Michael Hering
CSCI 3202

Assignment 2 Write up

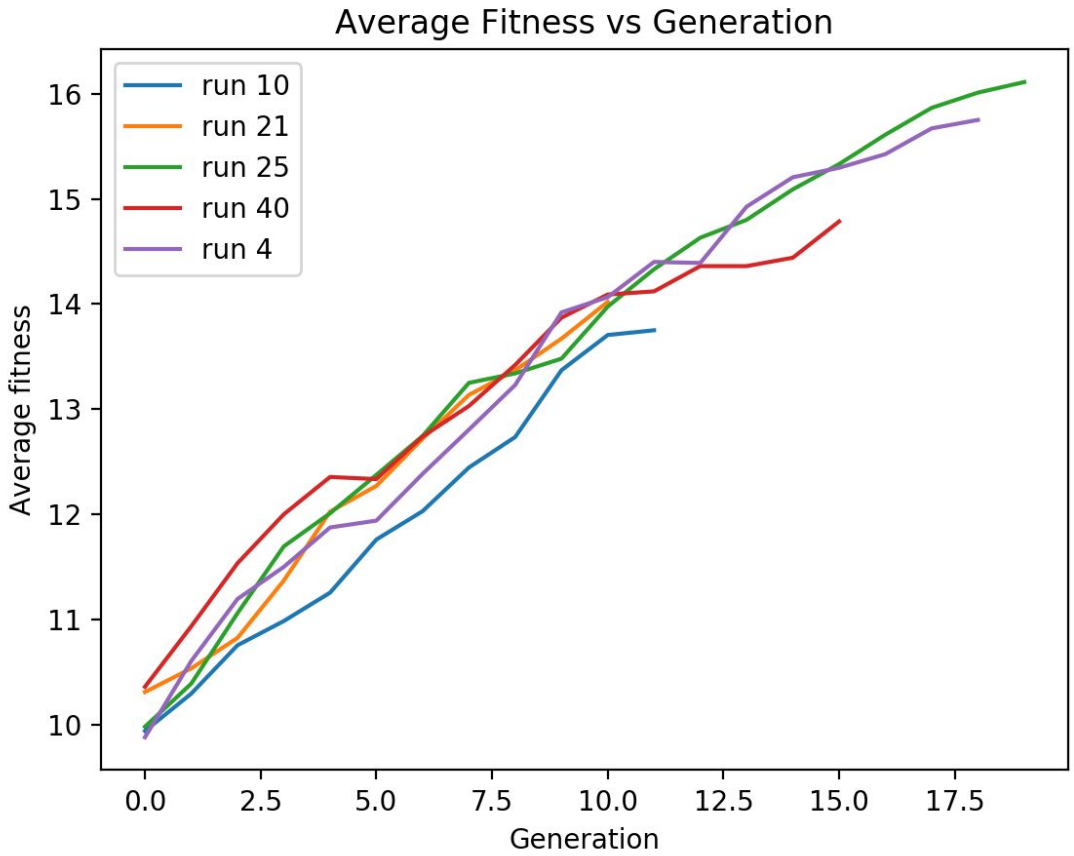
Part 1

1.

Running the GA 50 times with the parameters: population size = 100, single-point crossover rate = 0.7, bitwise mutation rate = 0.001, the algorithm discovered the optimal string at a minimum of 7 generations, a maximum of 32 generations, and an average of 18 generations.

2.

