Kissat_MAB: Combining VSIDS and CHB through Multi-Armed Bandit

Mohamed Sami Cherif, Djamal Habet and Cyril Terrioux
Aix-Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
{mohamed-sami.cherif, djamal.habet, cyril.terrioux}@univ-amu.fr

***Abstract*—This document describes the Kissat_MAB solver which is based on the solver Kissat, winner of 2020 SAT competition. We augmented Kissat with a Multi-Armed Bandit (MAB) framework which combines the Variable State Independent Decaying Sum (VSIDS) and the Conflict-History Based (CHB) branching heuristics by adaptively choosing a relevant heuristic at each restart using the Upper Confidence Bound (UCB) strategy.**

***Index Terms*—SAT solver, Heuristics, Multi-Armed Bandit**

I. Introduction

Conflict Driven Clause Learning (CDCL) [8] solvers are known to be efficient on structured instances and manage to solve ones with a large number of variables and clauses. An important component in such solvers is the branching heuristic which picks the next variable to branch on. The Variable State Independent Decaying Sum (VSIDS) [9] has been the dominant heuristic since its introduction two decades ago. Recently, Liang and al. devised a new heuristic for SAT, called Conflict History-Based (CHB) branching heuristic [6], and showed that it is competitive with VSIDS. In the last years, VSIDS and CHB have dominated the heuristics landscape as practically all the CDCL solvers presented in recent SAT competitions and races incorporate a variant of one of them.

Recent research has shown the interest of machine learning in designing efficient search heuristics for SAT [5]–[7] as well as for other decision problems [4], [10]–[12]. One of the main challenges is defining a heuristic which can have high performance on any considered instance. Indeed, a heuristic can perform very well on a family of instances while failing drastically on another. To this end, we use reinforcement learning under the Multi-Armed Bandit (MAB) framework to pick an adequate heuristic among CHB and VSIDS for each instance. The MAB takes advantage of the restart mechanism in modern CDCL algorithms to evaluate each heuristic and choose the best one accordingly. The MAB uses the Upper Confidence Bounds (UCB) [1] strategy to select an arm at each restart.

II. Combining VSIDS and CHB through MAB

Let $A = \{VSIDS, CHB\}$ be the set of arms for the MAB containing different candidate heuristics. The proposed framework selects a heuristic $a \in A$ at each restart of the backtracking algorithm according to the Upper Confidence Bound (UCB) [1] policy. To choose an arm, UCB relies on a reward function calculated during each run to estimate the performance of the chosen branching heuristic. We choose a reward function that estimates the ability of a heuristic to reach conflicts quickly and efficiently. If t denotes the current run, the reward of arm $a \in A$ is calculated as follows:

$$r_t(a) = \frac{log_2(decisions_t)}{decidedVars_t}$$

$decisions_t$ and $decidedVars_t$ respectively denote the number of decisions and the number of decision variables, i.e. variables which were branched on at least once, in the run t. This reward function is adapted from the explored sub-tree measure introduced in [10].

The UCB1 algorithm [1] is used to select the next branching heuristic within the set of candidate heuristics A. The following parameters are maintained for each candidate arm $a \in A$:

- $n_t(a)$ is the number of times the arm a is selected during the t runs,
- $\widehat{r}_t(a)$ is the empirical mean of the rewards of arm a over the t runs.

UCB1 thus selects the arm $a \in A$ that maximizes $UCB(a)$ which is defined as follows :

$$UCB(a) = \widehat{r}_t(a) + c.\sqrt{\frac{ln(t)}{n_t(a)}}$$

The left-side term of $UCB(a)$ aims to put emphasis on arms that received the highest rewards. Conversely, the right-side term ensures the exploration of underused arms. The parameter c can help to appropriately balance the interchange between the exploitation and exploration phases in the MAB framework.

III. Implementation

We implement this idea in Kissat [3] which won first place in the main track of the SAT Competition 2020. Note that this solver is a condensed and improved reimplementation of the reference and competitive solver CaDiCaL [2], [3] in C. We maintain the VSIDS variant already implemented in Kissat which is similar to Chaff's where all analyzed variables are bumped after every conflict [9]. We also augment the solver with the heuristic CHB as specified in [6] except that we update the scores of the variables in the last decision level after unit propagation. In addition, we set the parameter c to 2. The rewards in UCB are initialized by launching each heuristic once during the first restarts. It is important to note that the only modified components of the solver are

the decision component and the restart component, i.e. all the other components as well as the default parameters of the solver are left untouched.

REFERENCES

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Mach. Learn.*, 47(2-3):235–256, 2002.

[2] Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In Tomáš Balyo, Marijn Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, pages 14–15. University of Helsinki, 2017.

[3] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[4] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. On the Refinement of Conflict History Search Through Multi-Armed Bandit. In Miltos Alamaniotis and Shimei Pan, editors, *Proceedings of 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 264–271. IEEE, 2020.

[5] Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Improving SAT Solver Heuristics with Graph Networks and Reinforcement Learning. *CoRR*, 2019.

[6] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3434–3440, 2016.

[7] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, pages 123–140, 2016.

[8] João P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[9] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, pages 530–535, 2001.

[10] Anastasia Paparrizou and Hugues Wattez. Perturbing branching heuristics in constraint solving. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming*, pages 496–513, Cham, 2020. Springer International Publishing.

[11] Hugues Wattez, Frederic Koriche, Christophe Lecoutre, Anastasia Paparrizou, and Sébastien Tabary. Learning Variable Ordering Heuristics with Multi-Armed Bandits and Restarts. In *Proceedings of the European Conference on Artificial Intelligence*, 2020.

[12] Wei Xia and Roland H. C. Yap. Learning Robust Search Strategies Using a Bandit-Based Approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6657–6665, 2018.