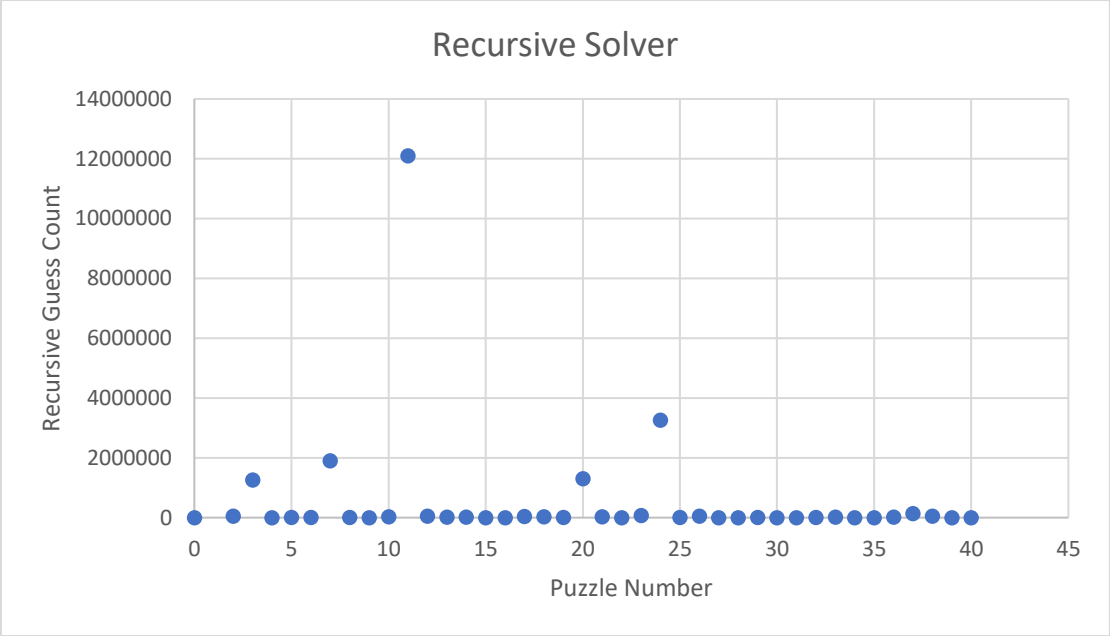
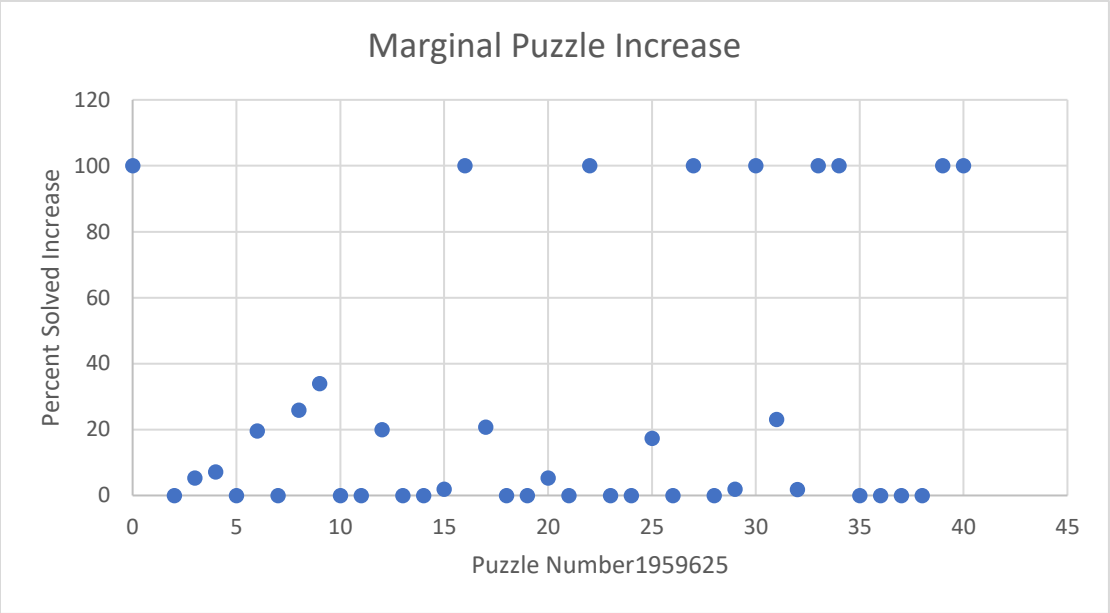
Landon Crowther
CS 2420
Assignment 3 - Sudoku

