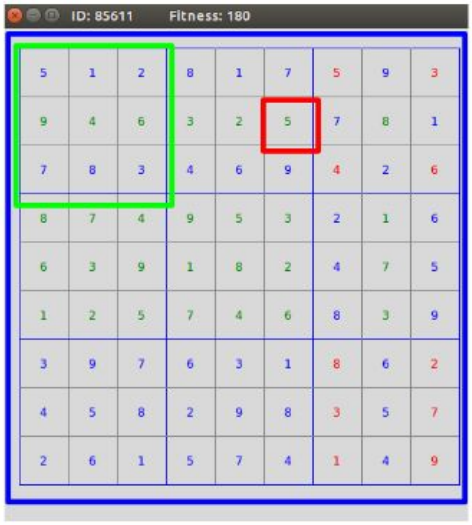
Glen Simon
CSE 848
Homework 4
Due: 10-30-17

Sudoku GA

A) Basic Terminology:

For easier understanding of my explanations I'm going to define some of the terms that I use in this report and my code in this section.

1. Puzzle - The whole N by N Sudoku Puzzle (blue box)
2. Block - A sqrt(N) by sqrt(N) subsection of the Sudoku Puzzle (green box)
3. Element - Single value in the Sudoku Puzzle (red box)



4. Three Rules of Sudoku -
    a. A block must be equal to the set of integers {1 - N}
    b. A row must be equal to the set of integers {1 - N}
    c. A column must be equal to the set of integers {1 - N}

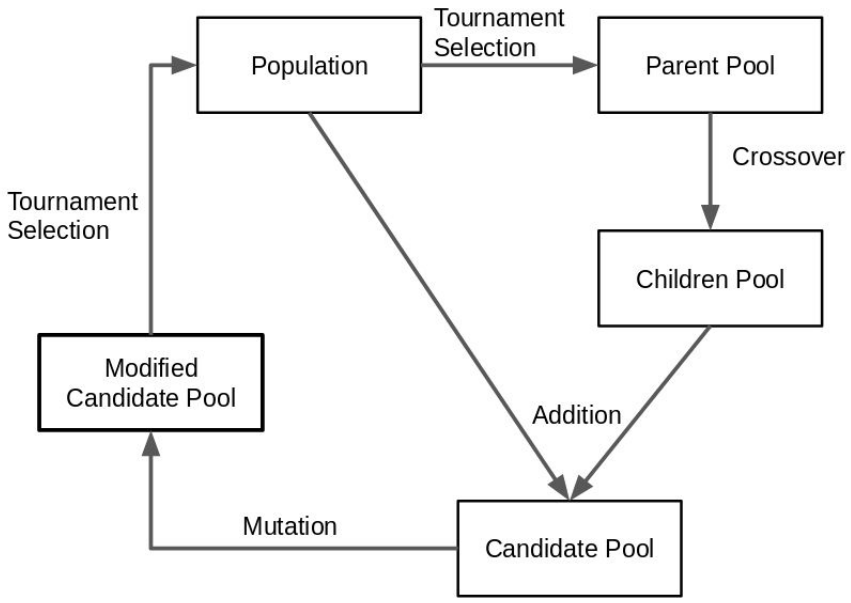
B) Representation and Methodology

1. Representation of Sudoku Puzzle
    a. I represented each Sudoku puzzle as a sqrt(N) by sqrt(N) matrix of blocks, where each block consisted of sqrt(N) by sqrt(N) elements. For example, if we wanted a 9 by 9 sudoku puzzle. I would have a 3 by 3 matrix of blocks that each consisted of a 3 by 3 matrix of elements. I did this because as per the instructions on the homework, the puzzle was to be initialized with each block being a set of numbers between 1 and N. This would mean that one of the three rules of Sudoku would be satisfied and I made all operations work on a block level so that a block would never stop satisfying the rule.
2. Fitness Function
    a. For a fitness function, I implemented what I call a fitness map. This looked at each element in the puzzle and gave it 1 point for each of the three rules that it

satisfied. The total fitness for a puzzle could be calculated by taking the sum of the fitness map.

b. The fitness map is also used in the visual representation of the puzzles, where each element is colored based off of its fitness map value.
    i. Black = 0 (Does not appear because all blocks are initialized as sets)
    ii. Red = 1
    iii. Blue = 2
    iv. Green = 3 (meaning that this element satisfies all rules)