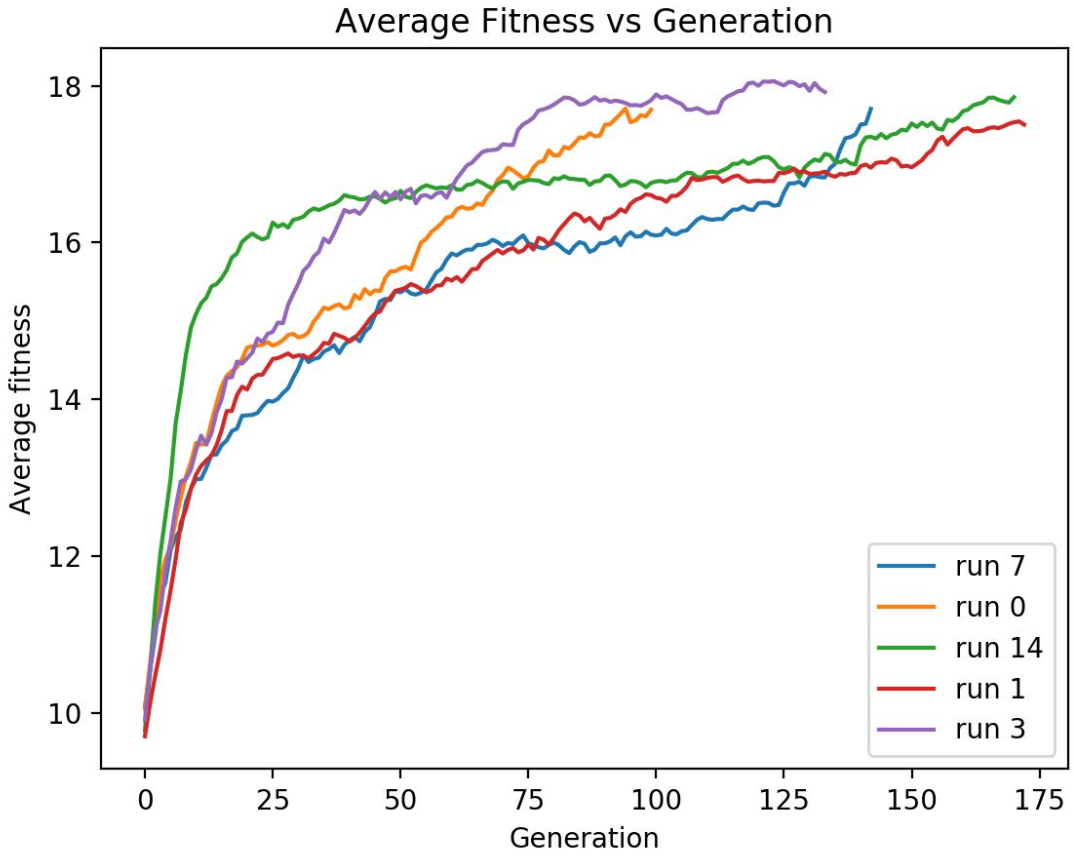
The plot below portrays the average fitness of 5 randomly selected runs of the algorithm, with the same parameters as above, over time (with each new generation).




Looking at the plot, the average fitness of all of the runs of the algorithm shown here progress at a similar, and linear, rate. Additionally, the average fitness is usually increasing over time. I also notice that in run 10, the average fitness is still very low (13.75) when the algorithm finds the optimal solution, compared to runs 4 and 25.

3.

The plot below portrays the average fitness of 5 randomly selected runs of the algorithm, with the same parameters as above excluding the crossover rate which has been set to 0, over time (with each new generation).

