1 Introduction

In this project, you will equip your CDCL solver with more advanced techniques in various aspects. Over the last years, the area of SAT solving has seen tremendous progress. Many problems (e.g., in hardware and software verification) that seemed to be completely out of reach a decade ago can now be handled routinely. Besides new algorithms and better heuristics, the efficiency of algorithm implementation also turned out to be vital for the success. Given that you have already implemented a CDCL solver in Homework 3, it would be an interesting next step to further draw inspiration from recent improvements in SAT solving to boost the vanilla CDCL algorithm.

2 Grouping

Please check the online sheet and reach the TAs if you have any questions.

3 Requirements

Based on your CDCL implementation in Homework 3, you are required to equip the solver with the following techniques. Experimental comparison of the effect of different techniques is usually necessary. To verify whether certain techniques are effective, you can test them on large CNF instances from the SAT competitions after 2018. For your convenience, we provide some instances collected from SAT Competition 2018 and 2022. You can find them in the attachment. Note that there are various possible ways to implement these techniques and any reasonable solutions are welcomed.

1. Better branching heuristics: Consider improving the VSIDS heuristic. One possible solution is to formulate the problem of deciding which variable to branch on next as a multi-armed bandit problem. This idea has been explored in previous works [1, 2]. Try implementing LRB [1] and CHB [2] to replace the VSIDS algorithm for calculating the ranking of variables. ~~You can also incorporate classic bandit algorithms like UCB and EXP3 into the CDCL solver.~~ You are supposed to implement at least one bandit-based branching heuristics.

2. Restarting: Restarting is a way of dealing with the heavy-tailed distribution of running time often found in combinatorial search. Intuitively, restarting is meant to prevent the solver from getting stuck in a part of the search space that contains no solution. The solver can often get into such situation because some incorrect assignments were committed early on, and unit resolution was unable to detect them. If the search space is large, it may take very long for these incorrect assignments to be fixed. Hence, in practice, SAT solvers usually restart after a certain number of conflicts is detected (indicating that the current search space is difficult), hoping that, with additional information, they can make better early assignments. Try incorporating a restart policy into the CDCL solver by referring to related materials like [3, 4] and Section 7.3 of [5], and apply classic bandit algorithms like UCB and EXP3 to switch between candidate branching heuristics after restarting for a better and more diverse exploration of the search space. See [3].

3. (Bonus) Preprocessing techniques: The vanilla CDCL algorithm can be incorporated with pre- and in-processing steps to further speedup SAT solving. For example, bounded variable elimination (BVE), as a landmark in the history of preprocessing, can reduce the number

of variables and clauses in a formula at any stage in a solver's execution. Try implementing BVE or other techniques presented in [6] and Chapter 9 of [5].

4. (Bonus) Better algorithm implementation: You can also consider speeding up your CDCL solvers with efforts like improved data structures, better scheduling of inprocessing, and optimized algorithms with lower computational complexity. Check the implementation details of well-known SAT solvers like Cadical [6] and Kissat [7] for inspiration.

4 Evaluation

In this project, the final score will be determined by:

1. Completion & performance (15 pts): The submitted solver needs to be bug-free and correctly equipped with the required techniques. Your solver will be evaluated on large CNF instances and more efficient solvers get higher scores.

2. Report (5 pts): You need to provide a technical report (with the NeurIPS template) to describe your implementation and experimental details. The report length is limited to 8 pages for main contents (including figures and tables) and unrestricted for references. Additional pages containing references are allowed. Besides, it is required to explicitly summarize the contributions of each group member in the report, in terms of job assignment and percentage of contribution.

3. Presentation (5 pts): In the last class, each group will give a 5-min presentation (plus 1 minutes for Q&A).

Note that your submission should contain one single SAT solver, which will be evaluated with instances other than those provided. Any operations after the `.cnf` file is read (including preprocessing) will be recorded in terms of the amount of time it takes to complete. We will grade each member based on both group performance, workload, and individual contribution.

References

[1] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2016.

[2] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential recency weighted average branching heuristic for SAT solvers. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 3434–3440. AAAI Press, 2016.

[3] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. Combining VSIDS and CHB using restarts in SAT. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[4] Jia Hui Liang, Chanseok Oh, Minu Mathew, Ciza Thomas, Chunxiao Li, and Vijay Ganesh. Machine learning-based restart policy for CDCL SAT solvers. In Olaf Beyersdorff and Christoph M.

Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 94–110. Springer, 2018.

[5] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

[6] 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.

[7] Armin Biere Katalin Fazekas Mathias Fleury and Maximilian Heisinger. Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020. *SAT COMPETITION*, 50:2020, 2020.