3. General Flow of the GA



4. Crossover Operation
    a. Crossover is implemented by creating a parent pool from the population by tournament selection. Sets of parents are then chosen at random and produce offspring by performing a single point crossover at the block level. The children are then added to the children pool. The children pool is grown up to the population size every generation.
5. Candidate Pool
    a. To increase the diversity in the population the current population and all of the children are pooled together into a candidate pool. This larger pool is then mutated and another tournament is ran to select the next generation with the most elite individual guaranteed a spot in the next generation.
6. Mutation Operation
    a. Mutation is implemented by assigning each individual in the candidate pool a random value. If this value is less than the current mutation rate, then a random block is selected and the value in it are shuffled.
    b. The mutation rate starts low, but can increase when the best fitness in the population does not increase for X generations. This can happen repeatively, if

there is no improvement for many generations. The mutation rate is capped at 90% and only gets this high if the evolution process has stalled for a very long time.

i. This is used to force the population out of local optimum.
ii. When an individual is found with a fitness better than the previous best, the mutation rate is greatly reduced to give that individual a chance to reproduce.

C) Implementation

I implemented this project in python using the Tkinter package to implement the GUI. Plots were generated using Matlab.

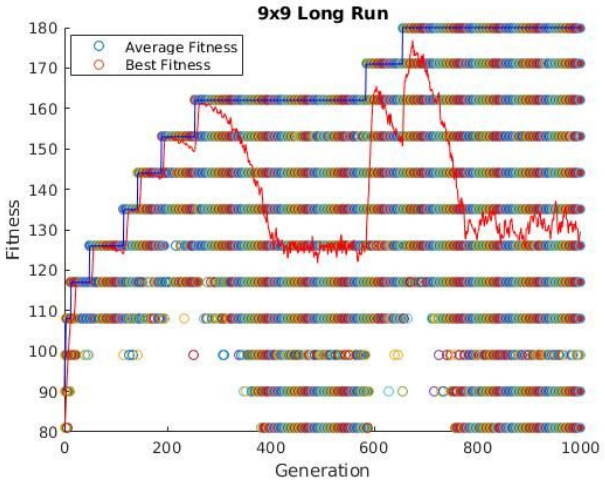
The complete code can be found at:

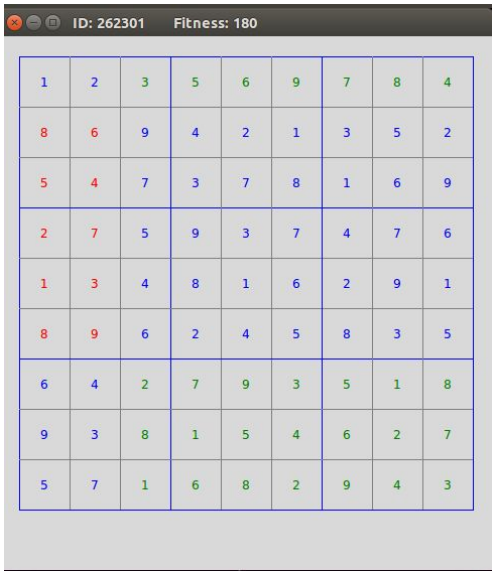
https://github.com/gsimon2/Sudoku_GA

D) Evaluation

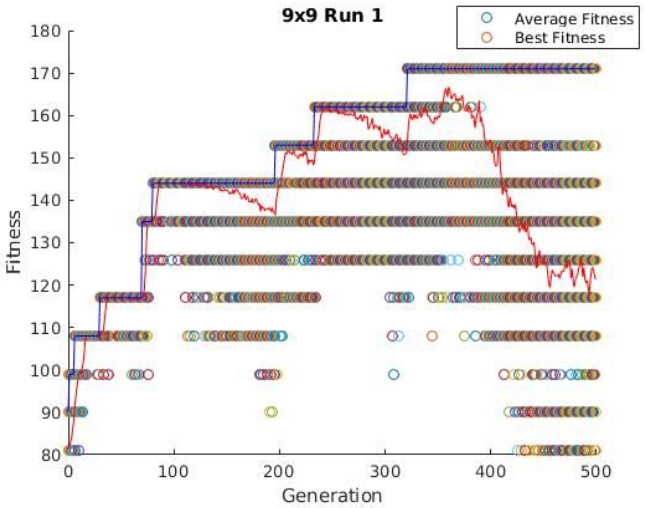
9 by 9 Sudoku Puzzle						
Run Name	Pop Size	Generation Limit	Tourn Size	Base Fitness	Best Fitness	Optimal Fitness
long_run	400	1000	2	81	180	243
run_1	200	500	2	81	171	243
run_2	200	500	2	81	144	243
run_3	200	500	2	81	162	243
run_4	200	500	20	81	162	243
run_5	200	500	20	81	180	243

