Report: Sudoku and propositional logic

Juan C. Torres Ramirez

Introduction

My objective is to find solutions for Sudoku puzzles by representing the sudoku puzzles as satisfiability problems. T do so, we have a set of clauses and we must satisfy every clause to solve the problem. Each clause is presented in conjunctive normal form. In doing so, I can use GSAT and WalkSAT to solve the boards.

Implementation

In this algorithm, I initialize each variable in the model with a random value and change the value of one variable in each iteration until the model matches all the constraints. For both GSAT and WalkSAT, in each iteration there is a probability that this variable to change will be chosen randomly; otherwise, the algorithm optimizes which variable t choose by minimizing the number of unsatisfied clauses when flipping a variable. GSAT and WalkSAT differ in how they make this choice.

Working with .cnf files

I parse and internalize the information contained in .cnf files as follows:

- I store every variable that appears in a clause in a set, `variables`
- I create an index for every variable; this index is the one I will use when checking clauses and changing the model. I create two dictionaries: one for translating from variables to indices and one for doing the opposite.
- I store each clause as a dictionary of variable indices to booleans. For instance, `-113 114` might be stored as `{3: False, 4: True}`. This increases memory usage compared to using a set or a list, but makes dealing with constraints significantly easier.

I have implemented several variables common to in a common class, `SAT`. These variables include the list of variables, dictionaries to go from variable to index numbers and back, and so on.

GSAT

GSAT optimizes the choice of a variable to flip in each iteration (if not chosen at random) by flipping the variable tha maximizes the number of clauses satisfied by the model, which is equivalent to minimizing the number of unsatisfie clauses in the model. I implemented this in the `choose_best_variable(model)` function, which calls a couple helpers in short, this function flips each variable and counts the number of constraints satisfied with that new model. I get tl variables that satisfy the maximum number of constraints and choose one at random.

WalkSAT

In contrast to GSAT, WalkSAT does not consider all variables when choosing which one to flip; instead, it first finds the unsatisfied clauses in the previous iteration and chooses one of those clauses at random; it then considers only the variables in that clause when maximizing the number of constraints satisfied. To do so, I call `choose_candidate_variables` and pass those candidate variables to `choose_best_variable` so that it considers onl these variables when optimizing the best variable to flip.

Testing

GSAT

One cell

```
Iterations needed: 5
9 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
---------------------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
---------------------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
```

All cells

```
Iterations needed: 458
1 1 4 | 9 6 7 | 5 5 1
5 7 4 | 2 8 9 | 6 3 6
3 1 1 | 5 1 2 | 3 2 7
---------------------
5 8 1 | 8 4 7 | 3 7 6
1 1 3 | 3 5 1 | 3 6 5
2 8 7 | 4 9 5 | 7 9 3
---------------------
6 5 6 | 2 7 6 | 1 5 3
1 1 1 | 7 8 7 | 5 5 7
1 8 4 | 3 7 6 | 8 8 1
```

Rows

```
Iterations needed: 690
4 1 8 | 3 5 2 | 6 9 7
6 7 2 | 3 5 4 | 9 8 1
5 1 9 | 8 4 6 | 2 3 7
---------------------
6 5 8 | 4 3 7 | 9 1 2
8 1 4 | 7 5 9 | 2 6 3
2 3 6 | 9 7 5 | 1 8 4
---------------------
2 7 1 | 6 9 4 | 5 8 3
9 2 4 | 3 8 7 | 5 1 6
6 8 3 | 7 9 4 | 2 1 5
```

Any more complex sudokus proved to take too long to report results.

WalkSAT

One cell

```
Iterations needed: 3
9 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
---------------------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
---------------------
0 0 0 | 0 0 0 | 0 0 0
```

```
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
```

All cells

```
Iterations needed: 410
4 6 4 | 8 7 6 | 7 9 4
4 7 2 | 1 1 4 | 7 2 5
8 6 8 | 8 4 4 | 4 1 4
---------------------
5 7 5 | 1 9 5 | 5 9 1
2 5 7 | 1 9 6 | 4 6 1
2 4 8 | 4 4 4 | 2 9 4
---------------------
2 7 2 | 4 4 4 | 8 5 7
6 1 6 | 2 2 7 | 7 1 3
1 2 6 | 9 2 4 | 1 7 8
```

Rows

```
Iterations needed: 542
8 6 4 | 3 5 1 | 7 9 2
5 2 6 | 8 1 9 | 4 3 7
4 1 9 | 8 2 5 | 7 6 3
---------------------
5 7 6 | 4 1 8 | 2 3 9
1 4 5 | 8 9 7 | 3 6 2
7 1 3 | 8 6 5 | 2 9 4
---------------------
2 1 4 | 8 9 6 | 7 3 5
4 9 6 | 2 8 5 | 1 7 3
7 1 5 | 4 9 6 | 3 2 8
```

Rows and cols

```
Iterations needed: 2730
6 3 4 | 1 5 8 | 9 7 2
4 9 2 | 6 7 1 | 8 3 5
8 4 1 | 5 2 9 | 7 6 3
---------------------
2 1 3 | 4 9 7 | 5 8 6
7 2 5 | 8 3 4 | 6 1 9
5 8 6 | 7 1 3 | 2 9 4
---------------------
1 5 9 | 2 8 6 | 3 4 7
9 7 8 | 3 6 5 | 4 2 1
3 6 7 | 9 4 2 | 1 5 8
```

Rules

```
Iterations needed: 7480
9 3 5 | 4 6 8 | 7 2 1
8 1 2 | 5 7 9 | 4 3 6
4 7 6 | 2 1 3 | 8 5 9
---------------------
6 8 1 | 3 9 5 | 2 7 4
7 4 3 | 1 8 2 | 9 6 5
5 2 9 | 6 4 7 | 3 1 8
---------------------
2 5 4 | 8 3 6 | 1 9 7
```

```
3 9 8 | 7 5 1 | 6 4 2
1 6 7 | 9 2 4 | 5 8 3
```

Puzzle 1

```
Iterations needed: 75374
5 9 7 | 4 2 6 | 8 1 3
6 8 1 | 3 9 5 | 4 2 7
4 2 3 | 7 8 1 | 5 6 9
---------------------
8 3 2 | 1 6 7 | 9 5 4
1 6 9 | 8 5 4 | 3 7 2
7 5 4 | 9 3 2 | 1 8 6
---------------------
9 7 6 | 5 4 8 | 2 3 1
2 4 8 | 6 1 3 | 7 9 5
3 1 5 | 2 7 9 | 6 4 8
```

Puzzle 2

Puzzle 2 takes too long to be solved in a reasonable time.