Genetic Algorithms

Samuel Leonard

COSC 420

April 8, 2019

Introduction:

Genetic Algorithms model evolution by applying natural selection. They are used to find an optimum solution to various problems. Genetic Algorithms utilize a fitness function to quantify potential solutions. The individuals with higher fitness results will be selected more often. This leads to spawning generations of higher and higher fitness scores, and better solutions. The purpose of this document is to demonstrate the behavior of Genetic Algorithms using software, datasheets, and graphs.

Theory:

Genetic Algorithms work off natural selection. They should select the best individuals from a given population to spawn equal or better offspring. This leads to certain questions about the outcomes of Genetic Algorithms. What does an optimum solution look like? In this case, an optimum solution should be all 1's. What will happen to the average fitness score of a population with each new generation? If only the best individuals are being selected, then the average fitness score should continue to increase with each new generation. What fitness score will the best individual have? As each generation passes, the fitness score of the best individual should get better and better until the score is very close to one. What affect will there be on the number of correct bits in a population when the best individuals are selected more often to produce offspring? The number of correct bits should increase since the higher fitness individuals have more correct bits. To prove this behavior, several methods and calculations needed to be performed.

Methods and Calculations:

First, a program was needed to simulate the behavior of a basic Genetic Algorithm. Python was chosen to be the language of the program because of its math and scientific libraries. The program would take inputs for the length of the individuals (number of bits in a bit string), the size of the population (number of bit strings), the number of generations to be spanned, the probability of mutation, and the probability of crossover. Next, the program would randomly populate the bit strings with 1's and 0's using a random number generator. After the bit strings were populated, the fitness scores for each individual was calculated using the formula below in **Equation 1**.

$$F(s)=(x/2^{l})^{10}$$

Equation 1: Fitness Function

Then these scores were normalized, and a running total of these normalized values were stored. Selection of the parents took place next. The program generated two random numbers, so it could randomly choose two parents based on their running total values. The closest number higher than the random number was chosen as a parent. Once the parents were chosen, a random number would be generated to determine if there was crossover between the parents to spawn offspring. The program would check to see if the random number was below the probability of crossover which was one of the command line inputs. If there was crossover, then single point crossover was performed by randomly generating an index in the bit string for the bits of one parent to be copied up to, and then filling the remaining bits with the second parent. This was reversed for the second offspring. An example of what single point crossover looks like is demonstrated below in **Figure 1**.

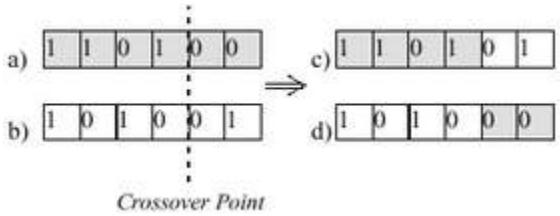

Figure 1: Single-Point Crossover Example

If crossover was not performed, then the two offspring would just be copies of the parents. One child would inherit all bits from one parent and the other child would inherit from the second parent. After establishing crossover or no crossover, the next step to be performed was to check for mutation of the bit strings. The program would check each bit in the bit strings to see if mutation of the bit (flipping the bit) occurred. A random number again would be obtained and if that number was below the probability of mutation, then mutation of the bit would happen. If the original bit was a 1 then the mutated bit would now be a 0 and vice versa. An example of mutation is demonstrated in **Figure 2** below.

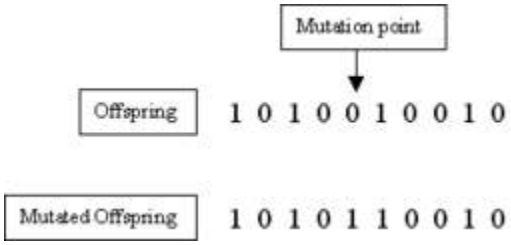

Figure 2: Bit Mutation Example

After the bit strings were processed for possible mutation, the program added the two new bit strings to an array. The program continued to produce new offspring until the number of new individuals matched the number of old individuals. Then, the new population replaced the old

population in the program. This constituted a new generation. The program would continue to produce new generations until it met the number inputted on the command line. For each generation, the software determined the average fitness of the population, the best fitness score for an individual in the population, and the average number of correct bits (number of 1's) in the population. The software would also create a .csv file for the data produced and create graphs representing the average fitness, best fitness score, and average correct bits for each generation. Below are some relevant graphs demonstrating the Genetic Algorithm's behavior.

Graphs:

The graphs below in demonstrate the behavior of a basic Genetic Algorithm. The parameter inputs are varied to show how they affect the algorithm. The graphs have been generated by the data produced in the Genetic Algorithm program which has been saved in a .csv file. The raw data can be found there.

The first graphs below in **Figures 3, 4, and 5** show the behavior of the Genetic Algorithm with the recommended starting parameters. The string length **(l)** is 20, the population size **(N)** is 30, the number of generations **(G)** is 10, the probability of mutation **(Pm**) is 0.033 which is the inverse of the population, and the probability of crossover **(Pc)** is 0.6. The graphs below show the average fitness, the best fitness, and the average number of correct bits with each passing generation.

