PA 5 - Logic

CS 76 AI, Fall 2021

Melissa Valencia

Overall Design

`SATInstance.py`, represents an SAT problem and outputs the solution files. The SAT problem is defined by the `cnf_filename` passed in a To implement solvers for propositional logic satisfiability problems such as those represented by `SATInstance.py`, `SAT.py` the general SAT algorithm, along with the necessary functions defining the variations necessary for the GSAT and WalkSAT algorithms.

Implementation

In `SAT.py`, the general SAT algorithm applied to both GSAT and WalkSAT is `sat` which takes in a limit of iterations and a `gsat_dict` or a `wsat_list`. This algorithm then loops through all the satisfied clauses. In each iteration, `choose_variable` returns a variable either randomly chosen from the given dict/list or chosen from scoring the variables based off each's net satisfied clauses. `get_overall_satisfied` scores the variables using their domains which is a dictionary containing the only their associated clauses. This function counts how many clauses are satisfied using `valid_clause`, both before and after flipping the variable, and lastly returns the net satisfied. The variable is then randomly chosen from a list of best score variables. The chosen variable is then flipped and the algorithm continues until all clauses are satisfied.

GSAT Description

To implement GSAT within the general `sat` algorithm, `gsat_dict` is used to define the GSAT variable selection which chooses randomly from all the variables. Then, the limit of iterations and this are passed into `sat` which is called in `gsat`.

Evaluation

GSAT had a longer runtime than WalkSAT and was unable to get solutions to the puzzles provided. This function makes the change which minimizes the number of unsatisfied clauses in the new assignment or picks a variable randomly. This algorithm performed as expected with all the files provided.

Test Cases

I first tested `gsat` within `SAT.py`. Count refers to the number of iterations on the `sat` algorithm.

```
$ python3 SAT.py one_cell.cnf
```

```
count 6
```

```
$ python3 SAT.py all_cells.cnf
```

```
count 389
```

```
$ python3 SAT.py rows.cnf
```

```
count 560
```

WalkSAT Description

To implement GSAT within the general `sat` algorithm, `wsat_list` is used to define the WalkSAT variable selection which randomly chooses from an unsatisfied clause. Then, the limit of iterations and this are passed into `sat` which is called in `wsat`.

Evaluation

WalkSAT had a better runtime and greater scalability than GSAT in this case, as it was able to get solutions for the provided puzzles. Since it chooses from a smaller number of candidate variables that might be flipped, there are fewer possibilities and thus there is less calculations to be done. This algorithm performed as expected with all the files provided.

Test Cases

I first tested `wsat` within `SAT.py`. Count refers to the number of iterations on the `sat` algorithm.

```
$ python3 SAT.py one_cell.cnf
```

```
count 4
```

```
$ python3 SAT.py all_cells.cnf
```

```
count 306
```

```
$ python3 SAT.py rows.cnf
```

```
count 475
```

```
$ python3 SAT.py rows_and_cols.cnf
```

```
count 1526
```

```
$ python3 SAT.py rules.cnf
```

```
count 3588
```

```
$ python3 SAT.py puzzle1.cnf
```

```
count 26528

```

```
$ python3 SAT.py puzzle2.cnf
```

```
count 26857
```

Here is the output from `solve_sudoku.py` running `wsat` with max iteration of 100000 and `puzzle1.cnf`. Count refers to the number of iterations on the `sat` algorithm.

```
$ python3 solve_sudoku.py puzzle1.cnf
```

```
count 23440
5 6 1 | 4 8 7 | 9 3 2
2 8 4 | 3 9 6 | 5 7 1
3 9 7 | 1 2 5 | 8 6 4
---------------------
8 1 9 | 7 6 2 | 4 5 3
6 5 2 | 8 4 3 | 1 9 7
7 4 3 | 9 5 1 | 6 2 8
---------------------
1 7 5 | 6 3 8 | 2 4 9
4 3 6 | 2 1 9 | 7 8 5
9 2 8 | 5 7 4 | 3 1 6
```

Here is the output from `solve_sudoku.py` running `wsat` with max iteration of 100000 and `puzzle2.cnf`. Count refers to the number of iterations on the `sat` algorithm.

```
$ python3 solve_sudoku.py puzzle1.cnf
```

```
count 29546
5 3 4 | 2 9 8 | 1 6 7
2 7 6 | 1 4 3 | 5 9 8
1 9 8 | 6 5 7 | 4 3 2
---------------------
8 5 2 | 7 6 4 | 9 1 3
6 1 9 | 8 3 5 | 2 7 4
7 4 3 | 9 1 2 | 8 5 6
---------------------
9 8 7 | 5 2 6 | 3 4 1
4 2 1 | 3 7 9 | 6 8 5
3 6 5 | 4 8 1 | 7 2 9
```