COSC 76: Artificial Intelligence

Programming Assignment 5: Logic

Student: Amittai Wekesa (github: @siavava)

Fall 2021

To solve the Sudoku problems, I implemented a generic handler, `SAT`, in SAT.py that reads in satisfiability clauses in CNF form and works on finding an assignment that ensures all clauses are satisfied.

DESIGN DECISIONS

1. Tracking a Two-Way Mapping While implementing my `SAT` class, I found myself needing to map CNF elements to their respective indices in the assignments array and later to map back from indices to corresponding clauses. To keep it clean, I implemented a `TwoWayDict` data structure that masks two Python dictionaries and allows one to easily get a key associated to a value and vice versa.

See TwoWayDict.py for implementation details.

2. Which variables to track In the SAT class, I found it most efficient to keep an array of assignments over a dictionary. Because many assignments are done and undone, we don't have to use dictionary lookups twice as much. On the other hand, I ended up building a mapping of cnf clauses to correspoknding indices sets do not keep indices. I chose to implement an interface to a 2-way mapping (described above) so that reverse mappings are fast when needed (e.g. when writing the solutions into a file). I also track a threshold probability (defaulted to 0.3, but can be overriden by an explicit value when initializing the `SAT`). This threshold is used in `WalkSAT` and `GSAT` to determine if the next flip should be done randomly or greedily.

3. Scoring Both algorithms need a way of scoring (albeit each differs from the other's way).

1. For GSAT, I implemented a method, `gsat_highest_var`, that iterates over **every** index in the assignment array, flips the variable and tests satisfiability, then flips back to the original value. This obviously takes long when the problem has many CNF clauses, and is why GSAT takes longer *per iteration* than WalkSAT.
2. For WalkSAT, I implemented a method, `walksat_highest_variable`, that iterates over pre-selected candidates, flips them, and checks the satisfiability score. This is faster since we only consider unsatisfied clauses.

4. GSAT Algorithm The GSAT algorithm loops until a specified number of maximum iterations is reached, either greedily flipping an assignment that yields best satisfiability rate or randomly flipping an assignment until all constraints have been satisfied. The algorithm uses the scoring function above, `gsat_highest_vars`, to score candidates and is therefore quite slow over each iterations since it has to score every possible flip in the problem.

5. WalkSAT Algorithm The WalkSAT algorithm works similarly to GSAT: while the maximum number of iterations has not been reached, either greedily *an unsatisfied* assignment that yields best satisfaction rate or randomly flip *any unsatisfied assignment.* Here, we do not consider satisfied assignments, so the algorithm runs faster on each iteration as it does not have to score every other flip in the problem. However, some problems typically need thousands of iterations to be solved. Perhaps a deterministic solver such as DPLL would have better performance.

A quirk: for WalkSAT, we need to efficiently build a set of candidates for the next flip – that is, a set of all unsatisfied propositions. The most efficient way to do this is to check satisfiability while scoring states and save unsatisfied clauses as candidates for the next flip. That way, we don't have to deviate later on to find candidates.

6. Functionality: GSAT My GSAT algorithm makes significant progress on all problems and is able to solve simple ones such as the one_cell (in a few seconds, after around 10 iterations) and all_cells (in a several minutes after around 200 to 500 iterations). However, on more complicated problems such as puzzle1, GSAT takes too long to make meaningful progress since it spends considerable time on each iteration.

7. Functionality: WalkSAT My WalkSAT algorithm works, way better (faster) than GSAT. It was able to solve puzzle2 in 19109 iterations, lasting a total of 465 seconds. Every other puzzle in the tests I ran took a shorter time. To see sample WalkSAT runs (truncated for readability), see the runtimes folder output