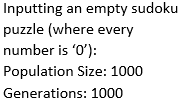
Analysis of Program 4: Solving Sudoku Puzzles

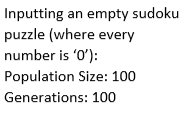
Eric Pham & Timothy Kozlov

3/11/21

**Program Performance**

For every generation we have in our program, we create Sudoku puzzles based off of our population size. We use a constant time algorithm that evaluates the efficiency for each puzzle in the generation. Even though it is a constant time algorithm, the performance is still relatively slow; each row, column, and 3x3 subdivision must be checked for each puzzle. During culling a population, we use an O(n2) algorithm that is a modified, optimized version of a selection sort. We use a single array because we do not need to re-instantiate a dynamic array whenever a new generation is created since the size is unchanged. Our program was able to process a population size of 100000 within a few seconds.

**Test Cases (Empty Sudoku)**



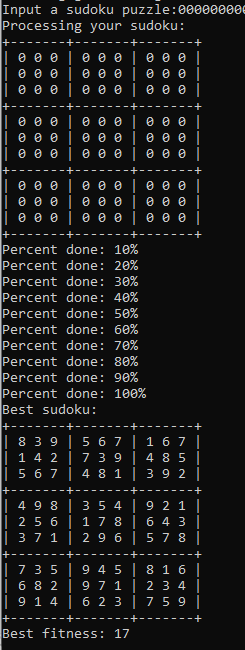


Figure 1 - Empty Population Size of 100 & 100 Generations

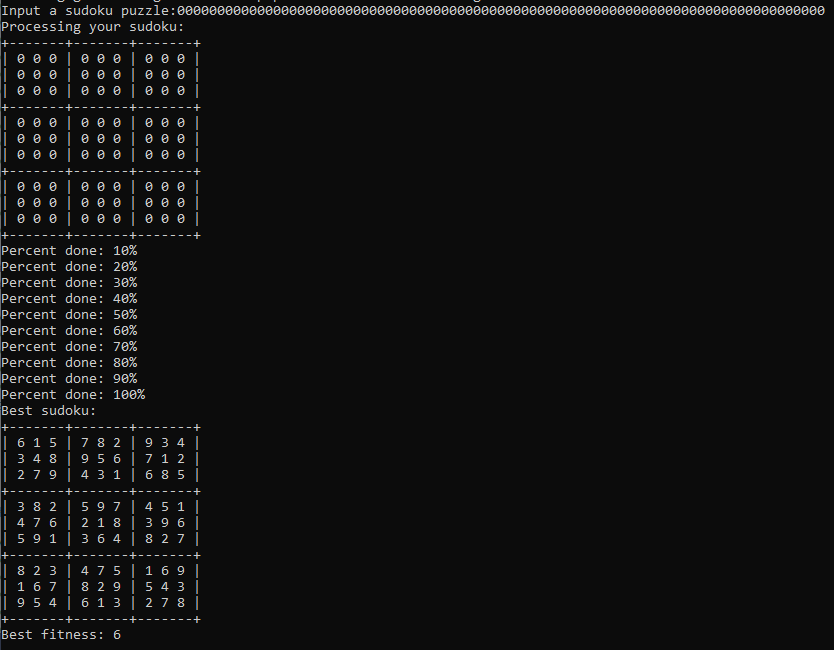
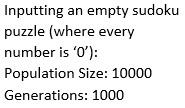


Figure 2 - Empty Population Size of 1000 & 1000 Generations



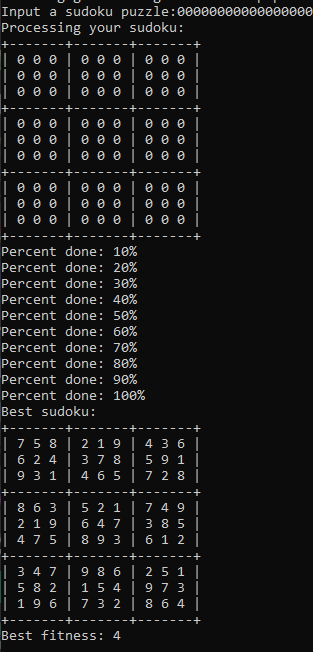
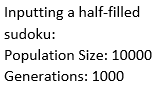
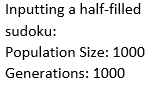
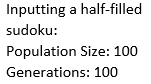


