upper limit of the soft clause weight is proportional to its original weight $w_{org}(c)$. Set a parameter s_{avg}, for an instance I whose average weight of soft clauses is $ws_{avg}(I)$, then the upper limit of the weight of each soft clause c in instance I is $\frac{s_{avg} \times w_{org}(c)}{ws_{avg}(I)} + \delta$, where δ is a parameter.

For each unweighted instances I, $ws_{avg}(I) = 1$, the weight of each soft clause in I is 1 (i.e., $w(c)$ = 1), then the upper limit of the weight of each soft clause c in I is $s_{avg} + \delta$.

Based on the above two ideas, we present our new weighting scheme. For each clause, the initial weight is set to 1. When the algorithm encounters a local optimum, the clause weights are updated as follows:

- For hard clauses: with probability h_sp and the condition that a feasible solution is found in the current round of local search, for each satisfied hard clause c, $w(c) := w(c) - h\_inc$ if $w(c) > h\_inc$; otherwise, for each falsified hard clause c, $w(c) := w(c) + h\_inc$.
- For soft clauses: the weights of soft clauses are only updated if $cost(\alpha) \geq cost(\alpha^*)$. Specifically, with probability s_sp, for each satisfied soft clause c, $w(c) := w(c)\text{-}s\_inc$ if $w(c) > s\_inc$; with probability $1 - s\_sp$, if α is feasible, then for each falsified soft clause c, $w(c) := w(c) + s\_inc$ if $w(c) < \frac{s_{avg} \times w_{org}(c)}{ws_{avg}(I)} + \delta$. (We use α to denote the current assignment, α^* is used to denote the best solution found, $cost(\alpha)$ is used to denote the sum of $w_{org}(c)$ of all the unsatisfied soft clauses under α.)

III. Hybrid Solver: NuWLS-c

We combine NuWLS with the state of the art SAT-based solvers TT-Open-WBO-inc [2], leading to the hybrid solver NuWLS-c.

The framework of NuWLS-c is similar to SATLike-c [3].

References

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

[2] 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

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

noSAT-MaxSAT

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

Sibylle Schupp
Institute for Software Systems
Hamburg University of Technolog (TUUH)
Hamburg, Germany
schupp@tuhh.de

***Abstract*—Since 2019, all solvers in the incomplete track of the MaxSAT Evaluation (MSE) include a dedicated SAT solver. In resource-constrained computing environments, e.g., embedded systems, such algorithm designs can be hard to realize. Yet, the MSE results show an undeniable increase in efficiency when using a SAT solver. With *noSAT-MaxSAT*, we propose to replace the call to a SAT solver by executing the MaxSAT solver only on the hard clauses of the formula instead. The algorithm is based on SATLike, which was was the only algorithm that did not rely on a SAT solver in the MaxSAT Evaluation 2018. Additionally, our implementation satisfies a set of requirements often found in embedded system software.**

I. Introduction

In recent years the solvers in the incomplete track of the MaxSAT Evaluation (MSE) have converged to employ algorithms that rely on complete SAT solvers. Indeed, since 2019, that is true for all incomplete solvers that entered the MSE. 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].

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 with regards to 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-MaxSAT is based on the SATLike algorithm by Lei and Cai [4], which is the most recently proposed algorithm in the MSE that does not rely on a SAT solver. SATLike entered the MSE 2018 together with its variant SATLike-c, which does employ an external SAT solver to obtain a satisfying assignment for hard clauses and consistently performed better than the former [5]. On the one hand, its performance highlights the undeniable effectiveness of solving MaxSAT problems through SAT. On the other hand, this leads to the core idea of *noSAT-MaxSAT*: Replacing the execution of a SAT solver in SATLike-c by running SATLike itself instead, but only on the hard clauses.

In the following, we introduce the requirements and architecture of *noSAT-MaxSAT*, and briefly describe the SATLike algorithm and the modifications made for *noSAT-MaxSAT*.

II. 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. It applies the SATLike algorithm as described in the following section. 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 much 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.

III. Algorithm

Algorithm 1 noSAT-MaxSAT

Require: partial weighted MaxSAT formula F, unsigned integers $maxFlips$ and $maxTries$, SATLike parameters
Ensure: a feasible assignment for F and the resulting total cost, or no assignment

```
minCost ← ∞
bestAssignment ← random
for t ← 0;
t < maxTries and no. of unsat. clauses > 0;
increment t by 1
 do
  assign bestAssignment to F
  preprocess F
  if no. of soft clauses > 0 then
    execute noSAT-MaxSAT on hard clauses of F
  end if
  for f ← 0;
  f < maxFlips and no. of unsat. clauses > 0;
  increment f by 1
   do
    if current cost < minCost then
      minCost ← current cost
      bestAssignment ← current assignment
      f ← 0
    end if
    v ← select variable according to SATLike
    flip v
  end for
  if current cost < minCost then
    minCost ← current cost
    bestAssignment ← current assignment
  end if
end for
```

A high-level description of *noSAT-MaxSAT* is given in Algorithm 1. It is based on the SATLike algorithm by Lei and Cai that employs an effective clause-reweighting and variable selection mechanism [4]. For preprocessing it is combined with a unit clause propagation-based method that was introduced by Cai et al. [6] and was also present in the original implementation. After preprocessing, SATLike is executed only on the hard clauses of the formula, trying to obtain a satisfying assignment for them. As mentioned earlier, such a step can greatly improve efficiency, but is usually performed by a dedicated SAT solving algorithm. Details on the SATLike algorithm and its parameters are omitted here for brevity; we use the same parameters as reported by Lei and Cai [4]. The parameter $maxTries$ is set to the largest possible value (`UINT64_MAX`) so the solver runs until it is stopped by an external signal, as the MSE rules for incomplete solvers require. $maxFlips$ is initialized with 10 times the number of variables of the input formula to allow for restarts and ensure the feedback mechanism of the preprocessing method is actually used (conflicting unit clauses are resolved by setting the affected variable to its value from $bestAssignment$) [6].

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 (visited on 05/13/2022).

[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 (visited on 05/13/2022).

[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 (visited on 04/25/2022).

[4] Zhendong Lei and Shaowei Cai. "Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI-18. Stockholm, Sweden, July 2018, pp. 1346–1352. DOI: 10.24963/ijcai.2018/187.

[5] Ruben Martins, Matti Jarvisalo, and Fahiem Bacchus. "MaxSAT Evaluation 2018". SAT 2018 (Oxford, UK). July 2018. URL: https://maxsat-evaluations.github.io/2018/mse18-talk.pdf (visited on 05/13/2022).

[6] 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.

Open-WBO-Inc in MaxSAT Evaluation 2022

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

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

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

I. Introduction

Open-WBO-Inc [1], [2] is developed on top of Open-WBO [3], [4], [5] and placed first and second on the weighted incomplete tracks for 60 and 300 seconds respectively in the MaxSAT Evaluation 2018, and third on both these tracks in the MaxSAT Evaluation 2019. For many applications that can be encoded into MaxSAT, it is important to quickly find solutions even though these may not be optimal. Open-WBO-Inc is designed to find a good solution[1] in a short amount of time. Since Open-WBO-Inc is based on Open-WBO, it can use any MiniSAT-like solver [6]. For this evaluation, we use Glucose 4.1 [7] as our back-end SAT solver. Open-WBO-Inc 2022 uses a modified parser that supports to the new format where hard clauses are marked with 'h' and the 'p-line' is removed.

