Exploring Optimization Techniques for Sudoku

May 17, 2019

Chad Breece, Zarin Ohiba, Malcolm Yang, Aleksandr Savchenkov

Randomized Optimization:

Within Random Optimization there were a few techniques used to try to solve this problem.

First, we implemented a Genetic Algorithm which did not work very well. The GA would very often get stuck in local maxima and had an extremely hard time finding the optimal solution. If properly coded, this method could theoretically find the optimal solution but it would take half an hour of running to do so.

The second attempt at a randomized method was a Hill Climbing algorithm which we thought would be better than the genetic and it seemed to although it still took a very long time to reach the optimal solution, if at all.

Third attempt was to implement simulated annealing as a method to prevent the hill climbing from getting stuck in the local maximum and instead actually be able to find the optimal solution within a reasonable amount of time.

Overall, the Random Optimization methods used did not seem likely to do better than Linear Programming would. With the lack of clear constraints and the random nature, these algorithms could in theory find the optimal solution and in a somewhat reasonable time but there are simply too many local optima that these algorithms will get stuck in while attempting to converge to the global optimum. Furthermore, I think that the way sudoku is and the way we create a fitness function for a game of sudoku makes it very hard for these algorithms to learn what to do and move in the right direction. There is no close answer in sudoku, there is only a correct one, and for this reason the algorithm is more likely to end in a local optima. For these reason, random optimization will not work very well.

Linear Programming:

We used the paper “A Warm Restart Strategy for Solving Sudoku by Sparse Optimization Methods” to implement the linear programming method to solve the Sudoku problem. Initially we only attempted minimizing L1 norm of x with constraints but we couldn’t make it more better than the given base code. We then solved the problem of minimizing L1 norm of weighted X. Starting with minimizing L1 norm of |Wx| with the given constraints, we find the matrix W. And then we use sco.linprog to solve the minimization problem given the constraints. After getting

some potential solution, we check to see if that solution has any duplicate/repeated values using our defined function “repeats”. After deleting repeated values and adjusting for the initial given clues, we get some x value which we use to pass to clud_constraints function to get updated constraint and solve the same problem. After 2-3 iteration we return the x value we get. So, after the solver returns the x values, we check that against the actual solution and count the number of correct Sudoku puzzles solved. And then calculate accuracy by dividing total correct count by sample size chosen.
Reporting accuracy on the datasets using LP approach:

Small1	100%
Small2	63%
Large1	93%
Large2	100%

Neural Networks:

Two neural network based techniques were evaluated, a CNN, and one titled “Relational Recurrent Networks.

For the CNN we test the novel state-of-art convolutional neural network to solve the Sudoku challenge. We borough the idea from https://github.com/Kyubyong/sudoku. Even though there is no clues how CNN work for identifying the number. It does not read or interpret the the constraints, but the CNN is incredibly powerful for some cases. For accuracy rate the CNN having some issue but for uniqueness rate the CNN can provide a good rate.

Small1: 100%
Small2: 32%
Large1: 82%
Large2: 86%

The Recurrent Relational Networks solve sudoku similarly to how humans do. Each cell has a state vector that is then mapped to an output probability. The next state of this cell is dependent on the current states of cells which are directly constrained to it in the same way that conventional RNNs connect cells temporally.

This functions to allow each cell to “tell” those it is constrained to what values it’s mapping to so that constrained cells choose separate values over time.

The code provided in the paper claims 97% accuracy, but this is only per cell, not for entire puzzles. We found that it was able to achieve 90% accuracy on easy puzzles, but failed outright on the hard puzzles provided in the **small2** dataset.

Conclusion:

Based on the discussed methods we tried, our conclusion is that the linear programming approach works best for the given set of problems.