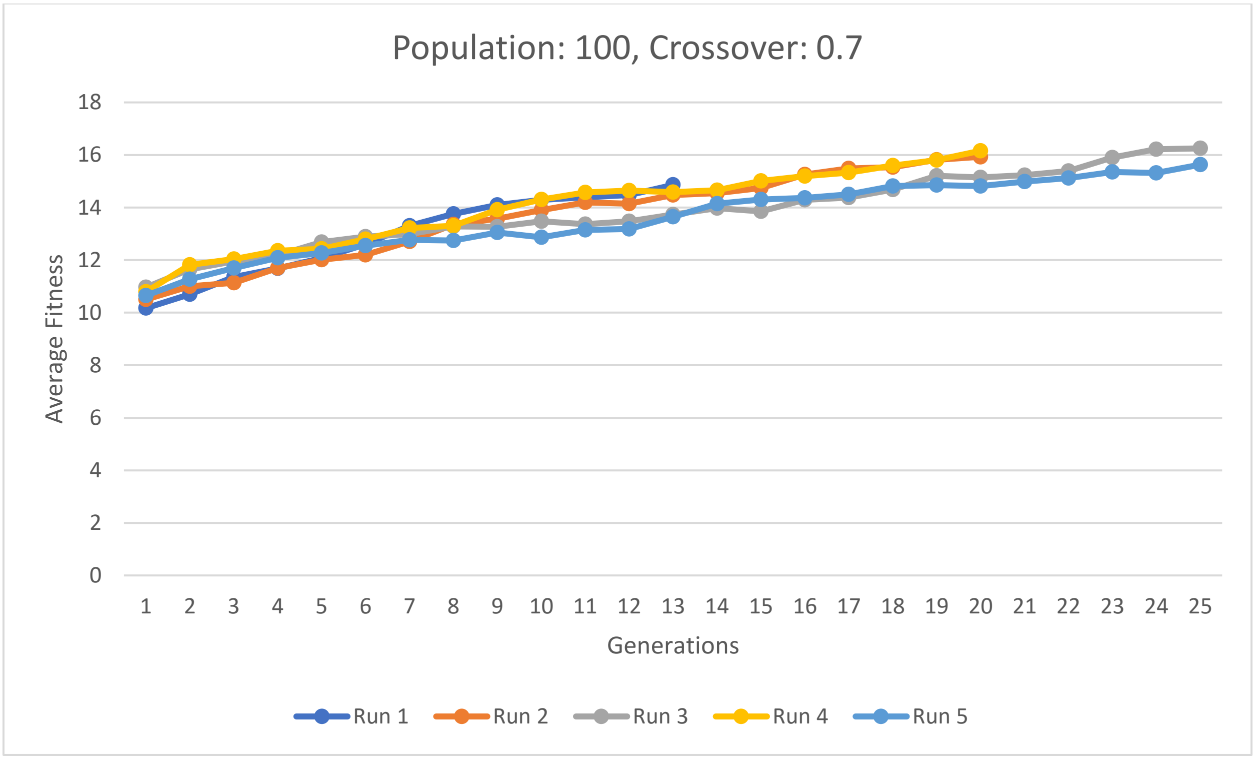
CSCI 3202

Trevor Buck

Assignment 2

Genetic Algorithm Report

1) Report:
    a. Maximum number of generations: 50
    b. Minimum number of generations: 5
    c. Average number of generations: 27.14
2)



3) Report with [Crossover = 0.0]:
    a. Maximum number of generations: 50
    b. Minimum number of generations: 50
    c. Average number of generations: 50.0
    d. With zero crossovers, this means that the only way the genes change is by selection and mutation. These both help, but they don't create any big changes. I noticed that as the runs continued, there was a gradual increase. This is because a weighted selection means that we are, on average, working with the better genes. However, we aren't really changing the genes. Mutation only affects one out of a thousand genes, and there

was a %50 chance that it hurt the gene, causing more of a random effect. These two operations (mutation and selection) are helpful in the overall product, but without crossover to significantly impact the genes, there will never be a high improvement rate.

4) Differing values:
    a. [Population = 10]:
        i. Maximum number of generations: 50
        ii. Minimum number of generations: 50
        iii. Average number of generations: 50.0
    b. [Population = 1000]:
        i. Maximum number of generations: 20
        ii. Minimum number of generations: 5
        iii. Average number of generations: 12.72
        iv. Note: Significantly longer time to complete
    c. [Crossover = 1.0]:
        i. Maximum number of generations: 46
        ii. Minimum number of generations: 8
        iii. Average number of generations: 22.3
    d. [Mutation = 0.1]:
        i. Maximum number of generations: 50
        ii. Minimum number of generations: 8
        iii. Average number of generations: 48.54
    e. Note: I did lots of other experiments, but these are the more notable ones
    f. Report:
        i. Varying the Population: The basic principle here is that the greater number of populations that you have, the quicker you will find/create one that has twenty '1's. However, there is a tradeoff here in time. Our algorithm is of 0(n^2) time where 'n' is the population.
        ii. Varying the Crossover: As mentioned before, if the crossover value is too low, there isn't enough to change to produce a gene with twenty '1's. However, when we increased the crossover rate to 100%, there was too much change, and the accuracy dropped. This means that you need find the sweet spot, which in this case, was right around that 70% mark.
        iii. Varying the Mutation: Because mutation has a 50% chance of hurting our gene, we need the rate of mutation to be low. A low value gives us some randomness, but doesn't hurt our average score.

Part 2: Robby's World

1) With my experiments, I tried to vary the parameter settings one at a time. I noticed that as I increased the population, I had very similar results to what happened in the first part of this assignment. While the overall score of the algorithm improved, the time it took to complete the algorithm was significantly higher. Granted, I don't think I have a very fast computer, but the time complexity will hold true for all computers, regardless of the specifics. Meaning that the more you increase the population, the longer it will take for the algorithm to complete. The next parameter that I did some experiments with was the mutation. There wasn't much change in the algorithm when the mutation was in the range (0.1 -> 0), but once you passed the 0.1 mark, the randomness of the algorithm increased. In the first couple of generations, the "best" fitness for each round was a little higher, but the overall average was a little lower. Once you got into later generations, the randomness seemed to hurt a lot more. You would have random, somewhat significant dips in the average fitness. After this, I started messing with the number of actions per cleaning section. This parameter was one of the more surprising factors for me. As I decreased the number of actions, my beginning sections got better. However, I realized that this is just because there aren't as many opportunities to run into the wall. In the later generations, I didn't get a very good "best fitness". And this is because there just aren't as many opportunities to increase your reward. In summary, decreasing this parameter lead to a more general score. Not as low *bad* scores, and not as high *good* scores. Last, but not least, I changed the crossover rate. Once again, the results were extremely similar to those found in part 1. Having the crossover rate at 1 helped the program quite a bit in the earlier generations. It led to quick changes and lots of improvement. However, in the later generations, it didn't do much different than say a 0.7 rate. If anything, it hurt the algorithm. In the end, the best results I got was with the [crossover rate = 0.9] and the [mutation rate = 0.005], with [actions = 200]. You'll notice that I didn't include the population size or the number of generations. That is because they both follow the rule, the more the merrier. The highest I ran the algorithm was with [population = 200] and [generations = 300], but that's not the limit. I believe that had I increased those two parameters, I would've achieved a better score. However, that is as far as I pushed the limits in this assignment.