II. Algorithms

For the MaxSAT Evaluation 2022, we restrict Open-WBO-Inc to the weighted category where it uses the approximation algorithms that have been proposed in past work [1], [2]. In particular, we submitted two versions of Open-WBO-Inc: inc-bmo-complete and inc-bmo-satlike.

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

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

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

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

III. Availability

We submit the source of Open-WBO-Inc as part of our submissions to the MaxSAT Evaluation 2022. The inc-bmo-complete version and the full Open-WBO-Inc framework is available under a MIT license in GitHub at *https://github.com/sbjoshi/Open-WBO-Inc*.

Acknowledgments

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

References

[1] S. Joshi, P. Kumar, R. Martins, and S. Rao, "Approximation Strategies for Incomplete MaxSAT," in *CP*. Springer, 2018.
[2] S. Joshi, P. Kumar, S. Rao, and R. Martins, "Open-WBO-Inc: Approximation Strategies for Incomplete Weighted MaxSAT," in *Journal on Satisfiability, Boolean Modeling and Computation*. IOS Press, 2019.
[3] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a Modular MaxSAT Solver," in *SAT*, ser. LNCS, vol. 8561. Springer, 2014, pp. 438–445.
[4] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental Cardinality Constraints for MaxSAT," in *CP*. Springer, 2014, pp. 531–548.
[5] M. Neves, R. Martins, M. Janota, I. Lynce, and V. Manquinho, "Exploiting Resolution-Based Representations for MaxSAT Solving," in *SAT*. Springer, 2015, pp. 272–286.
[6] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*. Springer, 2003, pp. 502–518.
[7] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI*, 2009, pp. 399–404.

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

[9] D. Le Berre and A. Parrain, "The Sat4j library, release 2.2," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, no. 2-3, pp. 59–6, 2010.

[10] Z. Lei and S. Cai, "Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT," in *IJCAI*. ijcai.org, 2018, pp. 1346–1352.

[11] A. Nadel, "Anytime weighted maxsat with improved polarity selection and bit-vector optimization," in *Proc. Formal Methods in Computer Aided Design*, C. W. Barrett and J. Yang, Eds. IEEE, 2019, pp. 193–202.

TT-Open-WBO-Inc-22: an Anytime MaxSAT Solver Entering MSE'22

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

***Abstract*—This document describes the solver TT-Open-WBO-Inc-22, submitted to the four incomplete tracks of MaxSAT Evaluation 2022. TT-Open-WBO-Inc-22 is the 2022 version of our solver TT-Open-WBO-Inc [8], itself based on Open-WBO-Inc [3]. The main innovation in TT-Open-WBO-Inc-22 is the integration of our new open-source SAT solver Intel® SAT Solver (IntelSAT) [5].**

I. Introduction

TT-Open-WBO-Inc [8] is our anytime MaxSAT solver, based on Open-WBO-Inc [3]. Similarly to the previous year's version [9], TT-Open-WBO-Inc-22 combines the following algorithms:

1) SATLike local search [2] for inprocessing.
2) The unweighted component uses Mrs. Beaver [6], enhanced by the following two heuristics from Sect. 4.1 in [4]: global stopping condition for OBV-BS and size-based switching to complete part.
3) The weighted component uses BMO-based clustering [3].
4) The Polosat SAT-based local search algorithm [7] 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 two latest MaxSAT Evaluations.

The main innovation this year is the development and integration of our new open-source SAT solver Intel® SAT Solver (IntelSAT) [5], available at [10]. IntelSAT is optimized for applications which generate many mostly satisfiable incremental SAT queries, such as unweighted anytime MaxSAT, for which IntelSAT's performance was specifically optimized [5].

We submitted three versions of TT-Open-WBO-Inc-22, the difference being the underlying SAT solver:

1) TT-Open-WBO-Inc-22(I): with IntelSAT.
2) TT-Open-WBO-Inc-22(IS): with IntelSAT, tuned for shorter invocations.
3) TT-Open-WBO-Inc-22(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] S. Cai and Z. Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artif. Intell.*, 287:103354, 2020.
[3] 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.
[4] A. Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *FMCAD 2019*, pages 193–202.
[5] A. Nadel. Introducing Intel(R) SAT Solver. In *SAT 2022*. To appear.
[6] A. Nadel. Solving MaxSAT with bit-vector optimization. In *SAT 2018*, pages 54–72, 2018.
[7] 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.
[8] A. Nadel. Polarity and variable selection heuristics for SAT-based anytime MaxSAT. *J. Satisf. Boolean Model. Comput.*, 12(1):17–22, 2020.
[9] A. Nadel. TT-Open-WBO-Inc-21: an Anytime MaxSAT Solver Entering MSE'21. Department of Computer Science Report Series B. University of Helsinki, 2021.
[10] A. Nadel. Intel® SAT Solver. https://github.com/alexander-nadel/intel_sat_solver, 2022.

Incremental version of EvalMaxSAT 2022

Carl-Elliott Bilodeau-Savaria*, Lancelot Normand†, Florent Avellaneda‡
Computer Science Departement, Université du Québec à Montréal
QC, Canada
Email: *bilodeau-savaria.carl-elliott@uqam.ca, †normand.lancelot@uqam.ca, ‡avellaneda.florent@uqam.ca

I. Introduction

EvalMaxSAT[1] is a MaxSAT solver written in modern C++ language mainly using the Standard Template Library (STL). The solver is built on top of the SAT solver 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].

II. Description

Our approach to solve a sequence of n MaxSAT formulas $\phi^1, \phi^2, \dots \phi^n$ is by using two instances of EvalMaxSAT solvers (E_1, E_2). For each iteration i, instance E_1 will try to satisfy formula $\phi^i = \phi_H \cup \phi_S$ (a set of soft and hard clauses). If E_1 found the formula ϕ^i to be satisfiable, then instance E_2 will be constructed as a copy of E_1. Instance E_2 will try to find the optimal solution to the original formula with the user assumptions added ($\phi^i \cup \phi_U$), reusing everything (cardinality constraints, weights, assumptions, etc) from instance E_1 that was generated to solve formula ϕ^i. Note that user assumptions (`ipamir_assume`) are added as hard unit clauses in instance E_2 rather than as conventional assumptions that would be passed to the SAT solver.

Instance E_2 will return the result it found, and this iteration will be complete. Instance E_1 will never be destroyed as it will be used to conserve relevant data structures (i.e., the cardinality constraints). If at some point instance E_1 found formula ϕ^k to be unsatisfiable, the program should stop since if ϕ^k is `UNSAT`, ϕ^{k+1} is also `UNSAT` since $\phi^k \in \phi^{k+1}$.

Should the weight of at least one soft literal be changed via `ipamir_add_soft_lit`, instance E_1 will restart from zero with the updated weights. Although simpler from a development standpoint, resetting E_1 at the first sign of a weight change is not efficient. This will be fixed in later versions. Furthermore, additional incremental approaches, such as reusing cores and cardinality constraints, will be added in future iterations to complement the techniques described above.