Sudoku Analysis:



1) This figures shows the number of recursive guesses required for the different puzzles in the .zip file that was provided. I calculated the average, and found that the average number of guesses required to complete the puzzle was 511,378.05. As I was computing these values, I realized that they were so scattered and arbitrary. I also calculated the standard deviation to help clarify if there was any “rhyme or reason” to the recursive guess count. The standard Deviation was 19,59,624.529. Because this number is so large, it confirms that there is no consistency with these puzzles. If I would have altered my code so that the first guess in a cell was 9 instead of 1, I could have completely different results.



2) For the most part, the constraint solver didn't work. More times than not, we were only able to partially complete the puzzle with the constraint solver. We chose to measure the "success" of the solver with the following value:

$$Marginal\ Completion\ Increasae = \frac{CellsFilledDuringAlgorithm}{NumberOfInitialEmptyCells}$$

We felt like this was a decent way to represent the success of the solver in a way that was scale scale-able to all puzzles. Judging by the results of our data, we were wrong. I do believe this measurement is better than "number of values solved" or "total fraction completed," though. This is because those two values depend on the starting conditions of the puzzle, which are different for each puzzle. Our measurement is relative to the starting conditions.

For the most part, the constraint algorithm either a) solved the whole puzzle (rare), b) didn't do anything (most common), or c) gave us a "marginal completion increase" of 1-20%.

3) I spent considerably more time on this project than I expected, probably totaling around 17-20 hours. This was super frustrating, because my partner and I had the algorithms working on Tuesday. The tests took way longer than we originally anticipated.
4) As mentioned before, the testing was the most time consuming part of this project. Not only was setting up the tests tricky, but with the amount of repetition the tests required it was easy to make simple mistakes. I wish I could have allocated more time at the beginning of the week for the project. However, this was difficult because it wasn't until after class on Tuesday that I felt like I could progress through the project. Getting over the initial syntax issues and concepts regarding the Sets was a bit of a learning curve.
5) Recursion functions can be hard to set up. It felt like it never worked, and then suddenly it did. I understand how this function works, though. The only thing that was slightly tricky was

determining when to “return true” or “return false” and exit the method, because sometimes I didn’t know if I had fulfilled all the checkmarks that the algorithm required at that point.

6) The constraint solver is like what would seem like the “logic test.” This is the strategy used by most people when they play Sudoku. It works by looking at the values of the puzzle, and assigning a set of values [1-9] to all empty cells. Then, anywhere there is a number, the sets in the corresponding row, column, and box of the number is cleared of that number. This is repeated until a set only has one number, and by logic, the value of that cell must be that number.
7) This project was exciting, however, I felt like the template/instructions were a bit vague and misleading at times. For example, we originally put the HashSets in the Sudoku() constructor. If this was the case, it would have made testing the prune methods much simpler. However, once we read the instructions of the solve_by_elimination() method, we realized that the HashSets were supposed to be constructed in that method.

   One thing I did like about this project though was the amount of variety that could be used to solve the problems. When I would talk to TAs, they said that they saw certain problems being solved so many ways. I think projects with this type of flexibility are generally a bit more fun, and better for learning as well. After all, nobody in the workplace is going to provide a template and method names for us to go off.