CSCI 2951O: Project Report

February 25, 2022

Alex Ding (CS: ading13. Screen: evian)

1 Design

My initial implementations were in Python, a terrible choice in hindsight. Next time, I'll probably start with C++ or Julia. I started with the DPLL algorithm shown on the slides (with both pure literal elimination and BCP), but the performance was terrible, only being able to solve the toy instances on time. After consulting relevant literatures, I decided to get rid of pure literal elimination (which rarely occurs) and implemented 2 watched literals, relying only on branching and BCP. Afterwards, I extracted the original pure literal elimination and BCP routines as a preprocessing technique to simplify the instance before the decide-BCP-repeat loop that uses watched literals. This setup was able to solve a good number of SAT instances in 2-3 minutes. A quick but effective improvement was to implement the Jeroslow-Wang heuristic for variable selection, which helped runtime tremendously. At this point, the model was struggling with UNSAT instances, and I was out of ideas on how to improve my code without CDCL, which is exactly what I implemented next. This was very tricky to get right, and I had to read up a lot. I ended up going with the same implementation Chaff proposed, terminating clause expansion at the first UIP and flipping the assignment after nonchronological backtracking. My code was performant at this point, but after profiling, I realized that Jeroslow-Wang was the bottleneck, as the number of clauses grows quickly. I then implemented VSIDS (EVSIDS specifically), which showed mixed results (I realized later that there was a bug in my code which caused EVSIDS to not work correctly). I also implemented random restarts with a Luby sequence, which also didn't help things too much.

At this point, I decided to move to a faster language. I chose Julia for its syntactic similarity with Python, which helped with the translation, but the fact that it's compiled and has performance comparable to C guaranteed a fast runtime. I never used the language before, so this was a really fun (and time-consuming) learning experience, but this cut the time spent on most instances by 5 times. I spent a lot of time optimizing the various parts of the code to shave some more time off. The solver can solve all except 5 instances in time, all of which are SAT. I have some suspicion that something is not right with my SAT termination conditions, but I can't figure it out for the life of me, and at this point I've already way too many hours on the project in 1 week, so I'm calling it quits.[1]

2 Failed Attempts

I observed that the number of clauses quickly balloons out of control in the SAT cases it can't solve, and this slows the solver down a lot. A lot of these learned clauses are super long and kind of useless, so I tried to implement an on-line clause deletion scheme, but when I delete an old clause it breaks the implication graph (e.g., some variable assignment is caused by the deleted clause; what should I do now?). I don't really have time to investigate this further, but something along this line looks promising.

I also pursued the idea that my code doesn't work on non-trivial satisfiable instances further. I found a bunch of satisfiable DIMACS files online, but my solver performed well on all of them, so I'm not sure what's going on with the satisfiable instances in the test suite. I noticed that they're all instances with very few variables (50 or 75) and relatively many clauses (1000-2000), so something more on a pure DPLL-based solution would work a lot better?

3 Total Time Spent

Around 60 hours. Lost a bit of my sanity, but it was very rewarding as well. Plus, I would love to see what is wrong with my SAT solver for these SAT instances. If someone can take a look that'd be fantastic.

[1] This was a wild experience. My life became SAT solvers, and I spent most of my waking hours thinking about it. It's somewhat discouraging, though, that my final solver, with all the bells and whistles, can't even solve a third of the test cases :(, but still I definitely learned a lot. Anyways, I'm definitely going to partner up next time to not go through this again.