CryptoMiniSat with CCAnr at the SAT Competition 2020

Mate Soos (National University of Singapore)
Shaowei Cai (State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences)
Jo Devriendt (Lund University & University of Copenhagen)
Arijit Shaw, Kuldeep S. Meel (National University of Singapore)

I. Introduction

This paper presents the conflict-driven clause-learning (CLDL) SAT solver CryptoMiniSat (*CMS*) augmented with the Stochastic Local Search (SLS) [4] solver CCAnr as submitted to SAT Competition 2020.

CryptoMiniSat aims to be a modern, open source SAT solver using inprocessing techniques, optimized data structures and finely-tuned timeouts to have good control over both memory and time usage of inprocessing steps. It also supports, when compiled as such, to recover XOR constraints and perform Gauss-Jordan elimination on them at every decision level. For the competition, this option was disabled. CryptoMiniSat is authored by Mate Soos.

CCAnr [4] is a stochastic local search (SLS) solver for SAT, which is based on the configuration checking strategy and has good performance on non-random SAT instances. CCAnr switches between two modes: it flips a variable according to the CCA (configuration checking with aspiration) heuristic if any; otherwise, it flips a variable in a random unsatisfied clause (which we refer to as the focused local search mode). The main novelty of CCAnr lies on the greedy heuristic in the focused local search mode, which contributes significantly to its good performance on structured instances

II. Composing the Two Solvers

The two solvers are composed together in a way that does *not* resemble portfolio solvers. The system runs the CDCL solver CryptoMiniSat, along with its periodic inprocessing, by default. However, at every 2nd inprocessing step, CryptoMiniSat's irredundant clauses are pushed into CCAnr (in case the predicted memory use is not too high). CCAnr is then allowed to run for a predefined number of steps. This in total leads to about 1% of all solving time dedicated to CCAnr. In case CCAnr finds a satisfying assignment, this is given back to the CDCL solver, which then performs all the necessary extension to the solution (e.g. for Bounded Variable Elimination, BVE [6]) and outputs the final solution.

In case CCAnr does not find a satisfying assignment, the following takes place. Firstly, the best variable setting found by CCAnr as measured by the number of satisfied clauses, is assigned as the polarity of the variables in the CDCL SAT solver. This idea has been taken from the solver CaDiCaL [3] as submitted to the 2019 SAT Race by Armin Biere. Secondly, after every successful execution of CCAnr, 100 variables' VSIDS are bumped in the following way. CCAnr uses a clause weighting technique and clauses with greater weight can be considered more difficult to satisfy. Once CCAnr finishes, CCAnr's clauses are sorted according to their weights. Then, these clauses' variables' VSIDS are bumped, from hardest-to-easiest clause order, until 100 variables' VSIDS have been bumped. This shows clear improvement in the combined solver's performance. We believe these two integrations point to potential tighter, as-yet unexplored integration opportunities of the two solvers.

Note that the inclusion of the SLS solver is full in the sense that assumptions-based solving, library-based solver use, and all other uses of the SAT solver is fully supported with SLS solving enabled. Hence, this is not some form of portfolio where a simple shell script determines which solver to run and then runs that solver. Instead, the SLS solver is a full member of the solver, much like any other inprocessing system, and works in tandem with it. For example, in case an inprocessing step has reduced the number of variables through BVE or increased it through BVA [9], the SLS solver will then try to solve the problem thus modified. In case the SLS solver finds a solution, the main solver will then correctly manipulate it to fit the needs of the "outside world", i.e. the caller.

As the two solvers are well-coupled, the combination of the two solvers can solve problems that neither system can solve on its own. Hence, *the system is more than just a union of its parts* which is not the case for traditional portfolio solvers.

III. Gauss-Jordan Elimination

As per the upcoming paper [12], the Gauss-Jordan elimination of CryptoMiniSat has been significantly improved. The average speed increase for moderately sized matrices is approx 3-6x, allowing the system to be ran at all times even when the matrix is not contributing as much to the overall solving. Hence, for the first time in CryptoMiniSat's 10 year history, Gauss-Jordan elimination is turned on by default for the NoLimits track.

IV. Symmetry Breaking using BreakID and Bliss

The BreakID [5] system is a cost-effective symmetry breaking preprocessor for SAT. Classic SAT symmetry pre-

processing [1] detects symmetry by converting the input formula to a graph and computing generators for this graph's automorphism group, and adds symmetry breaking clauses on a generator-by-generator basis. On top of this, BreakID heuristically searches for structure in the automorphism group, detecting *row interchangeability symmetry* (such as in the pigeonhole problem) and computing binary symmetry breaking clauses from orbits arising from the symmetry group. The resulting symmetry breaking clauses are more effective at reducing symmetrical assignments from the search space, both from a theory point of view as well as in practical experiments.

