Dynamic Break Parameter Adjustment in Stochastic Local Search
07-300, Fall 2021

Edward He
https://ehe9991.github.io/

February 3, 2022

1 Project Description

I will be working with Professor Marijn Heule in the Computer Science Department on trying to dynamically change the break parameter in stochastic local search (c_b) in order to improve performance of the YalSAT solver.

Determining whether a boolean formula is satisfiable or not is an NP-complete problem, meaning we are not sure whether it can be solved in polynomial time as of now. Even so, there are many approaches towards solving boolean satisfiability that work well in practice. Typically, these solvers take in a boolean formula in conjunctive normal form and output a satisfying assignment if there is one, and unsatisfiable if there is no such assignment.

The local search algorithm begins with a complete random assignment to the literals in the formula, and repeatedly changes a literal in an unsatisfied clause based on a heuristic. In this way, local search may not always output a satisfying assignment even if there is one. ProbSAT is an algorithm that chooses the literal in an unsatisfied clause based solely on how many clauses flipping the literal will "break", i.e. how many it will unsatisfy. This parameter is known as c_b. YalSAT is a specific efficient implementation of ProbSAT. Even though it is not complete (i.e. will not always output a satisfying assignment if there is one), ProbSAT works well in practice.

However, in ProbSAT, the optimal parameter for c_b is not known beforehand, nor is it evident based on the formula itself. In many problems, changing the value of c_b will drastically improve or hurt performance. For near-optimal values of c_b, ProbSAT is one of the best SAT solvers to discover a satisfying assignment. However, for poorly chosen values of c_b, ProbSAT may never return any satisfying assignment.

Our direction for this project is to dynamically adjust the break parameter based on whether the solver is making progress or not. Intuitively, this should improve performance because we would be approaching a more optimal constant if our original selection of c_b was bad.

The major challenge going into this project will be to determine whether or not the solver is making progress or not, and what direction to adjust our initial c_b. Due to the inherently random nature of ProbSAT, it may be difficult to improve the performance in every run of the solver. However, we do aim to improve performance on average. While analyzing the mathematical basis behind the potential performance changes is out of the scope of this project, we do aim to generate a significant performance improvement.

2 Project Goals

2.1 75% Project Goal

- Consistently determine whether local search is "making progress" or not, perhaps using a heuristic.
- Find one configuration of changing c_b where we can improve the performance of ProbSAT in a specific case (i.e. k-SAT for a particular k).

2.2 100% Project Goal

- Consistently determine whether local search is "making progress" or not, perhaps using a heuristic.
- Find one configuration of changing c_b where we can improve the performance of ProbSAT in a specific case (i.e. k-SAT for a particular k).
- Create multiple policies for changing c_b based on whether progress is being made, and find performance changes for each policy.
- Determine whether it is possible to have one policy for dynamically adjusting c_b that improves performance the most of ProbSAT for all k-SAT, and if so, find that policy.

2.3 125% Project Goal

- Consistently determine whether local search is "making progress" or not, perhaps using a heuristic.
- Find one configuration of changing c_b where we can improve the performance of ProbSAT in a specific case (i.e. k-SAT for a particular k).
- Create multiple policies for changing c_b based on whether progress is being made, and find performance changes for each policy.
- Determine whether it is possible to have one policy for dynamically adjusting c_b that improves performance the most of ProbSAT for all k-SAT, and if so, find that policy.
- Perform the same analysis on ProbSAT with polynomial literal selection, and see if the same policies can still improve on the runtime.
- Determine whether restarts can effectively be used with this dynamic adjustment, and see whether dynamically adjusting c_b without restarts is more effective than dynamically adjusting c_b after completely restarting.

3 Project Milestones

3.1 First Technical Milestone: End of Fall Semester

Gain an understanding of the underlying code in YalSAT, and experiment with random adjustments of the c_b parameter. Identify some heuristics which may dictate whether local search is making progress or is stalled. Investigate the c_b values given in the existing YalSAT code and ProbSAT paper, and verify their efficiency.

3.2 First Biweekly Milestone: February 15th

Identify and confirm whether heuristics are effective in judging whether a run is making progress.

3.3 Second Biweekly Milestone: March 1st

Draft policies of changing c_b based on which direction (either increase or decrease, and by how much) that potentially improve performance of YalSAT, and confirm whether they do indeed improve the performance.

3.4 Third Biweekly Milestone: March 15th

Confirm final selection of policies and heuristics, and if no viable policies/heuristics continue to search for some.

3.5 Fourth Biweekly Milestone: March 29th

Create testing framework that can measure the time which the solver takes to run, and begin empirically testing the performance of various policies, as well as default YalSAT.

3.6 Fifth Biweekly Milestone: April 12th

Analyze specific performance of the various policies compared to each other, determine whether the results mark a significant improvement over ProbSAT in the general case. Begin constructing figures/tables representing potential improvement.

3.7 Sixth Biweekly Milestone: April 26th

Draft paper describing the project, and if time permits explore ways to go beyond the 100% goals for the project.

3.8 Seventh Biweekly Milestone: May 10th

Finalize the paper, and explore future improvements of my project, and ways the project can be expanded on.

4 Literature Search

Many efficient stochastic local search algorithms have been developed [3] [4]. ProbSAT is a promising stochastic local search algorithm that can be much faster than existing methods based on c_b [1]. YalSAT is a competitive implementation of ProbSAT [2]. Already, there is research suggesting that random restarts can improve performance [5]. However, as of yet, there is no research attempting to dynamically adjust the c_b value during runtime.

5 Resources Needed

To conduct this research, we obtain the latest version of YalSAT from the Institute for Formal Models and Verification. We also may need a testing framework to build off of to measure the performance of each configuration. Apart from this, we need a standard machine that we can test all of the configurations on. We have most of the resources for the project, missing only a testing framework.

References

[1] Adrian Balint and Uwe Schöning. Choosing probability distributions for stochastic local search and the role of make versus break. In *SAT*, 2012.

[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] Holger H. Hoos and Thomas Stützle. Local search algorithms for sat: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, May 2000.

[4] Md Shibbir Hossen and Md Masbaul Alam Polash. An efficient local search sat solver with effective preprocessing for structured instances. *SN Computer Science*, 2(2):105, Feb 2021.

[5] Jan-Hendrik Lorenz and Julian Nickerl. The potential of restarts for probsat. *Lecture Notes in Computer Science*, page 352–360, 2020.