Maple_MBDR_Cent_PERM and Maple_MDBR_BJL in the 2021 SAT Solver Competition

Sima Jamali
Simon Fraser University
Vancouver, Canada
sja88@sfu.ca

David Mitchell
Simon Fraser University
Vancouver, Canada
mitchell@cs.sfu.ca

***Abstract*—We give a brief description of our solvers Maple_MBDR_Cent_PERM and Maple_MBDR_BJL. Maple_MBDR_Cent_PERM stores high-centrality learned clauses permanently. Maple_MBDR_BJL, uses a simple and cheap scheme to learn additional small clauses. Both solvers are based on Relaxed_Maple_LCMDCBDL_newtech, the second place solver from the main track of the 2020 SAT Solver Competition.**

***Index Terms*—High-Centrality Clauses, Permanent, Learning**

I. Introduction

Relaxed_Maple_LCMDCBDL_newtech won the silver medal of the main track of the 2020 SAT Solver Competition [3]. This is a recent member of the MapleSAT family of solvers that has been improved over the last 5 years [1], [2], [4], [6], [8], [9]. Our submissions modify the clause maintenance strategy of Relaxed_Maple_LCMDCBDL_newtech, which was inherited from COMiniSatPS and involves three stores of learnt clauses: Core, Tier2 and Local [5], [6]. Learned clauses are placed in one of these stores based on their LBD. Small LBD clauses are placed in Core and retained permanently.

Clause Centrality was introduced in [10] and shown to be a useful clause quality measure. Maple_MBDR_Cent_PERM places high-centrality clauses in Core regardless of their LBD. Small clauses are understood to be valuable, and many solvers store very small clauses permanently. Maple_MBDR_BJL has a simple scheme to learn an additional small clause after each backjump to a small decision level.

II. High-Centrality Permanent clauses

The centrality of a clause is the mean betweenness centrality of its variables. To compute centralities, we generate the primal graph of the input CNF (after pre-processing). The betweenness centrality of a vertex (variable) v is the number of shortest paths between pairs of vertices excluding v, that visit v. It is defined by $g(v) = \sum_{s \neq v \neq t}(\sigma_{s,t}(v)/\sigma_{s,t})$, where $\sigma_{s,t}$ is the number of shortest s-t paths and $\sigma_{s,t}(v)$ is the number of those passing through v, normalized to $[0, 1]$ [11].

This work was supported by the Natural Sciences and Engineering Council of Canada (NSERC) through a Discovery Grant to the second author.

We use Brandes algorithm [12] to compute centrality values. Generating the graph and computing centrality values are memory and time intensive for large formulas, so we compute centralities only for formulas with at most 100,000 clauses after pre-processing. We also limit the time for centrality computation to 150 seconds. We use the base solver without modification for formulas without centralities. For formulas with centralities, high centrality (HC) learned clauses (those with centrality greater than a threshold CT), are stored in Core, regardless of their LBD. We aim to include at least the 0.02% of learned clauses with highest centrality in Core. We set an initial threshold of $CT \geq 0.008$. Every $100,000$ conflicts, if the number of HC clauses in Core is less than 0.02% of all learned clauses, CT is reduced by 0.001, but it is never reduced below 0.004. We submitted two versions:

- Maple_MBDR_Cent_PERM_10K: This solver adds at most the first 10,000 HC clauses to Core.
- Maple_MBDR_Cent_PERM_75K: This solver adds at most the first 75,000K HC clauses to Core.

III. BackJump Learning

Standard conflict analysis schemes derive one clause, called the 1-UIP clause, at each conflict. To Maple_MBDR_BJL, we add a simple and inexpensive scheme to learn additional small clauses.

BackJump Learning Scheme (BJL): Assume a conflict at level x, meaning after assigning x literals $l_1, l_2, .., l_x$ to true, a conflict is reached. After conflict analysis the solver backjumps to a level b and learns a 1UIP clause $C = \{m_1, m_2, ..., m_{i-1}, m_i\}$. Only one literal m_i from C belongs to level x, and $b < x$, so after the first b decisions, if we had C in the clause database, unit propagation would prevent this conflict by assigning m_i true. Therefore, we can also learn clause $C_2 = \{\neg l_1, \neg l_2, ...\neg l_b, m_i\}$. For small values of b, this new learned clauses is small.

We submitted two versions:

- Maple_MBDR_BJL6_Local: This solver uses the BJL learning scheme described above and sets $b = 6$. The new learned clauses are stored in Local regardless of their LBD value.

- Maple_MBDR_BJL7_Tier2: This solver uses the BJL learning scheme described above and sets $b = 7$. The new learned clauses are stored in tier2 regardless of their LBD value.

REFERENCES

[1] A. Nadel and R. Vadim, "Chronological Backtracking," in Proceedings of SAT, 2018, pp. 111–121.

[2] A. Nadel and R. Vadim, "Maple_LCM_Dist_ChronoBT: Featuring Chronological Backtracking," in Proceedings SAT Competition 2018 - Solver and Benchmark Descriptions, 2018, pp. 29.

[3] SAT Competition 2020, *https://satcompetition.github.io/2020/*

[4] C. Oh, "Between SAT and UNSAT: The Fundamental Difference in CDCL SAT," in Proceedings of SAT, 2015, pp. 307–323.

[5] C. Oh, "Improving SAT solvers by exploiting empirical characteristics of CDCL", Ph.D. dissertation, New York University, 2016.

[6] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for SAT solvers," in Proceedings of SAT, 2016, pp. 123–140.

[7] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in Proceedings of IJCAI, 2009, pp. 399–404.

[8] M. Luo, C.-M. Li, F. Xiao, F. Manya, and Z. Lu, "An effective learnt clause minimization approach for CDCL SAT solvers," in Proceedings of IJCAI, 2017, pp. 703–711.

[9] X. Zhang and Sh. Cai, "Relaxed Backtracking with Rephasing," in Proceedings of SAT Competition 2020 - Solver and Benchmark Descriptions, 2020, pp. 15-16.

[10] S. Jamali and D. Mitchell, "Centrality-based improvements to CDCL heuristics Authors," in Proceedings of SAT, 2018, pp. 122-131.

[11] L. Freeman, "A set of measures of centrality based on betweenness," in Journal of Sociometry, volume 40(1), 1977, pp. 35-41.

[12] U. Brandes, "A faster algorithm for betweenness centrality," in Journal of Mathematical Sociometry, volume 25(2), 2001, pp. 163–177.