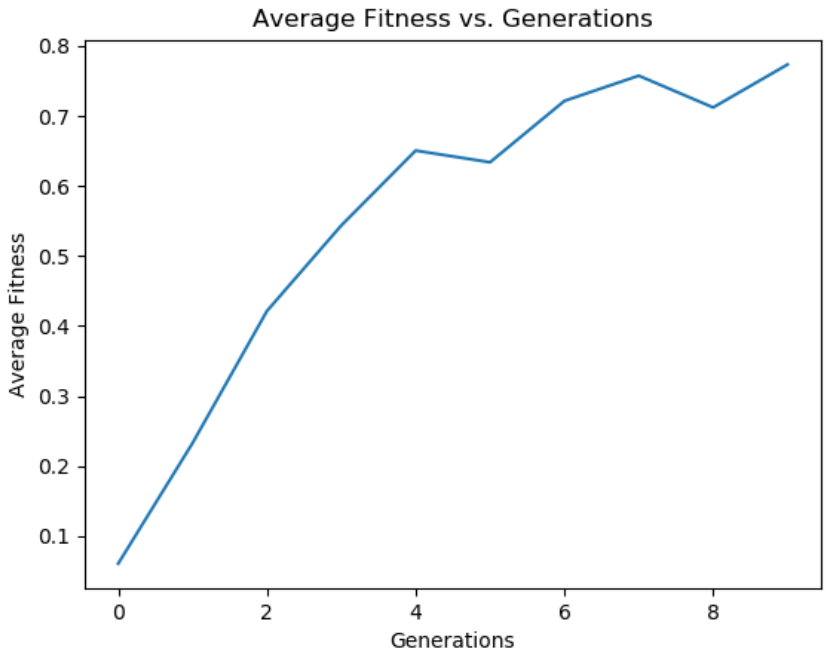

Figure 3: Average Fitness Graph for l=20, N=30, G=10, Pm=0.033, Pc=0.6

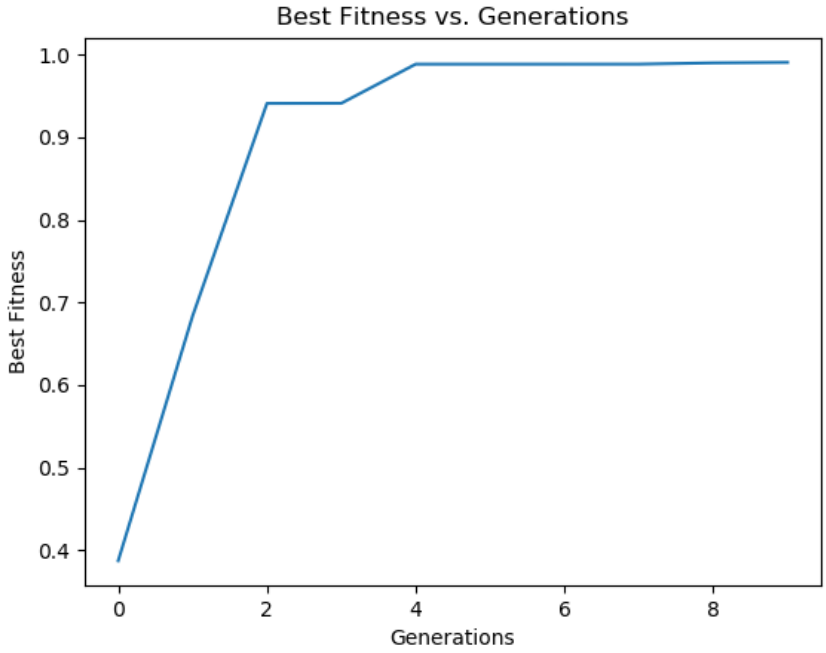

Figure 4: Best Fitness Graph for l=20, N=30, G=10, Pm=0.033, Pc=0.6

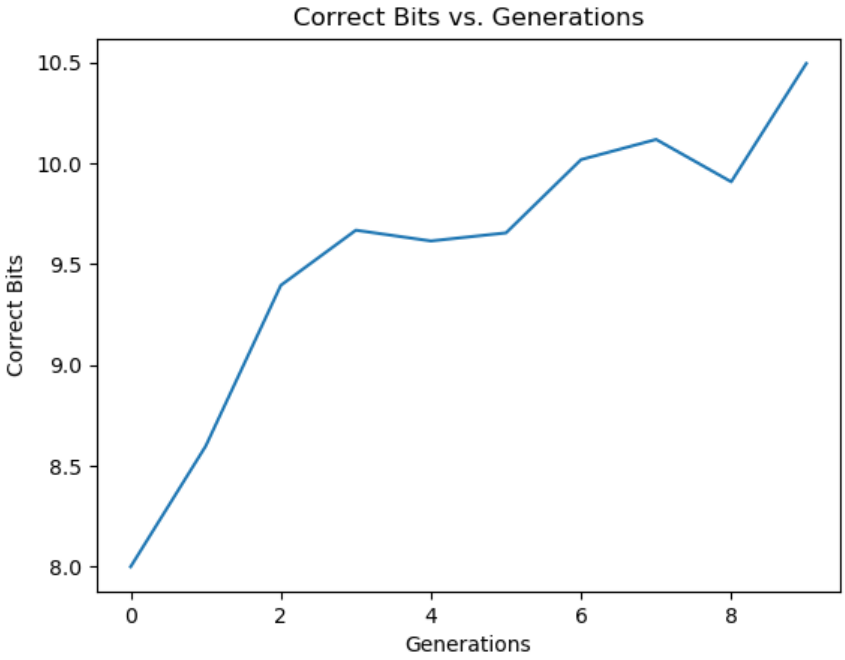

Figure 5: Average Correct Bits Graph for l=20, N=30, G=10, Pm=0.033, Pc=0.6

The next sets of graphs show the behavior as one of the parameters is varied while the others are held constant. This should give a good indication as to how each parameter affects the Genetic Algorithm. The first parameter to be varied will be the string length. The graphs below in **Figures 6, 7, and 8** will show how reducing and increasing the string lengths change the solution of the Genetic Algorithm as each new generation is spawned. The string length below is 10.

