MapleSSV SAT Solver for SAT Competition 2021

Saeed Nejati*‡, Md Solimul Chowdhury†‡, Vijay Ganesh*
* University of Waterloo, Waterloo, Canada
† University of Alberta, Alberta, Canada
‡ Joint first authors

***Abstract*—This document describes the SAT solver MapleSSV, as part of which we implemented a set of heuristics that are found to be useful in solving SAT benchmarks that encode ARX (Addition-Rotation-XOR) functions. These heuristics are inspired by machine learning-based optimization methods, namely, improving branching using exploration, Bayesian moment matching based search initialization, and multi-armed bandit based restarts.**

I. Introduction

We present the SAT solver MapleSSV, which is a modification of the SAT solver MapleLCMDistChronoBT [1] (winner of the SAT competition 2018). There are four main modifications that we made to the base solver, enhancing branching, search initialization, restarts, and pre-processing. First, we used *exploration* in phases that the solver goes into a conflict depression (large sequences of decisions without learning any clauses) to get the solver to a more fruitful sub-space. Second, we used a Bayesian moment matching formulation of SAT to arrive at a promising initial search point, initializing variable order and polarities. Third, we employed multi-armed bandit based restarts to adaptively choose between restart strategies. Finally, we added XOR pre-processing to simplify the formula. Sections II, III and IV, describes each of these additions in more detail.

Motivation for Machine Learning-based Solver Heuristics: While a Boolean SAT solver is a decision procedure that decides whether an input formula is satisfiable, internally it can be seen as an optimization procedure whose goal is to minimize its run time while correctly deciding the satisfiability of the input formula. Every sub-routine in a SAT solver can be viewed either as a logical reasoning engine (i.e., a proof rules such as resolution in the case of conflict clause learning scheme or unit resolution in the case of BCP), or as a heuristic aimed at optimizing the sequencing, selection, and initialization of proof rules (e.g., variable selection, polarity selection, restarts, etc.). These optimization heuristic can in turn be implemented effectively using machine learning algorithms, since solvers are a data-rich environment. This philosophy was first articulated in the SAT 2016 paper by Liang et al. [2] on the LRB branching heuristic, has since been widely adopted and underpins many solver heuristics for branching, restarts, and initialization developed in recent years.

II. Exploration amid Conflict Depression Phases

Here we describe our exploration-based branching heuristic `expVSIDS`. This approach is based on our observation that CDCL SAT solving entails clear non-random patterns of bursts of conflicts followed by longer phases of *conflict depression (CD)* [3]. During a CD phase a CDCL SAT solver is unable to generate conflicts for a consecutive number of decisions. To correct the course of such a search, we propose to use exploration to combat conflict depression. We therefore design a new SAT solver, called *expSAT*, which uses random walks in the context of CDCL SAT solving. In a conflict depression phase, random walks help identify more promising variables for branching. As a contrast, while exploration explores *future* search states, LRB and VSIDS relies on conflicts generated from the *past* search states. In [3], we proposed `expVSIDS`, the exploration based extension of VSIDS. In addition to `expVSIDS`, our submitted solver MapleSSV, uses `expLRB`, the exploration based extension of LRB.

III. Initialization Problem

Many modern branching heuristics in CDCL SAT solvers assume that all variables have the same *initial* activity score (typically 0) at the beginning of the run of a solver. However, it is well known that a solver's runtime can be greatly improved if the *initial* order and value assignment of variables is not fixed a priori but chosen via appropriate static analysis of the formula. By the term initial variable order (resp., initial value assignment), we refer to the order (resp. value assignment) at the start of the run of a solver. This problem of determining the optimal initial variable order and value assignment is often referred to as the initialization problem.

In this work, we used a solution to the initialization problem based on a Bayesian moment matching (BMM) formulation of solving SAT instances and a concomitant method we refer to as BMM-based initialization [4]. Our method is used as a pre-processing step before the solver starts its search (i.e., before it makes its first decision).

A. Bayesian Moment Matching (BMM)

The SAT problem, simply stated, is to determine whether a given Boolean formula is satisfiable. In order to reformulate the SAT problem in a Bayesian setting, we start by defining a random variable for each variable of the input formula, where $P(x = T)$ shows the probability of setting x to True in a

satisfying assignment, assuming the formula is satisfiable. We assume that each of these variables has a Beta distribution, and collectively they form our prior distribution. We have the constraint that all of the clauses must be satisfied (i.e., it is assumed that the formula is satisfiable), therefore the clauses can be seen as evidence as to how the probability distribution should look like such that they are all satisfied. We then apply Bayesian inference using each clause as evidence to arrive at a posterior distribution. Applying Bayesian inference, gives us a mixture model, and this makes the learning intractable as the number of components grows exponentially with the number of clauses. To avoid this blow up, we use the method of moments to approximate the mixture Beta distribution with a single Beta distribution.

