Sudoku Solver

August 30, 2021

Andrew Ramirez

Method

For this algorithm, I chose to follow the Linear Programming (LP) set through the given examples. Seeing as the given constraints for a sudoku game are essentially all that is needed, the method described within the examples is what made the most sense to me. So from there, I had to try and play around with different ways of manipulating the given code in order to improve the efficiency. Unfortunately I did not fare well in doing that, but I did see a minor improvement when using scipy.optimize.linprog with the method 'revised simplex' as opposed to the default 'interior-point'. However, that comes at the cost of significantly longer time to actually find the optimizations, and it is possible that the improvements were only because of the seed, and not because the method was actually better (though it is meant to be more accurate).

The actual problem that is being solved in this case is:

minimize ||x|| subject to

Let x_{ijk} indicate the event that the (i, j) element of the Sudoku grid contains k. If it is true, $x_{ijk} = 1$, otherwise $x_{ijk} = 0$. Then the constraints are

- Column, $\sum_{i=1}^{9} x_{ijk} = 1$ for $1 \leq j, k \leq 9$
- Row, $\sum_{j=1}^{9} x_{ijk} = 1$ for $1 \leq i, k \leq 9$
- Box, $\sum_{j=3p-2}^{3p} \sum_{i=3q-2}^{3q} x_{ijk} = 1$ for $1 \leq k \leq 9$ and $1 \leq p, q \leq 3$.
- Grid, $\sum_{k=1}^{9} x_{ijk} = 1$ for $1 \leq i, j \leq 9$.
- Clues, should be given from the problem.

The integer constraint is $x_{ijk} \in \{0, 1\}$.

The code works by essentially combining all of those constraints into a single equality, such that it can be represented as Ax = B, where A will include both the fixed constraints as well as the clue constraints, and B is simply a 1's matrix (all entries are 1).

Success Rates

Data	Success rate	Success rate %
Set A (small 1)	23/24	95.833%
Set A (small 2)	360/1000	36%
Set B (large 1)	836/1000	83.6%
Set B (large 2)	991/1000	99.1%

Most methods I used gave 100% success for small 1 and large 2. This one didn't, but it gave slightly improved accuracy for small 2 and for large 1, and an overall higher success rate because of that. I was hoping for a bigger improvement, but I struggled in actually implementing the changes I wanted and thus couldn't make enough of a change to improve the code.

Issues + Reflection

I'm aware that it really does not seem like I did much for this project (with regards to my own code). I did attempt a few other methods, as well as trying to build my own version of this LP from scratch, though it either didn't work properly, or just ended up being an incorrect implementation of the example code. There were also other methods I wanted to try, which involved different packages, but I didn't fully understand how to install any of them, so I couldn't even attempt to make use of them. I spent quite a lot of time trying to figure something out but I really couldn't. I understand what was expected, but for

some reason I wasn't able to apply what I've learned in a meaningful way. Maybe starting sooner could have helped, but I'm honestly not sure.

Also worth noting, before implementing the random seed, some of the chosen puzzles led to an error of "SVD does not converge". I tried to find work arounds to that, but again, I couldn't figure it out. When I implemented the random seed, it seems as though none of the chosen puzzles run into this problem, so I ultimately ignored the issue.

References

https://www.kaggle.com/gaz3ll3/simple-lp-solver-for-sudoku-success-rate-0-32

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html