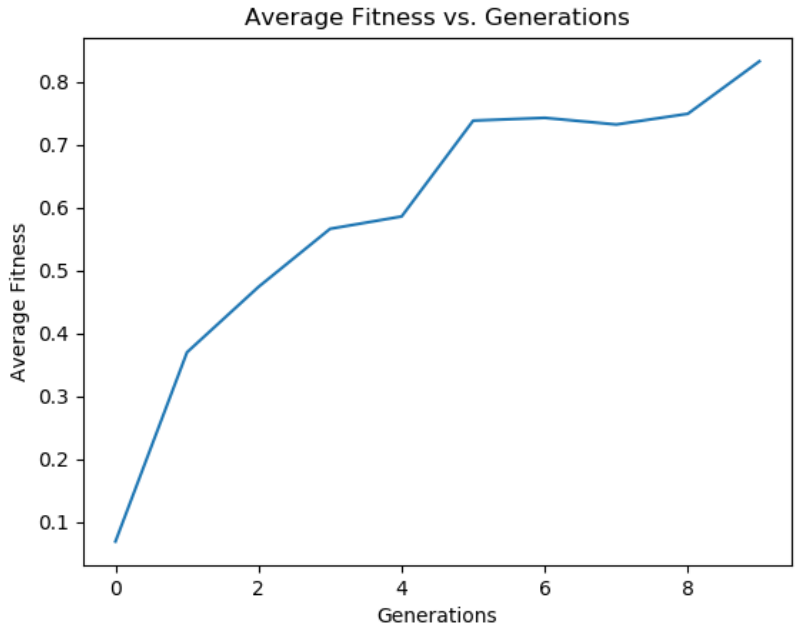

Figure 6: Average Fitness Graph for l=10, N=30, G=10, Pm=0.033, Pc=0.6

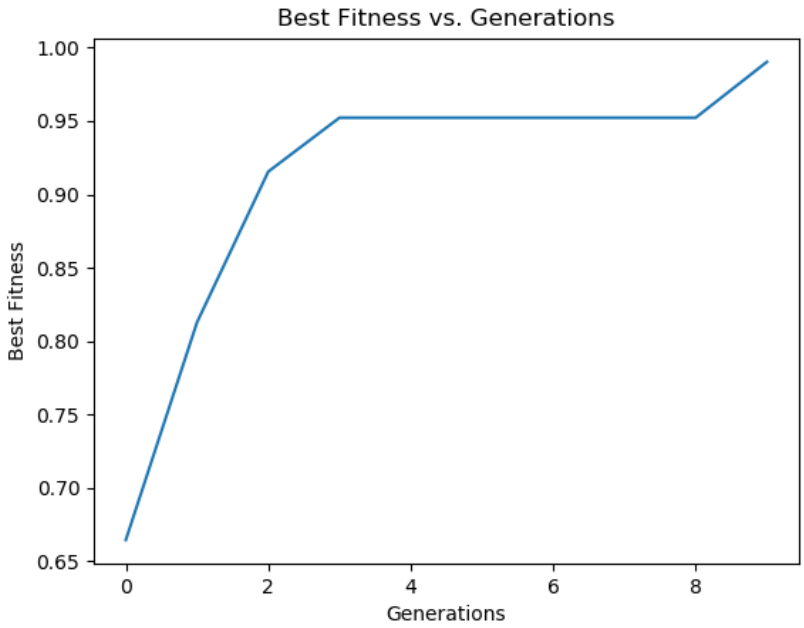

Figure 7: Best Fitness for l=10, N=30, G=10, Pm=0.033, Pc=0.6

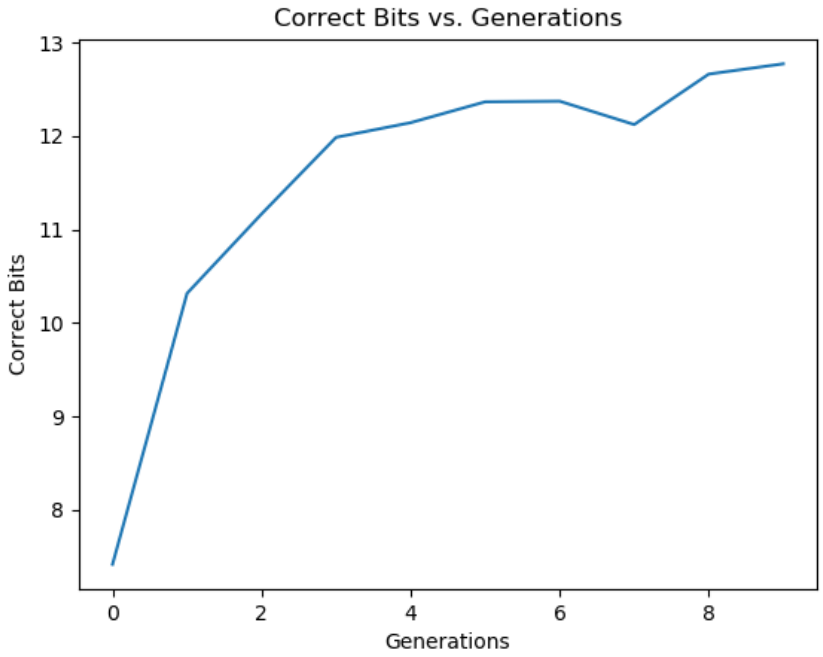

Figure 8: Average Correct Bits for l=10, N=30, G=10, Pm=0.033, Pc=0.6

The graphs below in **Figures 9, 10, and 11** show the average fitness, the best fitness, and the average correct bits produced when the string length has been increased to 40.

