Solving Sudoku with SAT Reductions

Aidan Boyle (20840670)

3 December 2022

1 Background

Sudoku is a well known and internationally popular puzzle, that can be played with pencil and paper. This game is often found in newspapers, or in small booklets that are sold at various book stores.

Definition: A Sudoku game is a 9x9 grid, which starts off with a number of the squares of the grid filled in with numbers 1,...,9, and the rest empty. The goal of the game is to fill in every entry of the board with a digit 1,...,9, such that each row, each column, and each 3x3 sub-grid contains each digit exactly once.

A simple way to find solutions to Sudoku boards, would be to employ a brute force method, such as backtracking. However, for this project, we solve Sudoku using SAT reductions.

In particular, we create a program that takes a partially solved Sudoku board as input, and converts it into a boolean CNF formula ϕ, such that ϕ has a solution if and only if the Sudoku board has a solution. As described in the paper by Inês Lynce and Joël Ouaknine [1], the boolean formula ϕ consists of 729 propositional variables, where each x_{ijk} semantically means that position (i, j) of the sudoku board has entry k. From here, the authors describe how to build the CNF formula for the Sudoku board, using what is known as the Minimal Encoding.

I have implemented the Minimal Encoding in `C++`, and complied into the executable `sud2sat`, which takes a partially solved sudoku board as input, and outputs the cooresponding boolean CNF formula in `DIMACS` format. From here, we can use a a SAT solver such as `minisat` on our newly constructed boolean formulas, which will assign truth values to each of our variables x_{ijk} that make ϕ true. Aftering running `minisat`, using the truth assignments of the varaibles x_{ijk}, we can now find the actual solution to the board.

I implemented this program in `C++`, and compiled into the executable `sat2sud`, which takes the assignment of variables outputted by `minisat`, and prints to standard output the solved board. This program was simple to implement, since all you need to do is determine which variables x_{ijk} are true, since if x_{ijk} is true, it means that the (i, j) entry of the solved sudoku board will have entry k.

2 Statistics

The tables below, outlines worstcase, average, and best case statistics for `minisat`.

Case	Restarts	Conflicts	Decisions	Propogations
Worst Case	1	123	169	4690
Average Case	1	9	18.92	983.6
Best Case	1	0	1	583

Case	Conflict Literals	Memory Used	CPU Time
Worst Case	900	11 MB	≈ 0.1 s
Average Case	57.24	11 MB	≈ 0.00865 s
Best Case	0	11 MB	≈ 0 s

3 Conclusion

After reading through Inês and Joël's paper, the actual implementation of the two programs was quite simple, however it was very eye opening, and really demonstrates the power of NP-Complete problems such as SAT. Furthermore, the actual reduction to SAT, is very analogous to the proof of the Cook-Levin theorem, which as a result, has helped me better understand the proof, and appreciate it for both for its eloquence, and powerful implications.

4 References

[1] Inês Lynce, and Joël Ouaknine. Sudouku as a SAT Problem. In *Ninth International Symposium on Artificial Intelligence and Mathematics*, January 2006.