Figure 3- Empty Population Size of 10000 & 1000 Generations

**Test Cases (Half-filled Sudoku)**



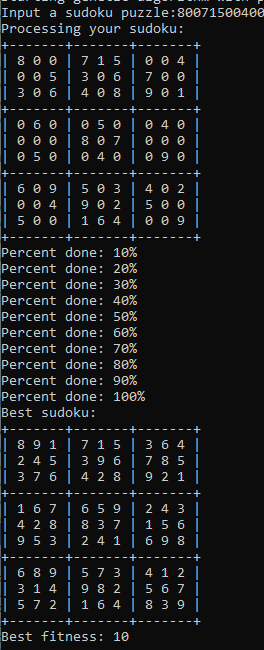


Figure 5 - Half-Filled Population Size of 1000 & 1000 Generations

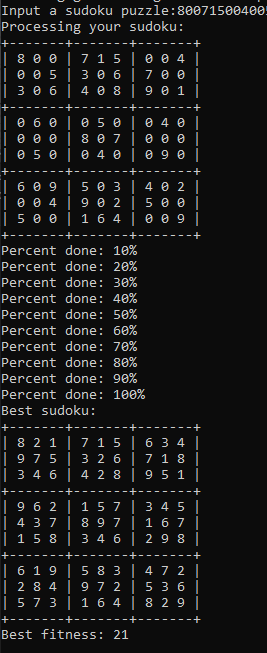


Figure 4 - Half-Filled Population Size of 100 & 100 Generations

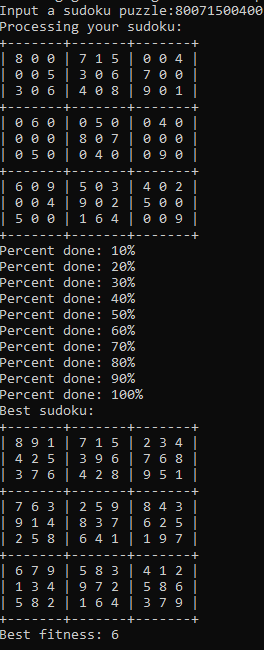


Figure 6 - Half-Filled Population Size of 10000 & 1000 Generations

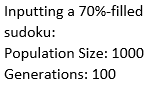
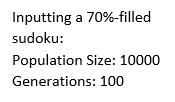
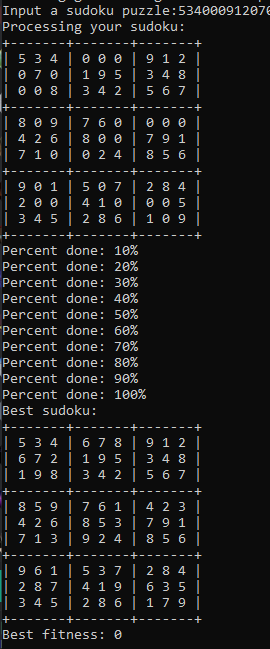
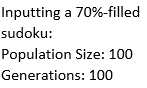
**Test Cases (70%-filled Sudoku)**

Figure 9 - 70%-Filled Population Size of 10000 & 1000 Generations



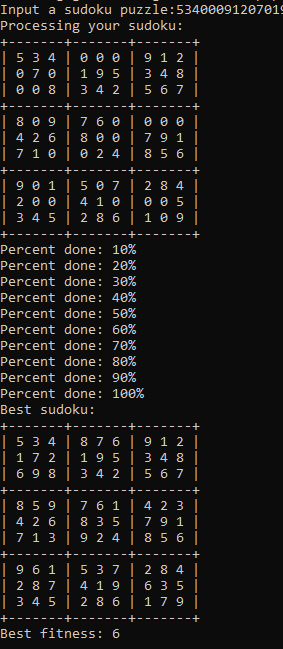
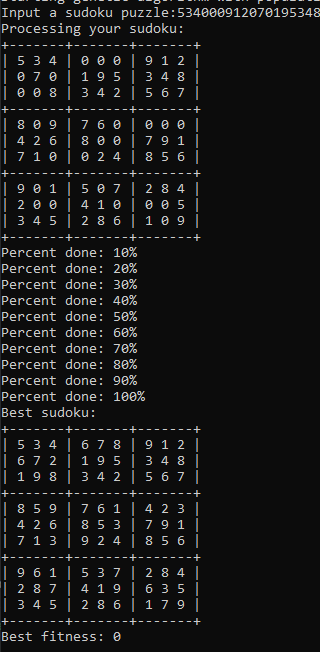


Figure 7 - 70%-Filled Population Size of 100 & 100 Generations

Figure 8 - 70%-Filled Population Size of 1000 & 1000 Generations

**Conclusion**

On average, in the three test cases above (empty-filled, half-filled, 70%-filled sudokus), regardless of which case it is, increasing either the population size or the number of generations lowers the fitness levels, where the lower the fitness level, the closer the sudoku puzzle is to perfection. 

**Empty Sudokus:** We found that for empty sudoku puzzles, when the population size and number of generations are both over 1000, the genetic algorithm performs relatively well because there are no fixed squares that might interfere with possible solutions. It can effectively find a sudoku puzzle with a fitness score of approximately less than 10. Empty sudokus are relatively hard in compared to sudoku puzzles that only require a small number of squares that need to be filled in, in order for it to be complete. As such, it takes more generations than relatively easier puzzles.

**Half-filled Sudokus:** Despite having more fixed squares than in the empty sudoku, the algorithm performed noticeably worse than that of the empty sudoku because there are a non-trivial number of fixed squares that can interfere with each possible solution, but not enough where it can guide the algorithm. For example, if only ten squares in a sudoku were un-filled, then the algorithm could easily evaluate the possible solutions since there are a limited number of options to choose from.

**70%-filled Sudokus:** The algorithm performed best in this test case because there are enough filled in squares that can effectively guide the algorithm to find the best possible solution. In *Figure 8* and *Figure 9*, the population size and number of generations were large enough to that it found a perfect solution for each respective test case.

Overall, increasing the population size and number of generations lowered the fitness level of the sudoku puzzle. Half-filled sudokus performed the worst out of all three test cases, empty sudoku fitness values were relatively low in comparison of half-filled sudokus, and 70%-filled sudokus showed the best overall fitness values where two of the three test case scenarios showed perfect results.