Algorithm 1

Input: A formula φ

1: $\delta_S \leftarrow [\ ]$ array of soft literals in φ
2: $\delta_H \leftarrow [\ ]$ array of hard clauses in φ
3: $E_1 \leftarrow newEvalMaxSAT()$
4: **for** l in δ_S **do**
5: $\quad E_1 \leftarrow l$
6: **end for**
7: **for** l in δ_H **do**
8: $\quad E_1 \leftarrow l$
9: **end for**
10: $resultE_1 \leftarrow E_1\ .\ solve()$
11: **if** $resultE_1$ is a satisfying assignment **then**
12: $\quad E_2 \leftarrow new\ EvalMaxSAT(*E_1)$
13: $\quad E_2\ .\ add\ assumptions()$
14: $\quad E_2\ .\ solve()$
15: **else**
16: $\quad$ terminate program as no solution can be found
17: **end if**

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

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

iMaxHS in MaxSAT Evaluation 2022

Andreas Niskanen, Jeremias Berg, Matti Järvisalo
HIIT, Department of Computer Science
University of Helsinki
Helsinki, Finland
andreas.niskanen, jeremias.berg, matti.jarvisalo@helsinki.fi

***Abstract*—We shortly describe iMaxHS–an incremental version of the implicit hitting set based MaxSAT solver MaxHS–participating in the incremental track of MaxSAT Evaluation 2022.**

***Index Terms*—maximum satisfiability, incremental solving**

I. Introduction

iMaxHS is an incremental version of the implicit hitting set (IHS) based solver MaxHS. The MaxHS algorithm iteratively uses a SAT solver to extract cores and an IP solver to compute minimum-cost hitting sets over the collection of cores [1]–[3], employing various additional techniques (e.g. [4], [5]). The incremental version iMaxHS is built using the MSE 2021 version of MaxHS, and implements all IPAMIR functionality [6], [7].

II. Incremental Solving

For an extensive description and details on how incrementality is enabled in iMaxHS, we refer the reader to [6], [7].

To summarize, before calling `ipamir_solve` all hard clauses added via `ipamir_add_hard` and soft literals declared using `ipamir_add_soft_lit` operate only on the internal representation of the MaxSAT instance. When `ipamir_solve` is called for the first time, a MaxSAT solver instance containing a SAT solver and an IP solver is constructed using the current MaxSAT instance. Subsequent calls of `ipamir_add_hard` add new hard clauses directly to the internal SAT solver, while `ipamir_add_soft_lit` changes the weights of the objective function in the internal IP solver [6] according to the new soft literal.

User-provided assumptions via `ipamir_assume` are handled via so-called conditional cores [7], which take into account which assumptions were made during core extraction, and by resetting the IP solver at each iteration. Conditional cores are stored in an additional SAT solver, which is used only as a unit propagation engine.

We note that enabling additional techniques beyond IHS-based solving [4], [5] require more care in the incremental setting [7]. In addition, iMaxHS performs several simplification procedures to the original MaxSAT instance before initializing the solver instance, and due to this the implementation of IPAMIR functions requires additional data structures, and some of the procedures have to be disabled [6], [7].

Work supported by Academy of Finland under grants 322869 and 342145.

III. Implementation

iMaxHS includes CaDiCaL[1] as the SAT solver and IBM ILOG CPLEX (version 22.1) as the IP solver. For the evaluation, we use the default configuration from the experiments in [7], enabling conditional cores and a separate core initialization phase in which an instance is first solved without assumptions for 100 seconds to allow for extracting standard cores. All IPAMIR functions apart from `ipamir_set_terminate` are supported by this version of iMaxHS. The only restriction is that only one instance of iMaxHS initialized via `ipamir_init` may exist at the same time.

IV. Availability

iMaxHS is available online at https://bitbucket.org/coreo-group/incremental-maxhs under an open-source license.

References

[1] J. Davies and F. Bacchus, "Solving MAXSAT by solving a sequence of simpler SAT instances," in *Proc. CP 2011*, ser. Lecture Notes in Computer Science, vol. 6876. Springer, 2011, pp. 225–239.
[2] ——, "Exploiting the power of MIP solvers in MAXSAT," in *Proc. SAT 2013*, ser. Lecture Notes in Computer Science, vol. 7962. Springer, 2013, pp. 166–181.
[3] ——, "Postponing optimization to speed up MAXSAT solving," in *Proc. CP 2013*, ser. Lecture Notes in Computer Science, vol. 8124. Springer, 2013, pp. 247–262.
[4] F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko, "Reduced cost fixing in MaxSAT," in *Proc. CP 2017*, ser. Lecture Notes in Computer Science, vol. 10416. Springer, 2017, pp. 641–651.
[5] J. Berg, F. Bacchus, and A. Poole, "Abstract cores in implicit hitting set MaxSat solving," in *Proc. SAT 2020*, ser. Lecture Notes in Computer Science, vol. 12178. Springer, 2020, pp. 277–294.
[6] A. Niskanen, J. Berg, and M. Järvisalo, "Enabling incrementality in the implicit hitting set approach to MaxSAT under changing weights," in *Proc. CP 2021*, ser. LIPIcs, vol. 210. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 44:1–44:19.
[7] ——, "Incremental maximum satisfiability," in *Proc. SAT 2022*, ser. LIPIcs, vol. 236. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 14:1–14:19, to appear.

[1] https://github.com/arminbiere/cadical

BENCHMARKS

AdaBoost for Learning Boosted Decision Trees via Incremental MaxSAT

Andreas Niskanen, Jeremias Berg, Matti Järvisalo
HIIT, Department of Computer Science
University of Helsinki
Helsinki, Finland
andreas.niskanen, jeremias.berg, matti.jarvisalo@helsinki.fi

***Abstract*—We shortly present a benchmark for the incremental track of MaxSAT Evaluation 2022. The benchmark application implements AdaBoost, an algorithm for learning boosted decision trees.**

***Index Terms*—decision trees, AdaBoost**

I. Introduction

This benchmark implements a recently-proposed method of learning boosted decision trees via MaxSAT [1] by employing incremental MaxSAT solving [2]. The implemented algorithm is AdaBoost, a well-known ensemble method where multiple weak classifiers are learned and combined into a single classifier. Briefly put, in the benchmark application a MaxSAT instance is first constructed using a MaxSAT encoding for learning the most accurate decision tree with a given maximum depth [1]. This encoding builds on an earlier SAT encoding [3]. Then, weights of soft clauses are iteratively changed (using `ipamir_add_soft_lit`), intuitively giving more priority to misclassified examples and less prority to correctly classified examples.