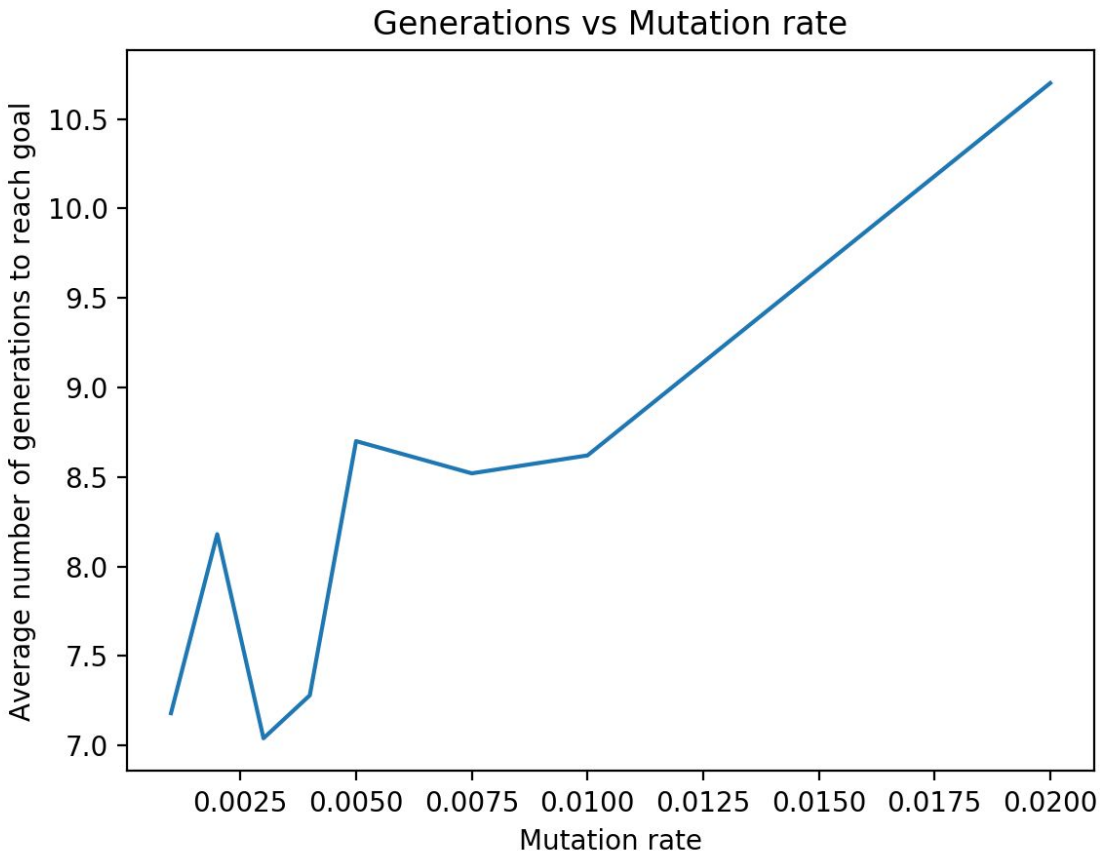



Changing the crossover rate to 0 seemed to drastically increase the number of generations that the algorithm took to find the optimal solution. Even though the average fitness will increase over time, due to more fit parents remaining from generation to generation, with the crossover rate set to 0 new populations must rely solely on mutation to find the optimal solution. Because the chance of mutation is small, the algorithm will take longer to converge and relies on selection without crossover and random chance to find the solution.

4.