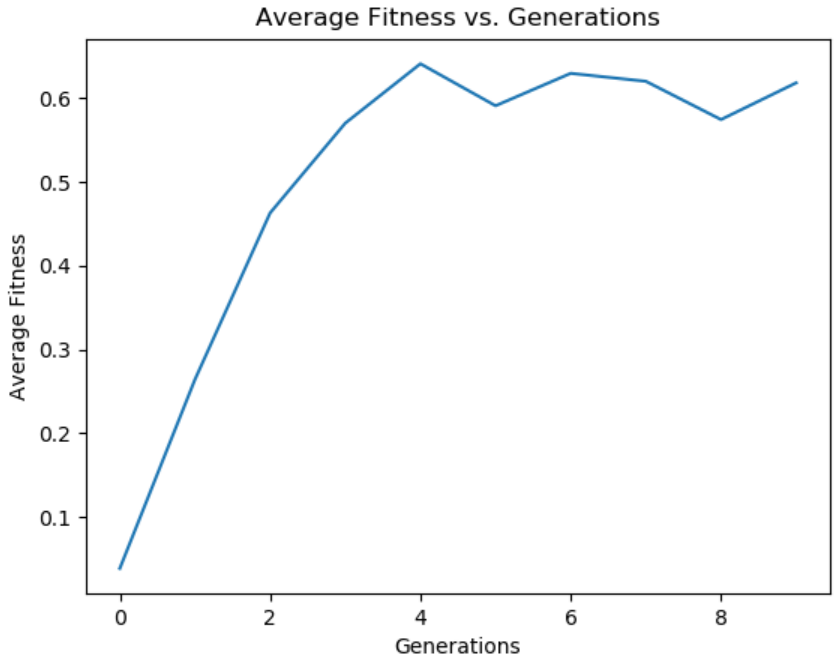

Figure 9: Average Fitness Graph for l=40, N=30, G=10, Pm=0.033, Pc=0.6

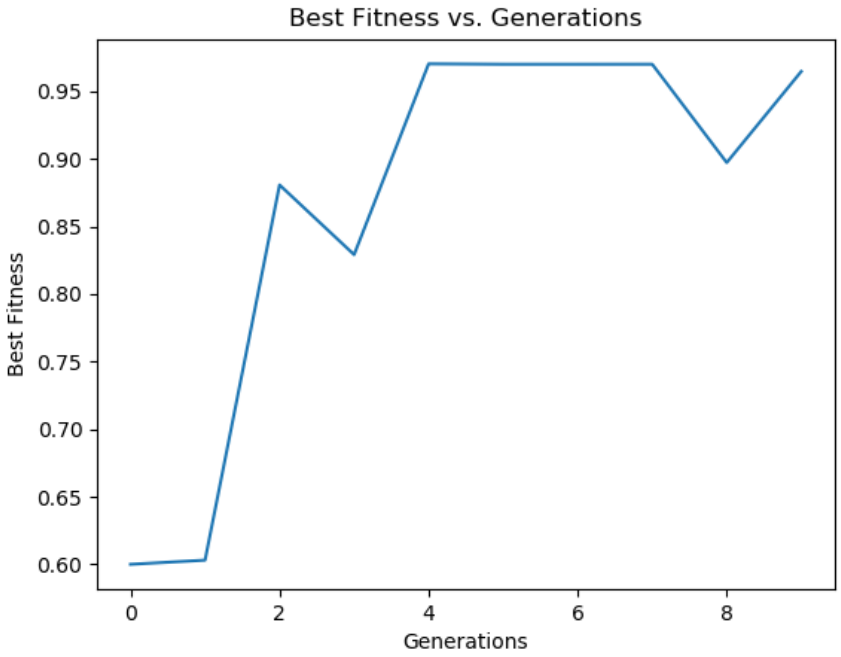

Figure 10: Best Fitness Graph for l=40, N=30, G=10, Pm=0.033, Pc=0.6

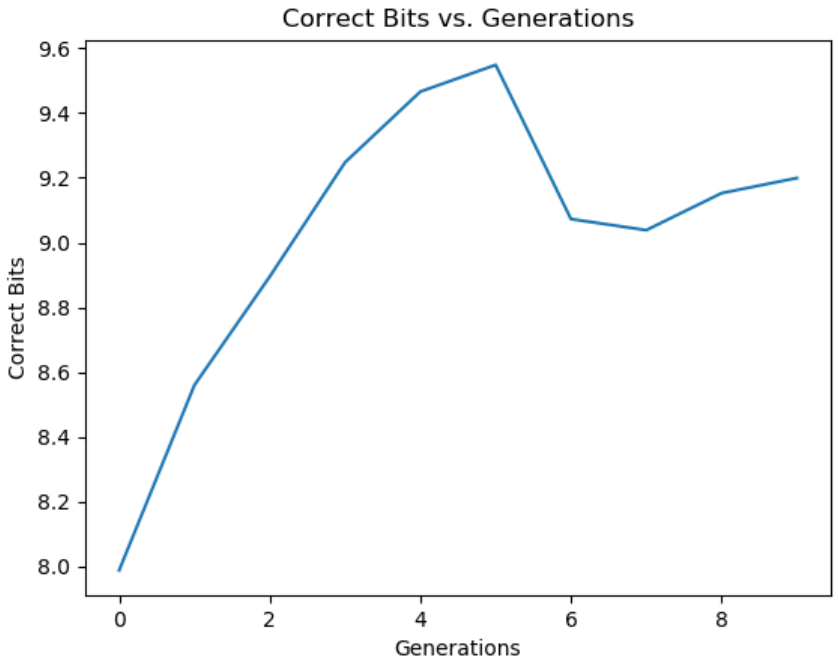

Figure 11: Average Correct Bits Graph for l=40, N=30, G=10, Pm=0.033, Pc=0.6

The graphs below show the outcomes when changing the number of generations while keeping the other parameters at the initial values. The graphs in **Figures 12, 13, and 14** have the number of generations increased to seventy.