II. Encoding

We overview the MaxSAT encoding for learning a decision tree of bounded depth which minimizes the training error [1], [3]. The input is a dataset $(\mathbf{X}_i, y_i)$, $i = 1, \ldots, n$ of binary examples $\mathbf{X}_i \in \{0,1\}^m$ and classes $y_i \in \{0,1\}$. Each coordinate $j = 1, \ldots, m$ of an example $\mathbf{X}_i = (x_i^1, \ldots, x_i^m)$ is called a feature. A decision tree is a binary classifier mapping each example in $\{0,1\}^m$ to a class in $\{0,1\}$ with the structure of a binary tree where leaf nodes correspond to classes and non-leaf nodes to features. In the MaxSAT encoding, hard clauses are used to encode the structure of a valid binary tree of depth at most D, the assignment of features to non-leaf nodes, and the classification result of every input example. In particular, variables b_i for $i = 1, \ldots, n$ are used to encode that example $\mathbf{X}_i$ is classified correctly. The training error is then minimized via soft clauses (b_i) for each $i = 1, \ldots, n$ with unit weights $w(b_i) = 1$.

III. AdaBoost Algorithm

The MaxSAT-based AdaBoost algorithm initializes a solver with the MaxSAT encoding and iteratively changes weights of soft clauses [1]. In more detail, after learning a decision tree via a MaxSAT solver call, we compute the training error ε of that tree, and set $\alpha = \frac{1}{2}\ln(\frac{1-\varepsilon}{\varepsilon})$. Then, the weight $w(b_i)$ of each soft clause b_i is updated via

$$\widehat{w}(b_i) = \frac{w(b_i) f_i}{\sum_{j=1}^{n} w(b_j) f_j}$$

where $f_i = \exp(-\alpha)$ if the ith example was classified correctly, and $f_i = \exp(\alpha)$ otherwise. Finally, weights are discretized via

$$w(b_i) = \text{round}\left(\frac{\widehat{w}(b_i)}{\min_{j=1}^{n} \widehat{w}(b_i)}\right).$$

IV. Benchmark Instances

For benchmark instances, we used all 15 datasets from [1], which were downloaded from CP4IM[1] and discretized[2]. For each dataset, we generated different training sets by taking 10%, 20%, ..., 90% of the available examples, and repeated this 5 times for each percentage. This resulted in 675 input datasets. We set the maximum depth to $D = 2$ and use 20 iterations for AdaBoost in the implementation.

References

[1] H. Hu, M. Siala, E. Hebrard, and M. Huguet, "Learning optimal decision trees with maxsat and its integration in adaboost," in *Proc. IJCAI 2020*. ijcai.org, 2020, pp. 1170–1176.

[2] A. Niskanen, J. Berg, and M. Järvisalo, "Enabling incrementality in the implicit hitting set approach to MaxSAT under changing weights," in *Proc. CP 2021*, ser. LIPIcs, vol. 210. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 44:1–44:19.

[3] N. Narodytska, A. Ignatiev, F. Pereira, and J. Marques-Silva, "Learning optimal decision trees with SAT," in *Proc IJCAI 2018*. ijcai.org, 2018, pp. 1362–1368.

Work supported by Academy of Finland under grants 322869 and 342145.

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

[2] https://gepgitlab.laas.fr/hhu/maxsat-decision-trees

MaxSAT Evaluation 2022: Solver and Benchmark Descriptions, volume B-2022-2 of Department of Computer Science Series of Publications B, University of Helsinki 2022.

BIOPTSAT and MLIC-SEESAW Benchmarks for the Incremental Track of MaxSAT Evaluation 2022

Christoph Jabs, Jeremias Berg, Andreas Niskanen, Matti Järvisalo
HIIT, Department of Computer Science
University of Helsinki
Helsinki, Finland
{christoph.jabs, jeremias.berg, andreas.niskanen, matti.jarvisalo}@helsinki.fi

***Abstract*—We describe the BIOPTSAT and MLIC-SEESAW incremental MaxSAT benchmarks submitted to MaxSAT Evaluation 2022. The benchmarks are based on the BIOPTSAT and Seesaw algorithms for bi-objective optimization and modify them to use a MaxSAT solver instance each. In addition to the algorithms making up the benchmarks we describe the provided instance files that go with the benchmark applications.**

***Index Terms*—SAT, MaxSAT, Bi-Objective Optimization**

I. INTRODUCTION

Bi-objective optimization problems arise naturally in many applications (e.g., [1]–[3]). Often, instead of treating the two objectives separately, a linear combination of the objectives is formed [4]. However, with this approach a value of importance has to be assigned to the two objectives *before* solving and even when solving with a wide range of these values, there is no guarantee that every "optimal" solution for the two objectives is found.

To mitigate the restriction of having to specify a preference over the objectives beforehand, algorithms solving bi-objective optimization under so-called Pareto optimality [5] have been developed. Recently, two SAT-based algorithms that can solve bi-objective optimization problems under Pareto optimality have been proposed. BIOPTSAT [6] follows the lexicographic method and originally makes use of a single incremental SAT solver. Seesaw [3] is an extension of the implicit hitting set framework, making it applicable to bi-objective optimization problems. We present two sets of benchmarks for the incremental track of the MaxSAT evaluation 2022 based on these two algorithms. While both algorithms can be modified to make use of *two* instances of a given IPAMIR MaxSAT solver each, since the initial version of IPAMIR did not explicitly specify that multiple solver instances can be created at the same time, we opt to only use *one* solver instance per benchmark.

II. PRELIMINARIES

First, we formalize the bi-objective Boolean optimization problem and define Pareto optimality. Second, we briefly survey the totalizer encoding for cardinality constraints, since it is used in one of the benchmarks. Finally, we define notation for a MaxSAT solver in pseudocode.

Work financially supported by Academy of Finland under grants 322869, 328718 and 342145.

A. Bi-Objective Boolean Optimization

A bi-objective Boolean optimization problem consists of three components: a CNF formula F describing the set of feasible solutions and two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$ that should be minimized. We denote the set of all literals in F as $\mathrm{LIT}(F)$. An objective O is a multiset of literals, which allows for representing objective functions with non-unit coefficients. The value $\mathrm{O}(\tau)$ of a truth assignment τ under O is $\mathrm{O}(\tau) = \sum_{l \in \mathrm{O}} \tau(l)$, i.e., the number of the literals in O that τ assigns to 1. Integer weighted objectives can be represented by adding a literal multiple times. Since both BIOPTSAT and Seesaw enumerate the Pareto-optimal solutions in an ordered fashion with the values for one objective increasing while the values for the other objective are decreasing, we call $\mathrm{O_I}$ *increasing* and $\mathrm{O_D}$ *decreasing*.

Given a CNF formula F, two objectives $\mathrm{O_I}, \mathrm{O_D} \subset \mathrm{LIT}(F)$ and solutions τ_1, τ_2 to F, we say that τ_1 dominates τ_2 if (i) $\mathrm{O}_i(\tau_1) \leq \mathrm{O}_i(\tau_2)$ for $i \in \{\mathrm{I}, \mathrm{D}\}$, and (ii) either $\mathrm{O_I}(\tau_1) < \mathrm{O_I}(\tau_2)$ or $\mathrm{O_D}(\tau_1) < \mathrm{O_D}(\tau_2)$. A solution τ is Pareto-optimal if no other solution dominates it. When the objectives are clear from context, we will simply say that a solution τ is a Pareto-optimal solution of F. The pair $(\mathrm{O_I}(\tau), \mathrm{O_D}(\tau))$ of a Pareto-optimal τ is a Pareto point (of F wrt $\mathrm{O_I}$ and $\mathrm{O_D}$). The given benchmarks solve the problem of finding the set of all Pareto points, but the algorithms can be extended to report the Pareto-optimal solutions (see implementation of [6]).