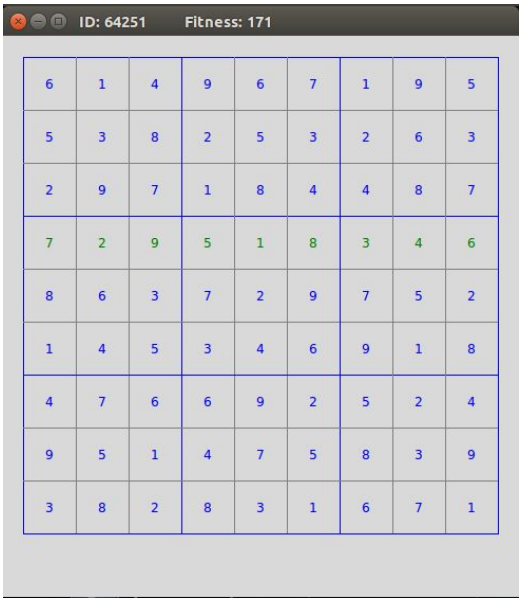
1. Long Run Analysis
    a. In the long run, there is a good rate of improvement to fitness for the first 250 generations. After which only two more improvements are made while in general the average individual of the population gets worse.
    b. As seen on the right, the best individual from generation 1000 was able to have a fair number of elements which satisfied all three rules, but was unable to completely solve the puzzle.
    c. Runs were made at over 1000 generations, but no improvements were seen.



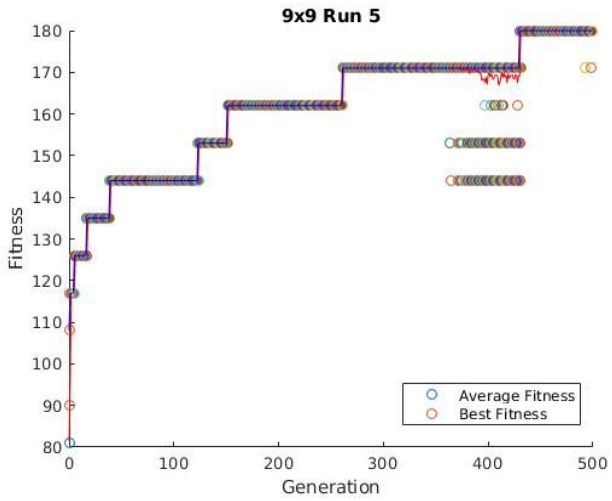


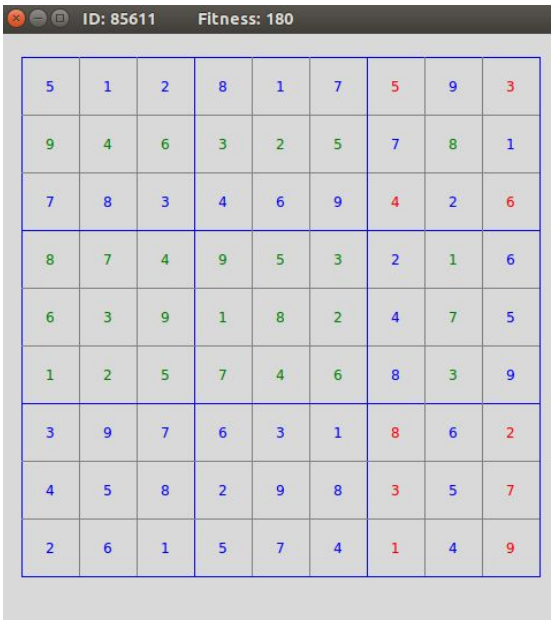
2. Run 1 Analysis
    a. Run 1 shows gradual improvement of the best fitness, which is preserved due to elitism being built into the GA. However, as the improvements to fitness become stagnant around generation 400, the average fitness drops greatly due to the mutation probability increasing forcing the population to explore the solution space.
    b. On the right is the elite individual in generation 500. You can see that all elements were able to satisfy two of the three rules and one row was able to satisfy all three rules.