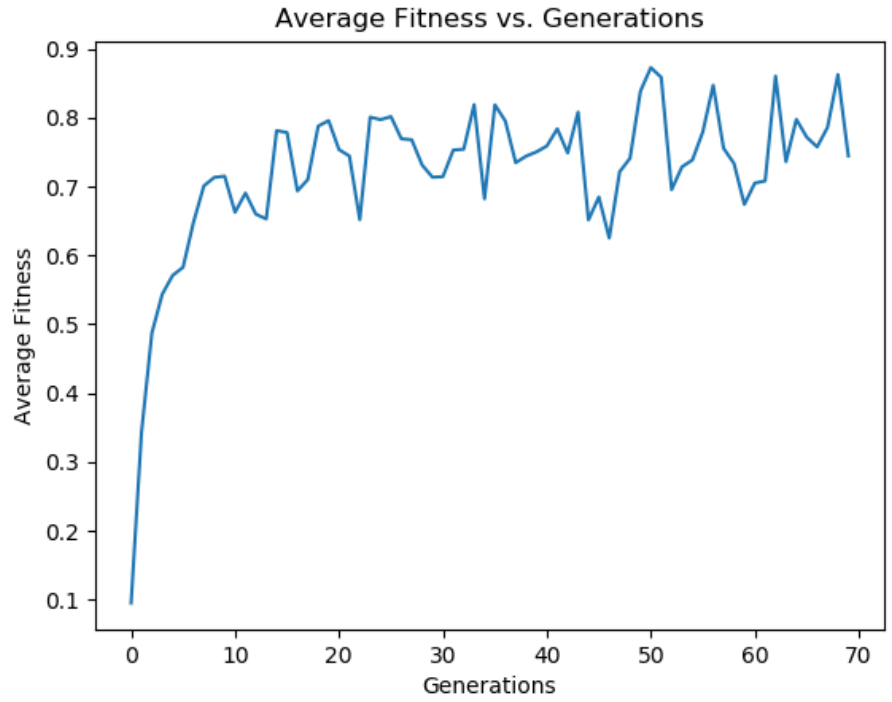

Figure 12: Average Fitness Graph for l=20, N=30, G=70, Pm=0.033, Pc=0.6

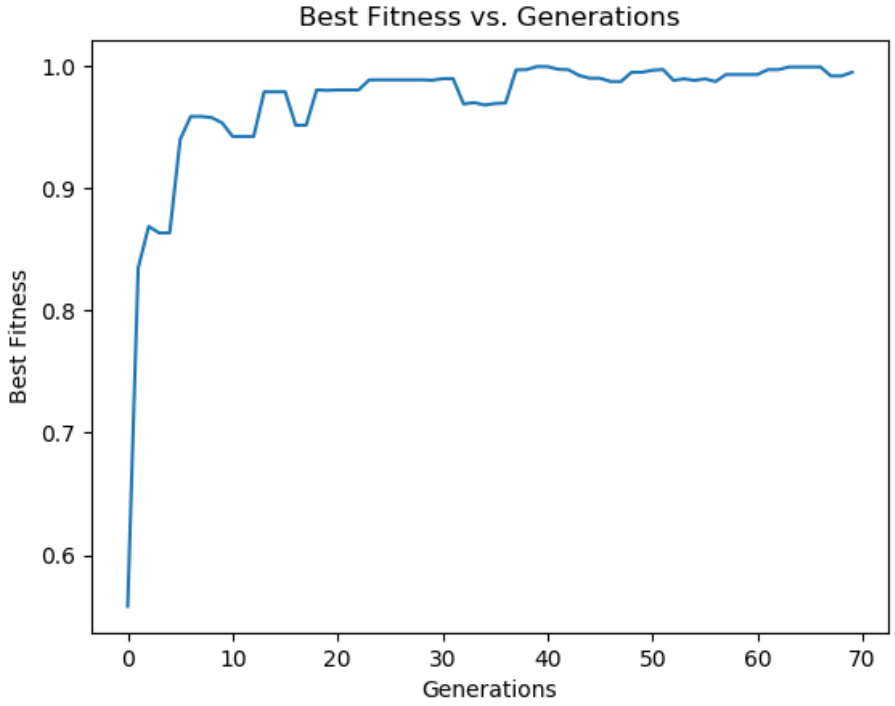

Figure 13: Best Fitness Graph for l=20, N=30, G=70, Pm=0.033, Pc=0.6

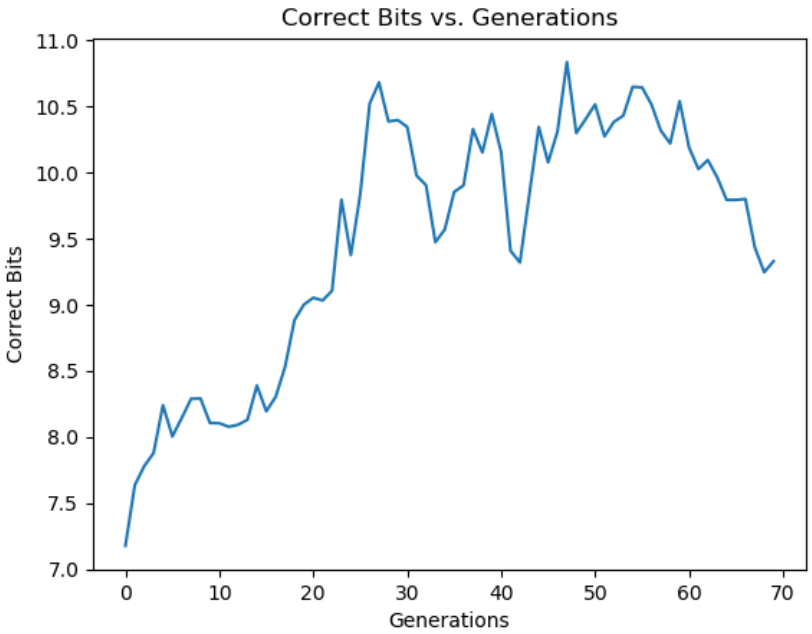

Figure 14: Average Correct Bits Graph for l=20, N=30, G=70, Pm=0.033, Pc=0.6

It is also important to look at how the Genetic Algorithm behaves when the population size is increased or decreased. The graphs below show what happens to the average fitness, the best fitness and the average correct bits when the population size has been changed. **Figures 15, 16, and 17** have the population size set to 10. It is important to note that the probability of mutation must be changed to comply with the inverse of the population rule.