B. The Totalizer Encoding

Given a set L of n input literals, the (incremental) totalizer [7], [8] encoding produces a CNF formula $\mathrm{TOT}(L)$ that defines a set $\{\langle L \leq 0\rangle, \ldots, \langle L \leq |L|\rangle\} \subset \mathrm{LIT}(\mathrm{TOT}(L))$ of *output literals* that—informally speaking—count the number of literals in L assigned to true by solutions to $\mathrm{TOT}(L)$: If τ is an assignment that satisfies $\mathrm{TOT}(L)$, then $\tau(\langle L \leq b\rangle) = 1$ if $\sum_{l \in L} \tau(l) \leq b$. The incremental totalizer supports building the totalizer only to a certain upper bound and extending this bound later on. The benchmarks build the totalizers incrementally.

C. MaxSAT Solver Notation

We denote the use of a MaxSAT solver in the pseudocode of the benchmarks with the following three subroutines: `MaxSATInit` initializes a new MaxSAT solver instance. It

Algorithm 1 IPAMIR BIOPTSAT

Input: A formula F, two objectives $\mathrm{O_I}$ and $\mathrm{O_D}$.
Output: All Pareto points of F wrt the two objectives.

1: $s_\mathrm{I} \leftarrow \texttt{MaxSATInit}(F \wedge \textsc{Tot}(\mathrm{O_D}), \mathrm{O_I})$
2: $s_\mathrm{D} \leftarrow \texttt{SATInit}(F \wedge \textsc{Tot}(\mathrm{O_I}) \wedge \mathrm{O_D})$
3: $\texttt{res}, b_\mathrm{I} \leftarrow \texttt{MaxSATSolve}(s_\mathrm{I}, \emptyset)$
4: **while** $\texttt{res} = \texttt{OPT}$ **do**
5: $\quad \texttt{res}, b_\mathrm{D} \leftarrow \texttt{SolImpr}(s_\mathrm{D}, \mathrm{O_D}, \{\langle \mathrm{O_I} \leq b_\mathrm{I} \rangle\})$
6: $\quad$ **yield** $(b_\mathrm{I}, b_\mathrm{D})$
7: $\quad \texttt{MaxSATAddClause}(s_\mathrm{I}, \langle \mathrm{O_D} \leq b_\mathrm{D} - 1 \rangle)$
8: $\quad \texttt{res}, b_\mathrm{I} \leftarrow \texttt{MaxSATSolve}(s_\mathrm{I}, \emptyset)$

takes a formula of hard clauses and an objective as a multiset of soft literals as parameters and returns a handle to the initialized solver. `MaxSATSolve` queries the solver specified by the handle as the first parameter for an optimal solution and returns the pair $(\texttt{res}, o)$, where `res` is either `OPT` if an optimal solution was found or `UNSAT` if the hard clauses are not satisfiable. If $\texttt{res} = \texttt{OPT}$, o is the optimal objective value found. As a second parameter, `MaxSATSolve` takes a set of assumptions. `MaxSATAddClause` allows for adding new hard clauses to an already initialized solver. The parameters are the solver handle and the clause to add.

III. THE BIOPTSAT BENCHMARK

The BIOPTSAT benchmark is a more naive version of the BIOPTSAT algorithm [6]. Other than the original algorithm, it does not use a single incremental SAT solver but one IPAMIR MaxSAT solver and solution improving search in a separate IPASIR SAT solver. The benchmark can solve any Boolean bi-objective optimization problems in the BICNF format that is similar to DIMACS WCNF.

A. Description of the Algorithm

The modified BIOPTSAT algorithm using two incremental MaxSAT solvers is described in Algorithm 1. It initializes one MaxSAT solvers on Line 1 and one SAT solver on Line 2. The MaxSAT solver is initialized with $\mathrm{O_I}$ as its objective function and F plus a totalizer over $\mathrm{O_D}$ as its hard clauses. In the MaxSAT solver, the totalizer is used to enforce bounds on the objective that the solver was not initialized with when making MaxSAT calls. The SAT solver is passed F plus a totalizer over $\mathrm{O_I}$ and $\mathrm{O_D}$ each as its clauses. This SAT solver is used in the `SolImpr` subroutine to perform solution-improving search over $\mathrm{O_D}$ while enforcing bounds on $\mathrm{O_I}$.

The BIOPTSAT benchmark enumerates all Pareto points by first minimizing the increasing objective under the constraint that the decreasing objective value is smaller than for the previously found Pareto point. After that, the decreasing objective is minimized as solution improving search without making the increasing objective worse than the optimal value found in the last MaxSAT call. Once no new Pareto point with a smaller objective value for the decreasing objective exists (i.e., the MaxSAT call on Line 8 returns `UNSAT`), the algorithm terminates.

B. Description of the Provided Instances

The instances for this benchmark are in a file format similar to the standard WCNF format for main track benchmarks. Lines starting with `h` denote hard clauses and lines starting with `1` or `2` denote soft literals for the increasing, respectively decreasing objective. The soft literal lines consist of the weight and the literal, terminated by a `0`.

There are four types of instances included with this benchmark. `small.bicnf` and `medium.bicnf` are two manually created instances for debugging purposes. The other three types are contained in the directories `mlic` and `set-cover`.

Instances in the `mlic` directory are binary datasets encoded for learning interpretable decision rules with the MLIC encoding [1]. The increasing objective for these instances consists of the b variables of the encoding, the decreasing objective of the η variables. This way, the increasing objective is the rule size and the decreasing objective its classification error. The datasets included were downloaded from the UCI Machine Learning Repository [9] and from Kaggle (https://www.kaggle.com). In addition to encoding the full datasets, we randomly and independently sampled subsets of `n_samp` $\in \{50, 100, 1000, 5000, 10000\}$ data samples, four of each size (when applicable). The instances encoded from subsets are named as `<dataset>_<n_samp>_<idx>.bicnf`, where `idx` is the index from 1 to 4. For more details on the datasets, see [6] and the corresponding repository.

In the `set-cover` directory, we provide bi-objective set covering instances generated with two different random procedures. The instances `fixed-element-prob-<n_elem>-<n_sets>-<p>-<idx>.bicnf` are generated with a process where each element has a probability of `p` of appearing in each set. With this, the cardinality of the sets differs. For the instances `fixed-set-card-<n_elem>-<n_sets>-<c_set>-<idx>.bicnf`, every set has fixed cardinality `c_set` and the elements in the sets are chosen uniformly at random. For all instances, the number of elements is `n_elem` and the number of sets `n_sets`. The two objective weights for each element are chosen uniformly at random from 1 to 100. There are five instances for every parameter combination in `n_elem` $\in \{100, 150, 200\}$, `n_sets` $\in \{20, 40, 60, 80\}$, `p` $\in \{0.1, 0.2\}$ and `c_set` $\in \{5, 10\}$.