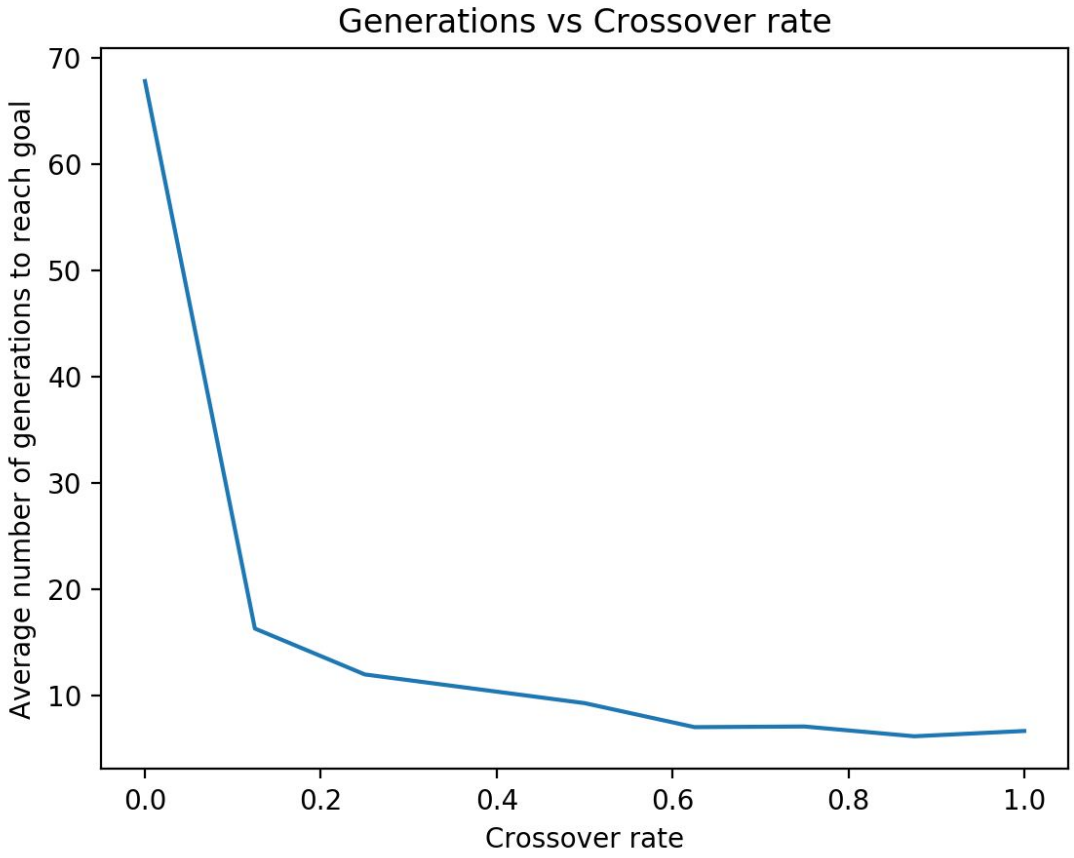
We will now explore how changing each of the parameters (mutation rate, crossover rate, and population size) affect the algorithm. We will only vary one parameter at a time through these experiments while holding the other parameters constant at their original values: population size = 100, single-point crossover rate = 0.7, bitwise mutation rate = 0.001.

First, we will explore how changing the mutation rate affects the algorithm by plotting the average number of generations the algorithm takes to find the optimal solution over 20 runs of the algorithm. We will use the following values for mutation rate:



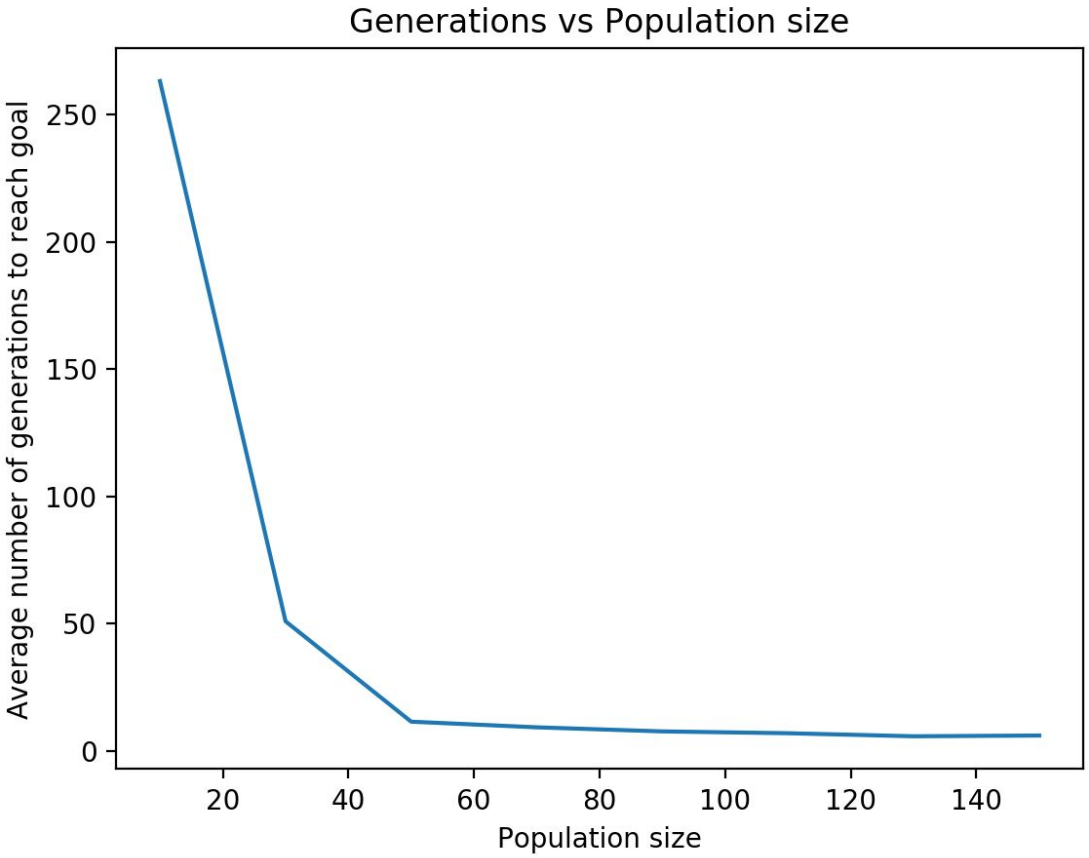

Looking at the plot, we can see that in general the average number of generations that the algorithm takes to find the optimal solution increases with the mutation rate. However, we notice optimal mutation rates of 0.001 and 0.003, with 0.002 being worse than both of these. Also note that while decreasing the mutation rate seems to help the algorithm, a mutation rate of 0 could cause the algorithm to never find the optimal solution if the starting population could not reach an optimal solution through crossover alone.

Next, we will explore how changing the crossover rate affects the algorithm by plotting the average number of generations the algorithm takes to find the optimal solution over 20 runs of the algorithm.




Looking at the plot, we can see that the average number of generations that the algorithm takes to find the optimal solution decreases as the crossover rate increases.

Finally, we will explore how changing the population size affects the algorithm by plotting the average number of generations the algorithm takes to find the optimal solution over 20 runs of the algorithm.



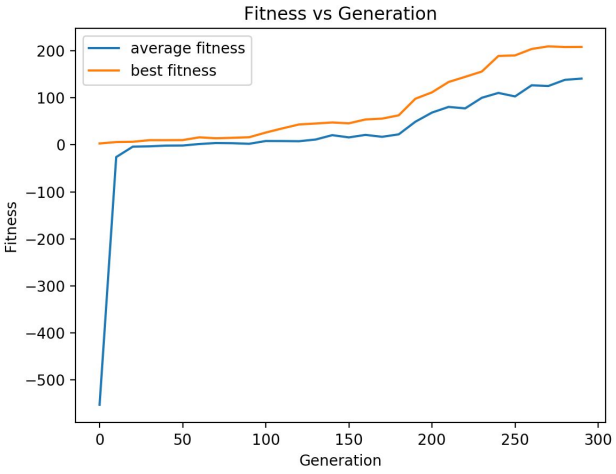

Looking at the plot, we can see that as we increase the population we drastically reduce the number of generations that the algorithm takes to find the optimal solution. It is important to note however that with too large of a population size we may end up finding the correct solution by complete chance rather than through the process of the genetic algorithm.