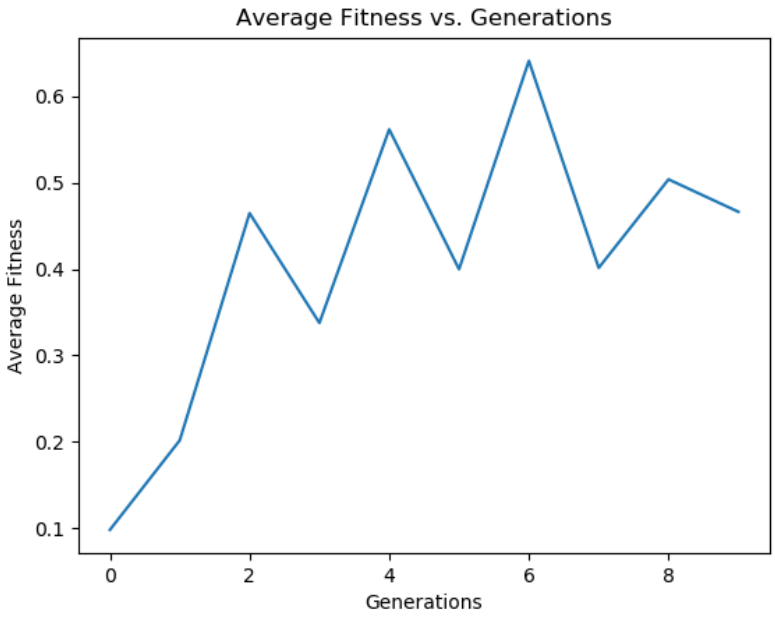

Figure 15: Average Fitness Graph for l=20, N=10, G=10, Pm=0.1, Pc=0.6

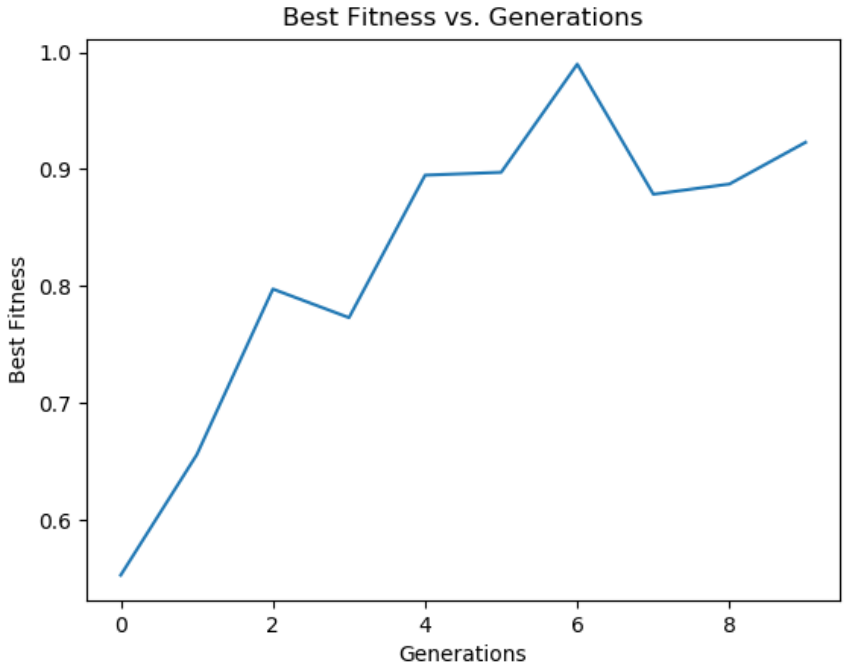

Figure 16: Best Fitness Graph for l=20, N=10, G=10, Pm=0.1, Pc=0.6

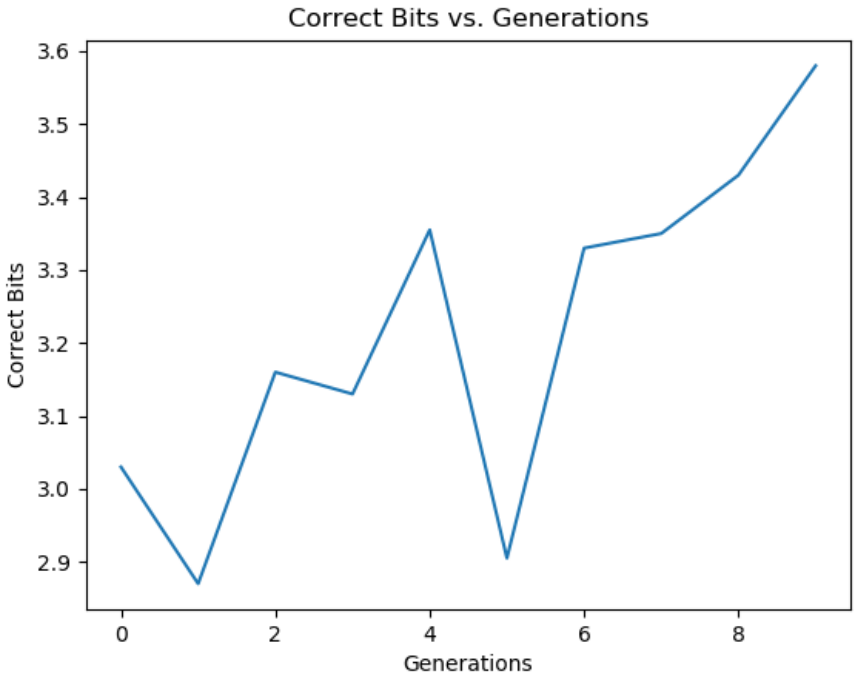

Figure 17: Average Correct Bits for l=20, N=10, G=10, Pm=0.1, Pc=0.6

The population size for the next set of graphs below in Figures **18, 19, and 20** is 50. The probability of mutation is set to 0.02.