IV. THE MLIC-SEESAW BENCHMARK

The MLIC-SEESAW benchmark is an instantiation of the Seesaw [3] framework, which generalizes the implicit hitting set framework to bi-objective problems. It solves the bi-objective optimization problem of finding Pareto-optimal interpretable classification decision rules for binary datasets. The problems are encoded with the MLIC encoding [1] and the two objectives are the size of the decision rule and its classification error.

A. Description of the Algorithm

The Seesaw framework, as presented in [3], requires the instantiation of four main components: the used universe, cost function, oracle function and core extraction strategy.

Algorithm 2 MLIC-SEESAW: Seesaw on the MLIC encoding

Input: A binary dataset $\mathcal{D}$.
Output: All points (f, g) for which a Pareto-optimal decision rule for $\mathcal{D}$ with size f a classification error g exists.

1: $F, \mathrm{O_I}, \mathrm{O_D} \leftarrow \texttt{encodeMLIC}(\mathcal{D})$
2: $s_{\text{cost}} \leftarrow \texttt{CplexInitHS}(\emptyset, \mathrm{O_I})$
3: $s_{\text{orac}} \leftarrow \texttt{MaxSATInit}(F, \mathrm{O_D})$
4: $g_{\text{best}}, f_{\text{best}} \leftarrow |\mathrm{O_I}| + 1, |\mathrm{O_I}| + 1$
5: $\texttt{res}, g, \texttt{hs} \leftarrow \texttt{CplexCompHS}(s_{\text{cost}})$
6: **while** $\texttt{res} = \texttt{OPT}$ **do**
7: **if** $g > g_{\text{best}}$ **then**
8: **yield** $(f_{\text{best}}, g_{\text{last}})$
9: $g_{\text{best}} \leftarrow |\mathrm{O_I}| + 1$
10: $\texttt{res}, f \leftarrow \texttt{MaxSATSolve}(s_{\text{orac}}, \{\neg\eta \mid \eta \in \mathrm{O_I} \setminus \texttt{hs}\})$
11: **if** $\texttt{res} = \text{OPT} \wedge f < f_{\text{best}}$ **then**
12: $f_{\text{best}}, g_{\text{best}} \leftarrow f, g$
13: $\kappa \leftarrow \texttt{extractCore}_{\text{am}}(\texttt{hs}, f, s_{\text{orac}})$
14: $\texttt{CplexAddCoreHS}(s_{\text{cost}}, \kappa)$
15: $\texttt{res}, g, \texttt{hs} \leftarrow \texttt{CplexCompHS}(s_{\text{cost}})$

In the benchmark, the universe is the set of η_i variables from the MLIC encoding, representing whether sample i is allowed to be incorrectly classified. The cost function is the cardinality of the hitting sets (i.e., subsets of the universe) making it the classification error of the decision rule. The oracle function computes the size of the smallest decision rule misclassifying no samples that are not in the hitting set. This oracle function is anti-monotonic, therefore, the improved core-extraction strategy presented in [3] can be used. In the benchmark computing the minimum hitting sets is done via an integer linear program in CPLEX and the oracle function is computed as a MaxSAT solver call.

Algorithm 2 shows the pseudocode of the benchmark. On Line 1, the dataset is encoded with the MLIC encoding [1]. The hard clauses are returned as F, the set of η variables as $\mathrm{O_I}$ and the set of b variables as $\mathrm{O_D}$. Next, a CPLEX hitting set solver and the oracle function are initialized on Lines 2 and 3. In the main loop, the oracle function under the condition of a current hitting set (defined by `hs`) is computed on Line 10. Next, if the found solution is better than the last, it is saved. Before a new hitting set is found on Line 15, a core is extracted and added to CPLEX on Lines 13f. The core extraction subroutine $\texttt{extractCore}_{\text{am}}$ is as defined in [3] and makes use of the oracle MaxSAT solver.

B. Description of the Provided Instances

This benchmark directly reads `.csv` files with "`;`" as the separator. It assumes that the first row is a header and that every column contains one binary feature, except for the last column, which contains the binary class label. We again provide the same full datasets and subsets with the same naming scheme as in Section III-B, but not encoded yet. For more details on the datasets, see [6] and the corresponding repository.

REFERENCES

[1] D. Malioutov and K. S. Meel, "MLIC: A MaxSAT-based framework for learning interpretable classification rules," in *Principles and Practice of Constraint Programming — 24th International Conference, CP 2018, Lille, France, August 27–31, 2018, Proceedings* (J. N. Hooker, ed.), vol. 11008 of *Lecture Notes in Computer Science*, pp. 312–327, Springer, 2018.

[2] 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 2018, July 13–19, 2018, Stockholm, Sweden* (J. Lang, ed.), pp. 1362–1368, ijcai.org, 2018.

[3] M. Janota, A. Morgado, J. F. Santos, and V. M. Manquinho, "The Seesaw algorithm: Function optimization using implicit hitting sets," in *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25–29, 2021* (L. D. Michel, ed.), vol. 210 of *LIPIcs*, pp. 31:1–31:16, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[4] M. Ehrgott, "The weighted sum method and related topics," in *Multicriteria Optimization (2. ed.)*, ch. 3, pp. 65–95, Springer, 2005.

[5] M. Ehrgott, "Efficiency and nondominance," in *Multicriteria Optimization (2. ed.)*, ch. 2, pp. 22–64, Springer, 2005.

[6] C. Jabs, J. Berg, A. Niskanen, and M. Järvisalo, "MaxSAT-based bi-objective boolean optimization," in *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022), Haifa, Israel, August 2–5, 2022* (K. S. Meel and O. Strichman, eds.), vol. 236 of *LIPIcs*, pp. 12:1–12:23, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[7] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming — CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 – October 3, 2003, Proceedings* (F. Rossi, ed.), vol. 2833 of *Lecture Notes in Computer Science*, pp. 108–122, Springer, 2003.

[8] R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce, "Incremental cardinality constraints for MaxSAT," in *Principles and Practice of Constraint Programming — 20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings* (B. O'Sullivan, ed.), vol. 8656 of *Lecture Notes in Computer Science*, pp. 531–548, Springer, 2014.

[9] D. Dua and C. Graff, "UCI machine learning repository," 2021.

CEGAR for Extension Enforcement in Abstract Argumentation via Incremental MaxSAT

Andreas Niskanen, Matti Järvisalo
HIIT, Department of Computer Science
University of Helsinki
Helsinki, Finland
andreas.niskanen, matti.jarvisalo@helsinki.fi

***Abstract*—We shortly present a benchmark for the incremental track of MaxSAT Evaluation 2022. The benchmark application implements a MaxSAT-based counterexample-guided abstraction refinement (CEGAR) algorithm for extension enforcement under preferred semantics in abstract argumentation, a problem which is hard for the second level of the polynomial hierarchy.**

***Index Terms*—abstract argumentation, extension enforcement, CEGAR**

I. Introduction

In abstract argumentation [1] a scenario is represented using a directed graph with nodes as arguments and edges as attacks between arguments, with so-called semantics identifying extensions, i.e., jointly acceptable subsets of arguments. Enforcement is a central problem concerning the dynamics of abstract argumentation with several different variants [2]. The goal of (strict) extension enforcement is to modify an argumentation framework to one where a given set of arguments is an extension [3], which is seen as an optimization problem by additionally minimizing the number of changes to an initial argumentation framework [4].

