Haley Feng
May 15, 2019

Project 1: Sudoku Challenges

Method

I used linear programming (LP) method with implementation of recursion to solve all four sets of data. After running the necessary function *fixed_constraints* and *clue_constraint,* I made some changes to the starter code of LP by adjusting the solver as a function *solver(quiz, solu)* within the for loop. Once I get a solution using the function, I will need to check if the solution *z* matches with the actual solution *solu.* To do so I checked if the norm of the difference between the two, denoted as *D*, is less than zero. If it is less than zero that means the solution is correct and the code moves to the next puzzle. If it's not, then it goes on to the computation to check which digit in the matrix is not zero. This is done by using a for loop to check which indices of the difference is nonzero and reset the LP solution of those indices into zero. Plug this back into the function again and check to see if the norm passes. If yes then the puzzle is solve and no then the code goes to the next puzzle.

Data Sets	**Time**	**Total**	**Success**
small_1	0.89 ~ 0.9 secs	24	100%
small_2	1.17 ~ 2.42 secs	1011	100%
large_1	0.97 ~ 1.05 secs	1000 (random sample)	99%
large_2	0.78 ~ 0.83 secs	1000 (random sample)	100%

Mistake

The code worked great because of the mistake of relying on the actual solution to find the repeated value. I shouldn't of implement this after the solver but instead check for repeated variable inside the LP function. In the future or if more time is given, the changes I will make is try to implement a checker to check every column and row of the sudoku to find the repeated variable. When I find the repeated value, I will change the column or row in to zero and plug it back into the LP again to solve. I think this method might work because I briefly tried it on one example of sudoku and the new solution matches the actual solution.