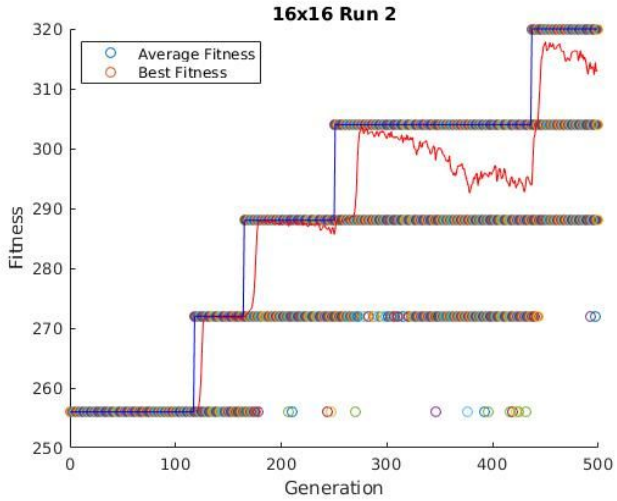
3. Run 5 Analysis
    a. In run 5, I increased the size of the tournament for the selection process in both the parents and for the next generation. This was in effort to force the population to be more like the most elite individual. As can be seen on the left, there is much less diversity in the population until improvement stagnates for a long time and the raise in mutation applies pressure for a more diversified population.

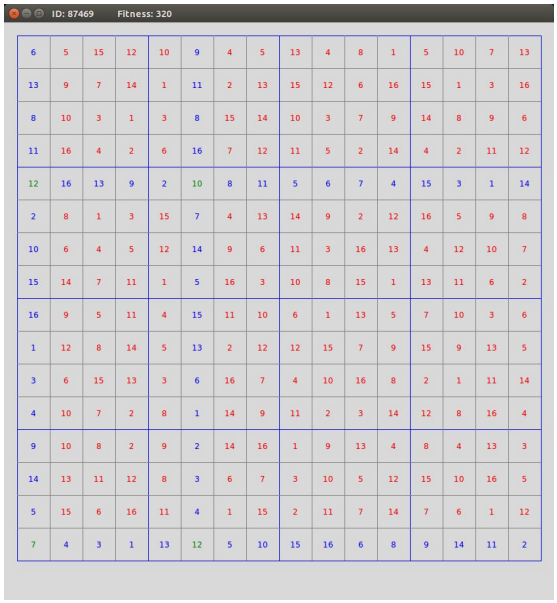




16 by 16 Sudoku Puzzle						
Run Name	Pop Size	Generation Limit	Tourn Size	Base Fitness	Best Fitness	Optimal Fitness
run_1	200	500	2	256	288	768
run_2	200	500	2	256	320	768
run_3	200	500	2	256	304	768

1. Run 2 Analysis
    a. With a larger dimension Sudoku puzzle, results take more generations to appear. This could be because the mutation operator is only affecting one block in the puzzle. As the dimension size increase, the number of blocks also increase resulting in mutation affecting less of the puzzle and slowly results.





E) Adaption of System to Solve Real Sudoku Puzzles

This system could fairly easily be adapted to solve real Sudoku puzzles. Instead of initializing the population with random numbers, a user could provide a representation of a real puzzle with only a few numbers filled in. The system would label these original numbers and prevent them from ever being moved over the course of evolution. The crossover operator could almost stay the same, except for leaving blocks that had original numbers in them. The mutation operator would have to be adapted to choose to either randomly fill in blank spaces, shuffle non-original numbers in a block, or to swap non-original filled in elements. I believe with these changes, the system could eventually solve real Sudoku puzzles.

F) Post Notes

- My system can handle any perfect square dimension size puzzles. I tested up to 25 by 25, but they took a considerable amount of time to run and do analyse on. The results were much like the 16 by 16, where the growth of the best fitness is elongated across more generations.
- I was never able to find an optimal solution for a puzzle, even after a lot of tweaking of the GA. If another student was able to, could you please either share their methods with the class or let me know? I am curious to see what can be done to solve this problem.