Here we consider extension enforcement under preferred semantics. The corresponding decision problem is known to be complete for the second level of the polynomial hierarchy [5], which means that presumably there is no compact (polynomial-sized) MaxSAT encoding for the optimization problem. This benchmark implements a MaxSAT-based CEGAR algorithm [5] for this problem using so-called strong refinements [6]. In the benchmark application a MaxSAT instance is first constructed using a MaxSAT encoding for extension enforcement under *complete semantics*. Iteratively, candidate solutions are obtained from the MaxSAT solver. An additional SAT solver is used to yield counterexamples from the candidate solution. If a counterexample is found, it is used to construct a hard clause which rules out non-solutions, and is added to the MaxSAT solver. The algorithm proceeds until no counterexample is found, i.e., the candidate solution is an actual solution.

II. Encoding

The input is an AF $F = (A, R)$ where A is the set of arguments and $R \subseteq A \times A$ is the attack relation, and a set $T \subseteq A$ The MaxSAT encoding uses variables $r_{a,b}$ for each $a, b \in A$ to represent a solution argumentation framework $F = (A, R')$ where T is a complete extension, interpreting $\tau(r_{a,b}) = 1$ iff attack $(a, b) \in R'$. For details on the encoding, we refer the reader to [5].

III. CEGAR Algorithm

After obtaining a candidate argumentation framework $F' = (A, R')$ where T is a complete extension, we use a SAT solver to check whether there is another complete extension $T' \supset T$ in F'. If there is, T is not a preferred extension. That is, T' is a counterexample in F', and is used to construct a strong refinement clause [6] which is added to the MaxSAT solver, ruling out F' and other non-solutions. This procedure is repeated until we obtain a candidate solution with no counterexample extension.

IV. Benchmark Instances

For generating benchmark instances, we follow [6]. We used all instances from the ICCMA 2019[1] argumentation solver competition with at most 500 arguments. For each of these 221 AFs, we generated five extension enforcement instances for each $|T|/|A| = 0.025, 0.05, 0.075, 0.1, 0.2, 0.3$ by picking $|T|$ uniformly at random, resulting in a total of 6630 instances. In the implementation, we use MiniSat (version 2.2.0) as the SAT solver for the counterexample check.

References

[1] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artif. Intell.*, vol. 77, no. 2, pp. 321–358, 1995.

[2] R. Baumann, S. Doutre, J. Mailly, and J. P. Wallner, "Enforcement in formal argumentation," *FLAP*, vol. 8, no. 6, pp. 1623–1678, 2021. [Online]. Available: https://collegepublications.co.uk/ifcolog/?00048

[3] R. Baumann and G. Brewka, "Expanding argumentation frameworks: Enforcing and monotonicity results," in *Proc. COMMA 2010*, ser. Frontiers in Artificial Intelligence and Applications, vol. 216. IOS Press, 2010, pp. 75–86.

[4] S. Coste-Marquis, S. Konieczny, J. Mailly, and P. Marquis, "Extension enforcement in abstract argumentation as an optimization problem," in *Proc. IJCAI 2015*. AAAI Press, 2015, pp. 2876–2882.

[5] J. P. Wallner, A. Niskanen, and M. Järvisalo, "Complexity results and algorithms for extension enforcement in abstract argumentation," *J. Artif. Intell. Res.*, vol. 60, pp. 1–40, 2017.

[6] A. Niskanen and M. Järvisalo, "Strong refinements for hard problems in argumentation dynamics," in *Proc. ECAI 2020*, ser. Frontiers in Artificial Intelligence and Applications, vol. 325. IOS Press, 2020, pp. 841–848.

Work supported by Academy of Finland under grants 322869 and 342145.

[1] https://www.iccma2019.dmi.unipg.it

Incremental Partial MaxSAT Application: Generalization of Proof Obligations in Bit-Level PDR

Tobias Seufert[1], Tobias Paxian[1], Felix Winterer[1],
Christoph Scholl[1], Karsten Scheibler[2], Armin Biere[1], Bernd Becker[1]
[1]Institute of Computer Science, University of Freiburg, Germany
{seufert, paxiant, winteref, scholl, biere, becker}@informatik.uni-freiburg.de
[2]BTC Embedded Systems AG, Germany, scheibler@btc-es.de

***Abstract*—We present a set of incremental MaxSAT benchmarks for MaxSAT Evaluation 2022. The problem instances stem from applying Property Directed Reachability (PDR / IC3 [1], [2]) to sequential circuits given in AIGER format [3]. In particular, the MaxSAT solver is used for generalizing proof obligations (or CTI) in PDR.**

I. Introduction

In 2011, the verification engine PDR resp. IC3 was introduced [1] and is nowadays widely considered as the most powerful algorithm for Hardware Model Checking. The idea of PDR is to avoid the unrolling of the transition relation as in Bounded Model Checking (BMC) [4] and to rather replace small numbers of large and hard SAT problems by many small and easy ones based on a single instance of the transition relation only. PDR repeatedly strengthens a proof by removing unreachable predecessors of unsafe states. Thereby, PDR tries to avoid enumerating single states, but puts a lot of effort into generalizing these predecessors – so called *proof obligations* – to preferably expressive state sets. PDR's efficiency relies heavily on an effective generalization of proof obligations [2], [5].

In [6] we discuss various techniques which we adapt to the generalization of proof obligations in PDR. One of these, we call it *MS01X*, is based on the use of a MaxSAT solver. Our recent experimental analysis in [6] indicates, that there is lots of potential in MaxSAT based generalization in PDR. In combination with another technique, *MS01X* was the strongest method in our evaluation, despite the fact that with Pacose [7] we were using a *non-incremental* MaxSAT solver for a task which is obviously predestined for an incremental application (as PDR is in general [2]).

II. Basics and Notations

In a finite state transition system we have a finite set of states and a transition relation which encodes transitions between states under certain inputs. States are obtained by assigning Boolean values to the (present) state variables $\vec{s} = (s_1, \ldots, s_m)$, inputs by assigning Boolean values to the input variables $\vec{i} = (i_1, \ldots, i_n)$. For representing transitions we introduce a second copy $\vec{s'}$ of the state variables, the so-called next state variables. The transition relation is then represented by a predicate $T(\vec{s}, \vec{i}, \vec{s'})$, the set of initial states by a predicate $I(\vec{s})$. The set of bad states (we verify invariant properties) are represented by a predicate $\neg P(\vec{s})$. For brevity, we often omit the arguments of the predicates and write them without parenthesis. A *literal* represents a Boolean variable or its negation. *Cubes* are conjunctions of literals, *clauses* are disjunctions of literals. The negation of a cube is a clause and vice versa. A Boolean formula in *Conjunctive Normal Form* (CNF) is a conjunction of clauses.

III. Generalization of Proof Obligations

We restrict ourselves to one particular part of PDR – the generalization of proof obligations. We remark, that in the following we assume that the (usually tseitin-transformed [8]) CNF representing the transition relation is a *function*[1]. For a more detailed description of PDR, we refer to [1], [2].