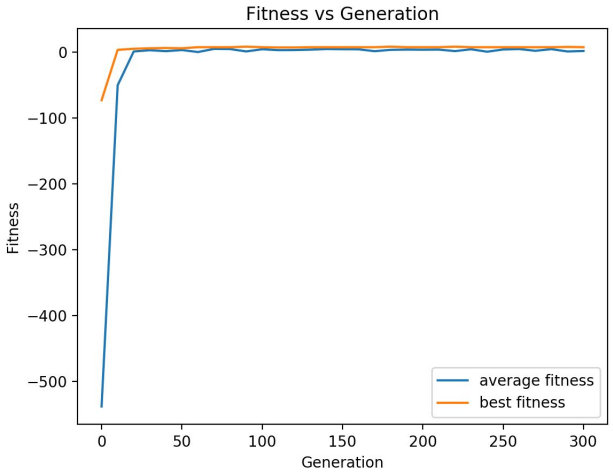
Part 2

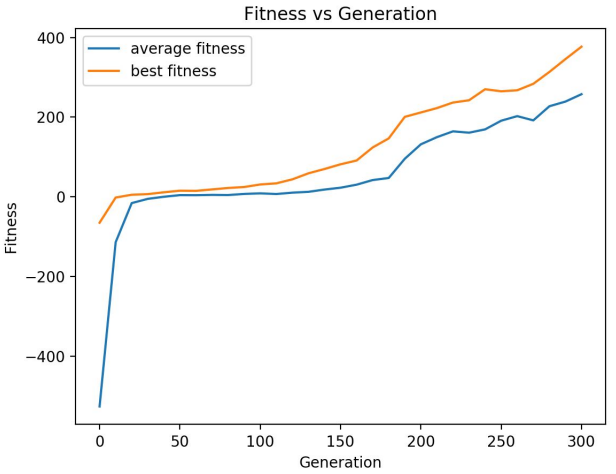
Based on the observations above and my test runs of this genetic algorithm - I will be testing my genetic algorithm to solve robby's world using 4 different runs, with the following parameters:

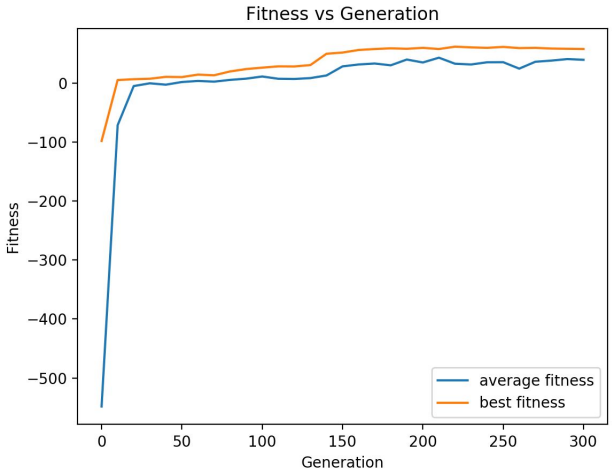
Run 1: population size = 100, single-point crossover rate = 1, mutation rate = 0.005.
Run 2: population size = 100, single-point crossover rate = 1, mutation rate = 0.003.
Run 3: population size = 140, single-point crossover rate = 1, mutation rate = 0.005.
Run 4: population size = 200, single-point crossover rate = 1, mutation rate = 0.005.

Below are plots of the average and maximum fitness of the populations in each of these runs over time. Run 1 (top-left), Run 2 (top-right), Run 3 (bottom-left), Run 4 (bottom-right)



Looking at the graphs above, we can see that run 3 produced the best results. I also notice that the fitness values grow to values around 0 to 70 very quickly, but then get stuck in this range until the fitness reaches over 100, at which point they progress consistently again to over 300. In these experiments, I chose to keep the crossover rate high at 1, as all of my tests from part1 showed that a high crossover rate leads to better results. The big factors that I chose to vary for part 2 were population size and mutation rate. Unlike my experiments in part 1, a mutation rate lower than 0.005 seemed to lead to worse results. Setting the mutation rate to 0.005, the next parameter that I experimented with was the population size and the lower population size of 140 led to a much better solution while also causing the algorithm to find a solution much faster.