BreakID has been modified to work as a library. It can receive the clauses on-the-fly from the SAT solver, and produce the breaking clauses as a function return value. Various small bugs have also be fixed, such as memory leaks, which were not an issue when ran as a single executable, but created isses when ran as a library. Furthermore, the underlying highly sophisticated graph automorphism detection system, Bliss [7], has been slightly improved to allow for time-outs and it, too, has been fixed not to leak memory. BreakID is fully integrated into CryptoMiniSat by calling it on every 5th inprocessing iteration, and asked to contribute breaking clauses. These are always added with an assumption literal, so they can be removed when the solving finishes. Hence, symmetry breaking also works when CryptoMiniSat is used as a library.

V. Phase Selection using LSIDS

LSIDS is a literal activity-based phase selection heuristic [10]. LSIDS activity is maintained for each literal, and the activity for a literal is updated in a manner similar to VSIDS. Phase selection is made based on LSIDS activity only if the last backtrack is chronological. The LSIDS based phase selection heuristic looks at the activity of both the literals of a given variable and selects the literal with higher activity.

VI. Further Improvements Relative to SAT Race 2019

Many of the inprocessing parameters have been tuned. A few bugs related to clause activities have been fixed. Clause distillation (or clause vivification) [8] is now used a lot more, similarly to the previous years' winning solvers. The VSIDS and Maple decay factors are now iteratively changed between 0.70 and 0.90 for Maple and 0.92 and 0.99 for VSIDS. Between each iteration there is an inprocessing step, as before. This seems to add heterogeneity and avoids having to tune these parameters to a "single best" value. Polarity caching is still used, but once in a while, so-called "stable" polarities are used, as per CaDiCaL [3] in the SAT Race of 2019. Ternary resolution is also used at every inprocessing step, thanks to the suggestion by Armin Biere.

VII. Thanks

Kuldeep S Meel, Arijit Shaw, and Mate Soos were supported in part by the National Research Foundation Singapore under its NRF Fellowship Programme[NRF-NRFFAI1-2019-0004] and AI Singapore Programme [AISG-RP-2018-005], and NUS ODPRT Grant [R-252-000-685-13].

Shaowei Cai was supported by Beijing Academy of Artificial Intelligence (BAAI), and Youth Innovation Pro-motion Association, Chinese Academy of Sciences [No. 2017150].

The computational work for this article was performed on resources of the National Supercomputing Center, Singapore[2]. Mate Soos would also like to thank all the users of CryptoMiniSat who have submitted over 600 issues and pull requests to the GitHub CMS repository[11].

References

[1] Aloul, F.A., Sakallah, K.A., Markov, I.L.: Efficient symmetry breaking for Boolean satisfiability. IEEE Trans. Comput. 55(5), 549–558 (May 2006), https://doi.org/10.1109/TC.2006.75

[2] ASTAR, NTU, NUS, SUTD: National Supercomputing Centre (NSCC) Singapore (2018), https://www.nscc.sg/about-nscc/overview/

[3] Biere, A.: CaDiCaL SAT solver GitHub page (2020), https://github.com/arminbiere/cadical

[4] Cai, S., Luo, C., Su, K.: Ccanr: A configuration checking based local search solver for non-random satisfiability. In: Heule, M., Weaver, S.A. (eds.) SAT 2015. LNCS, vol. 9340. Springer (2015)

[5] Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: Improved static symmetry breaking for SAT. In: Creignou, N., Le Berre, D. (eds.) Theory and Applications of Satisfiability Testing – SAT 2016. pp. 104–122. Springer International Publishing, Cham (2016), https://bitbucket.org/krr/breakid

[6] Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing. pp. 61–75. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[7] Junttila, T.A., Kaski, P.: Engineering an efficient canonical labeling tool for large and sparse graphs. In: ALENEX 2007. SIAM (2007)

[8] Li, C., Xiao, F., Luo, M., Manyà, F., Lü, Z., Li, Y.: Clause vivification by unit propagation in CDCL SAT solvers. Artif. Intell. 279 (2020)

[9] Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of boolean formulas. In: Biere, A., Nahir, A., Vos, T. (eds.) Hardware and Software: Verification and Testing. pp. 102–117. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[10] Shaw, A., Meel, K.S.: Designing new phase selection heuristics. In: SAT 2020 (2020)

[11] Soos, M.: CryptoMiniSat SAT solver GitHub page (2018), https://github.com/msoos/cryptominisat

[12] Soos, M., Gocht, S., Meel, K.S.: Accelerating approximate techniques for counting and sampling models through refined CNF-XOR solving. In: CAV 2020 (2020)