B. BMM as a Component in CDCL SAT Solvers

We implement an approximate version of the BMM method described above to solver the initialization problem of CDCL SAT solvers, since the complete method does not scale as the size of the input formulas increase. Fortunately, this approximate method is efficient and arrives at a promising point, as it attempts to satisfy as many clauses as possible. We take this starting point and initialize the preferred polarity and activity scores of each variable of an input formula, and then let the solver complete its search. The derived posterior distribution collectively represents a probabilistic assignment to the variables that satisfies most of the clauses. For polarity initialization, we used: $Polarity[x] = False$ if $P(x = T) < 0.5$ and True otherwise. For activity initialization, we gave higher priority to variables based on the confidence that BMM has about their values, i.e., $Activity[x] = \max(P(x = T), 1 - P(x = T))$. We initialized both VSIDS and LRB scores with the aforementioned methods.

IV. Multi-Armed Bandit Restart

Many restart policies have been proposed in the SAT literature [5], [6], in particular we focus on the uniform, linear, Luby, and geometric restart policies [7]. For a given SAT instance, we can not know a priori which of the 4 restart policies will perform the best. To compensate for this, we use multi-armed bandits (MAB) [8], a special case of reinforcement learning, to switch between the 4 policies dynamically during the run of the solver. We chose to use discounted UCB algorithm [9] from MAB literature, as it accounts for the non-stationary environment of the CDCL solver, in particular changes in the learnt clause database over time. Discounted UCB has 4 actions to choose from corresponding to the uniform, linear, Luby, and geometric restart policies. Once the action is selected, the solver proceeds to perform the CDCL backtracking search until the chosen restart policy decides to restart. The algorithm computes the average LBD of the learnt clauses generated since the action was selected, and the reciprocal of the average is the reward given to the selected action. Intuitively, a restart policy which generates small LBDs receives larger rewards and UCB increases the probability of selecting that restart policy in the future. Over time, this biases UCB towards restart policies that generate small LBDs for the give input SAT instance [10].

V. Availability and License

The source code of our solver have been made freely available under the MIT license. Note that the license of the M4RI library (which is used to implement Gaussian elimination) is GPLv2+.

Acknowledgment

We thank the authors of Glucose, GlueMiniSat, Lingeling, CryptoMiniSat, and MiniSAT for making their solvers available to us and answering many of our questions over the years.

References

[1] "Maplelcmdistchronobt, http://sat2018.forsyte.tuwien.ac.at/solvers/main_and_glucose_hack."

[2] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning Rate Based Branching Heuristic for SAT Solvers," in *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing*, ser. SAT'16, 2016.

[3] M. S. Chowdhury, M. Müller, and J. You, "Guiding CDCL SAT search via random exploration amid conflict depression," in *Proceedings of AAAI 2020*, 2020, pp. 1428–1435.

[4] H. Duan, S. Nejati, G. Trimponias, P. Poupart, and V. Ganesh, "Online bayesian moment matching based sat solver heuristics," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2710–2719.

[5] A. Biere, "Adaptive Restart Strategies for Conflict Driven SAT Solvers," in *Theory and Applications of Satisfiability Testing–SAT 2008*. Springer, 2008, pp. 28–33.

[6] G. Audemard and L. Simon, "Refining Restarts Strategies for SAT and UNSAT," in *Principles and Practice of Constraint Programming*. Springer, 2012, pp. 118–126.

[7] A. Biere and A. Fröhlich, "Evaluating CDCL Restart Schemes," in *Pragmatics of SAT*, 2015.

[8] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. MIT Press Cambridge, 1998, vol. 135.

[9] A. Garivier and E. Moulines, *Algorithmic Learning Theory: 22nd International Conference, ALT 2011, Espoo, Finland, October 5-7, 2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ch. On Upper-Confidence Bound Policies for Switching Bandit Problems, pp. 174–188.

[10] S. Nejati, J. H. Liang, C. Gebotys, K. Czarnecki, and V. Ganesh, "Adaptive restart and cegar-based solver for inverting cryptographic hash functions," in *Working Conference on Verified Software: Theories, Tools, and Experiments*. Springer, 2017, pp. 120–131.