Proof obligations are created as predecessors of a particular set of bad states[2] represented by a cube (conjunction of literals) d following a satisfiable SAT solver call to the formula

$$F_i \wedge T \wedge d' \tag{1}$$

whereas F_i is representing some overapproximation of reachable states in which we search for the predecessor of d.

We assume that the SAT solver yields us a full satisfying assignment m over the state variables – m is now considered a *proof obligation*. Our objective is, to generalize m such that we only add states which are exact predecessors of the bad states d. The most standard technique which is usually applied here, is *greedy 01X-simulation* with ternary[3] logic: We set one literal of m to X, simulate the circuit (origin of T), and check whether the X propagates to the next state variables of d'. If it does, we retract X and probe the next literal. If the X percolates, we may remove the respective literal l from m and therefore generalize m to $\hat{m} = m \setminus \{l\}$.

[1]This is obviously the case if we consider sequential circuits.
[2]Violating the invariant property, or being predecessors of such states.
[3]We extend the two valued semantics by $(X \wedge 0 = 0), (X \wedge 1 = X), (X \wedge X = X), (\neg X = X)$

IV. 01X-ENCODING

The aforementioned greedy 01X-simulation solves the underlying optimization problem only heuristically. Therefore, we introduce a partial MaxSAT [9]–[11] encoding to find a better approximate[4] solution. To 01X-encode the transition relation resulting from a Boolean circuit (and therefore representing a function), we introduce *two* variables $v^{(0)}$ and $v^{(1)}$ for each Boolean variable v which represents either an input, an output or an internal signal, while $((v^{(0)} = 0 \wedge v^{(1)} = 0) \leftrightarrow v = X)$ as well as $((v^{(0)} = 1 \wedge v^{(1)} = 0) \leftrightarrow v = 0)$ and $((v^{(0)} = 0 \wedge v^{(1)} = 1) \leftrightarrow v = 1)$; we explicitly forbid $(v^{(0)} = 1 \wedge v^{(1)} = 1)$. All gates are replaced by a two-rail encoding according to [12].

For each state variable s_i we introduce a new variable t_i and a unit soft clause $sc_i = \{t_i\}$ accompanied by the hard clauses representing $t_i \leftrightarrow ((s_i^{(0)} = 0) \wedge (s_i^{(1)} = 0))$. Starting with a satisfying solution to Eqn. 1 with $d' = ((d'_{i_1})^{\tau_1} \wedge \ldots \wedge (d'_{i_k})^{\tau_k})$[5] that provides full assignments $m = s_1^{\sigma_1} \wedge \ldots \wedge s_m^{\sigma_m}$ and $i = i_1^{\iota_1} \wedge \ldots \wedge i_n^{\iota_n}$, we introduce hard clauses fixing state bits s_i to X or σ_i, input bits i_j to ι_j, and next state bits d'_{i_j} to τ_j.

The other hard clauses of the considered MaxSAT problem correspond to the 01X-encoding of T. Maximizing the number of satisfied soft clauses means maximizing the number of present state bits which are assigned to X and are thus not included in the resulting c of m from which all transitions under i lead into d'. As already mentioned before we call the resulting MaxSAT problem *MS01X*.

V. BENCHMARK DESCRIPTION

We provide the publicly available benchmarks from Hardware Model Checking Competitions (HWMCC) 2015 and 2017 [13], [14]. The benchmarks are designs and specifications of sequential circuits represented in the AIGER format [3].

The benchmarks are verified using our tool which is derived from IC3ref [15]. Usually, the generalization of proof obligations and therefore the MaxSAT solver is invoked from ten to tens of thousands times, whereas the transition relation T remains the same and only the proof obligation cube m as well as the bad cube d' are altered (and provided to the solver via assumptions). This means the given benchmarks consist of a core of hard clauses to which we add in every round only hard clause assumptions. These assumptions will be slightly modified in each loop. The set of soft clauses remains the same in all iterations.

Our tool reports 0 if the design is *safe* with respect to the property, or 1 if the design is *unsafe*.

The resulting binary is called `IC3` and the HWMCC benchmarks in AIGER format are ended with `*.aig`.

[4]Even though MaxSAT is optimal, the solution is still approximate due to the shortcomings of 01X-encodings.

[5]A cube $c = s_{i_1}^{\sigma_1} \wedge \ldots \wedge s_{i_k}^{\sigma_k}$ of literals over state variables with $i_j \in \{1, \ldots, m\}$, $\sigma_j \in \{0, 1\}$, $s_{i_j}^0 = \neg s_{i_j}$ and $s_{i_j}^1 = s_{i_j}$ represents the set of all states where s_{i_j} is assigned to σ_j for all $j = 1, \ldots, k$.

REFERENCES

[1] A. R. Bradley, "Sat-based model checking without unrolling," in *VMCAI*, 2011, pp. 70–87.

[2] N. Eén, A. Mishchenko, and R. K. Brayton, "Efficient implementation of property directed reachability," in *FMCAD*, 2011, pp. 125–134.

[3] A. Biere, "The AIGER And-Inverter Graph (AIG) format version 20071012," Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, Tech. Rep. 07/1, 2007.

[4] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *TACAS*, 1999, pp. 193–207.

[5] A. Griggio and M. Roveri, "Comparing different variants of the ic3 algorithm for hardware model checking," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 6, pp. 1026–1039, 2016.

[6] T. Seufert, F. Winterer, C. Scholl, K. Scheibler, T. Paxian, and B. Becker, "Everything You Always Wanted to Know About Generalization of Proof Obligations in PDR," *arXiv preprint arXiv:2105.09169*, 2021. [Online]. Available: https://arxiv.org/abs/2105.09169

[7] T. Paxian, S. Reimer, and B. Becker, "Dynamic polynomial watchdog encoding for solving weighted maxsat," in *SAT*, 2018, pp. 37–53.

[8] G. Tseitin, "On the Complexity of Derivation in Propositional Calculus," *Studies in Constructive Mathematics and Mathematical Logic*, 1968.

[9] Z. Fu and S. Malik, "On solving the partial max-sat problem," in *International Conference on Theory and Applications of Satisfiability Testing*, 2006, pp. 252–265.

[10] C. M. Li and F. Manya, "Maxsat, hard and soft constraints." *Handbook of satisfiability*, vol. 185, pp. 613–631, 2009.

[11] C. Ansótegui, M. L. Bonet, and J. Levy, "Sat-based maxsat algorithms," *Artificial Intelligence*, vol. 196, pp. 77–105, 2013.

[12] A. Jain, V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and M. S. Hsiao, "Testing, verification, and diagnosis in the presence of unknowns," in *VTS*, 2000, pp. 263–270.

[13] A. Biere and K. Heljanko, "Hardware model checking competition," 2015. [Online]. Available: http://fmv.jku.at/hwmcc15/

[14] A. Biere, T. van Dijk, and K. Heljanko, "Hardware model checking competition," 2017. [Online]. Available: http://fmv.jku.at/hwmcc17/

[15] A. Bradley, "Ic3 reference implementation," 2013. [Online]. Available: https://github.com/arbrad/IC3ref

Solver Index

Benchmark Index

Author Index