PA-5 Logic and Sudoku

Sudharsan Balasubramani Credit: Discussed ideas with James Fleming and Mack Reiferson
Note: Read README.md

References

- Professor Alberto Quattrini-Li and Slack
- Wikipedia
- Artificial Intelligence: A Modern Approach

Introduction

The goal of this assignment is to work with SAT solvers. SATisfiability problems are extremely popular, and many computer science problems are actually reduced to SAT problems for a SAT solver to solve. This specific assignment designs solvers for propositional logic satisfiability problems. Specific cases we use are sentences derived from Sudoku logic puzzles, where the CNF of the sentence is based on work done by Ivor Spence. The solvers we make are the GSAT SAT solver and the WalkSAT solver. We model and solve propositional logic SAT problems with these.

Description

Both solver algorithms work in a similar manner; there are some key differences though. However, before we get into the algorithm, a bit about the parsing behind to set the problem up for the solver.

Framework

In general, a SAT solver needs a clauses from the CNF, a probability threshold to choose a random variable to flip, and the max number of flips allowed before we end and say there was no solution derived. How are we supposed to obtain all this from just a singular CNF file? When we create a SAT solver object, the constructor initializes with a given max_flips and threshold. Then, we initialize a list of clauses, an assignment dictionary, and a list of satisfied_clauses and unsatisfied_clauses. We then call generate_clauses. This function is the meat of the parsing. It opens the file, reads all the lines, which are clauses. Then for each clause, it splits it by the space character and then for each variable in that clause list, it adds to self.variables (a set) the truncated version of a symbol (if it had a "-" in front). Then it appends the entire clause list to self.clauses, a list of clauses. So our clauses are a list of lists, where each list is separated by an AND in CNF. Now we have all we need for the solvers!

GSAT

The GSAT algorithm differs from the WalkSAT in the methods used to select which variable to flip. GSAT makes the change which minimizes the number of unsatisfied clauses in the new assignment, or with some probability, picks a variable at random. The inputs for this would normally be max flips, clauses, and a probability, but that is all provided internally via the constructor. GSAT first generates a random assignment, where each var in the dictionary is set to a random val of true or false. Then, we run the following algorithm for the max amount of flips. We first check if the assignment satisfies the clauses. This is done with satisfy(), which runs through every clause in self.clauses and then through each literal in each clause and checks if there is at least one literal that makes that clause true. If so, that is added to the list of satisfied clauses. If not, then it is added to the list of unsatisfied clauses. If unsatisfied_clauses is of length 0, then you know you are done, for all clauses have been satisfied. If we are satisfied, we return the assignment; if not, then we move on. We first select a random probability and then compare with threshold. If it is above the threshold, then we randomly flip one of the variables, doesn't matter which one. This is done with random_flip.random_flip chooses one var, and then flips it in the assignment. If we don't go the probabilistic route, then we use flip_var with the parameter being the list of variables. flip_var then goes through all literals or vars in that list and calculates which var would satisfy the most clauses and unsatisfy the least clauses. Then, from the maximum of these, we choose one randomly, and flip it. This continues until we run out of flips and return no solution or we find a solution.

WalkSAT

The WalkSAT algorithm is different in that it works on a clause that is still unsatisfied by the current assignment, rather than flipping a random variable. This unsatisfied clause is chosen at random. Once again, the var in that clause that results in the fewest previously satisfied clauses becoming unsatisfied and the maximum clauses becoming satisfied will be preferred and will be randomly be chosen to flip among others with the same property. The beginning is essentially the same: we choose a random assignment, check if it satisfies, and return it if it does so. If not, we then take a slightly different path. Rather than just choosing a random var to flip or a max score of all vars, we first choose a random clause that is **unsatisfied**. From there, we once again we pick a probability and if above a threshold, flip a variable in that **randomly chosen unsatisfied clause** rather than just a random var. If not probabilistic, then we once again flip the variable that satisfies the most clauses and unsatisfies the least clauses, chosen from the **clause**. We continue until we either find a solution or until we reach max_flips, in which case we return none.

Evaluation

GSAT was an interesting algorithm. Indeed, it was quick with one_cell and somewhat quick with all_cells. At rows, it started to get a little slower, and then though I knew rows_and_cols would terminate, it took way too long. This is because it would have to loop through every single variable and find the var that would satisfy the most clauses rather than simply going through variables within a single clause as WalkSAT does. Now, playing with the threshold does help in speed for GSAT. This makes sense, as if we make it low, then most flips would be random. However, this would get us nowhere, as we are always just flipping a random variable and not ever going towards the assignment. On the contrary, it seems intuitive to just have the threshold be 1, so that we are always picking the value that maximizes the satisfaction. However, this isn't exactly the case. Indeed, the randomness is quite helpful in escaping the local minima. In fact, in both GSAT and WalkSAT, you can end up in situation where not any variable from either the clause list or variable list can satisfy clauses. The only way to reach a solution would be to go backwards, which is essentially what the random flip does: it make the assignment worse and goes back before it can choose a new path and make it better. The idea is to first flip some variables to reduce the total amount of satisfied clauses and then later on flip other variables to increase the net satisfied clauses. I also realized that just checking for the maximum satisfied clauses for each algorithm to pick the variable to flip was a little problematic, for one var can satisfy a lot of clauses, but ultimately unsatisfy more clauses. I made sure to then check the number of clauses we were unsatisfying and then subtracted it from the number satisfied to get the net satisfaction.

Here I discuss some of my findings. First I start with just one_cell. It took my GSAT a total of 4 flips while it took my WalkSAT a total of 3 flips. This makes sense, WalkSAT is slightly faster, but with one cell, not by much. We then move onto all_cells. There is a clear difference in speed. GSAT is much slower in flips, taking a total of 409 flips, for it must loop through all vars every time. Contrastingly, WalkSAT took much less flips, 292 flips, as it is looping through a smaller candidate list each time, and because of that, was incredibly fast. For rows, it was the same deal (GSAT: WalkSAT: 458). At rows_and_cols, my GSAT would most definitely terminate and reach a solution, but at that point, it was too computationally slow and expensive compared to WalkSAT to run anymore (GSAT: Too long, WalkSAT: 1866). WalkSAT is, again, incredibly fast. Now we begin to just use WalkSAT. With the rules and the puzzles onward, even WalkSAT took a while, but that is due to the catch of being at local minima. Both algorithms quickly reach the 10s in terms of unsatisfied clauses but stay for a while. For rules, WalkSAT takes 5554 flips. Puzzle 1 does also take a while; we reach sub-10 unsatisfied clauses really quickly but once again take a while to reach a solution, about 44895 flips. Similarly, puzzle 2 takes 54004 flips, due to more values already filled in (more constraints). All the solution files are provided in the zip-files to view!