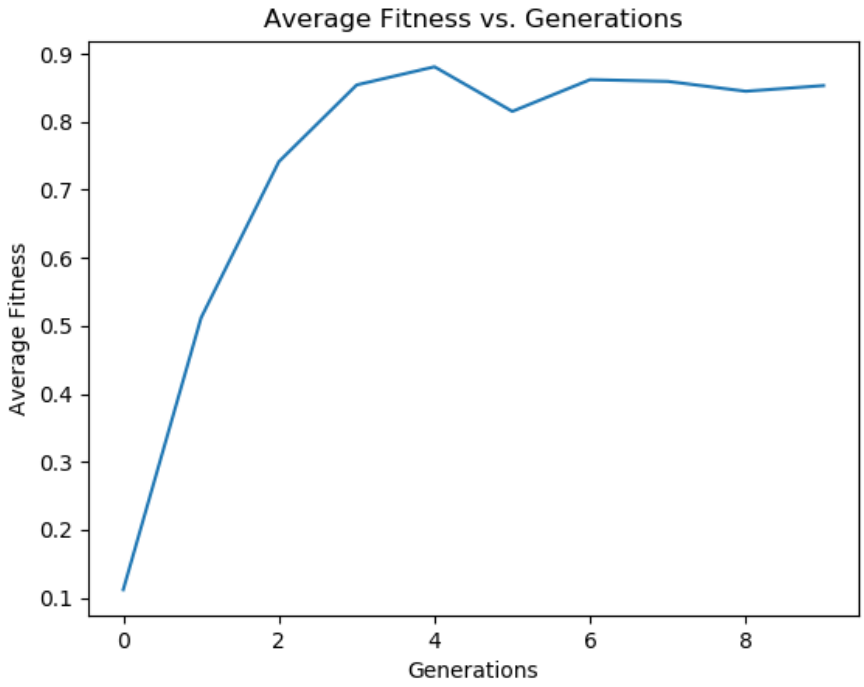

Figure 18: Average Fitness Graph for l=20, N=50, G=10, Pm=0.02, Pc=0.6

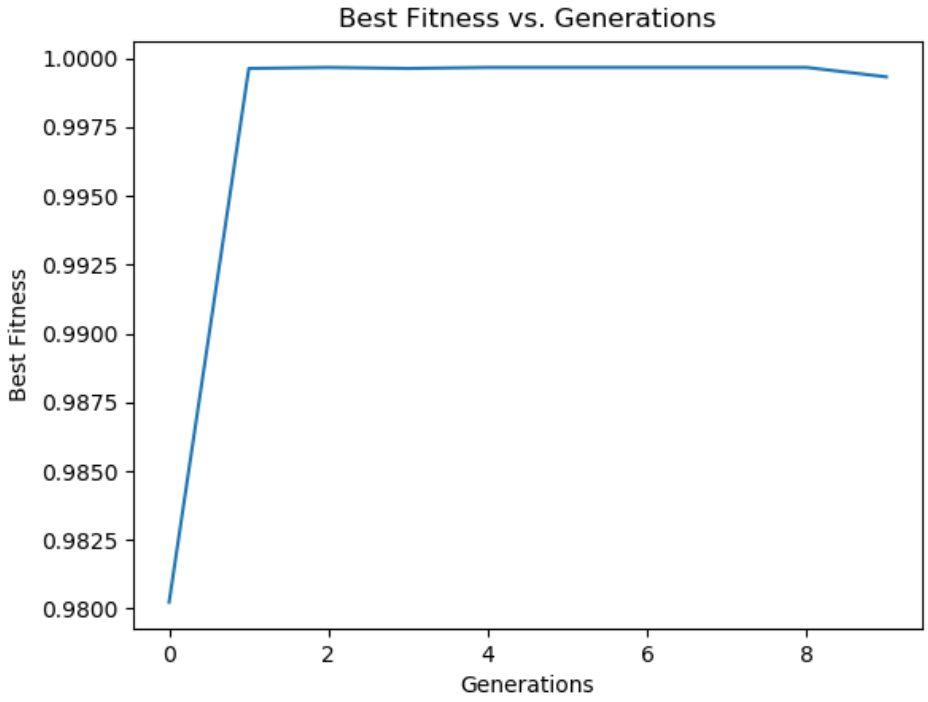

Figure 19: Best Fitness Graph for l=20, N=50, G=10, Pm=0.02, Pc=0.6

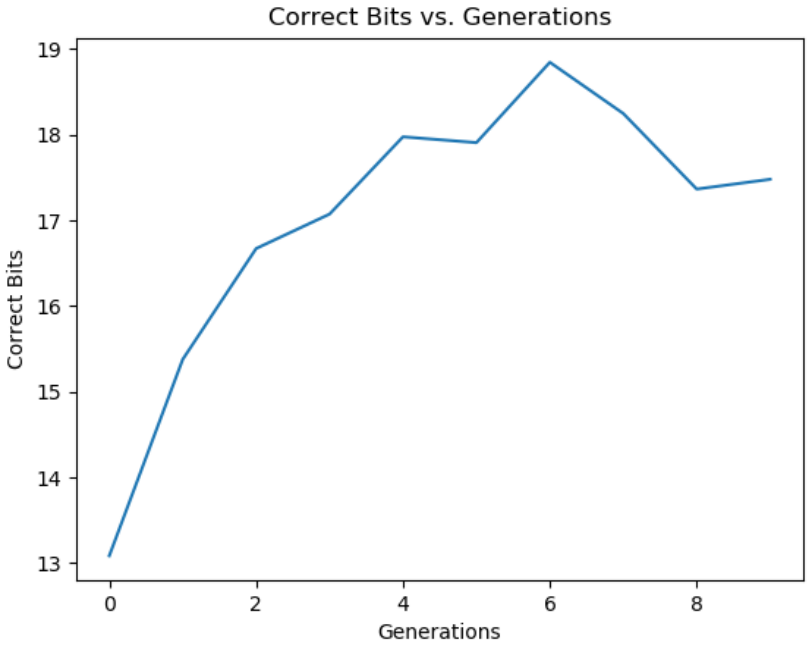

Figure 20: Average Correct Bits for l=20, N=50, G=10, Pm=0.02, Pc=0.6

The final three graphs below in **Figures 21, 22, and 23** show the best attempt to optimize the average fitness, best fitness, and average correct bits of a population after several generations have occurred. The parameters for these graphs were the string length set to 20, the population size set to 50, the number of generations set to 30, the probability of mutation set to 0.02, and the probability of crossover set to 0.6.

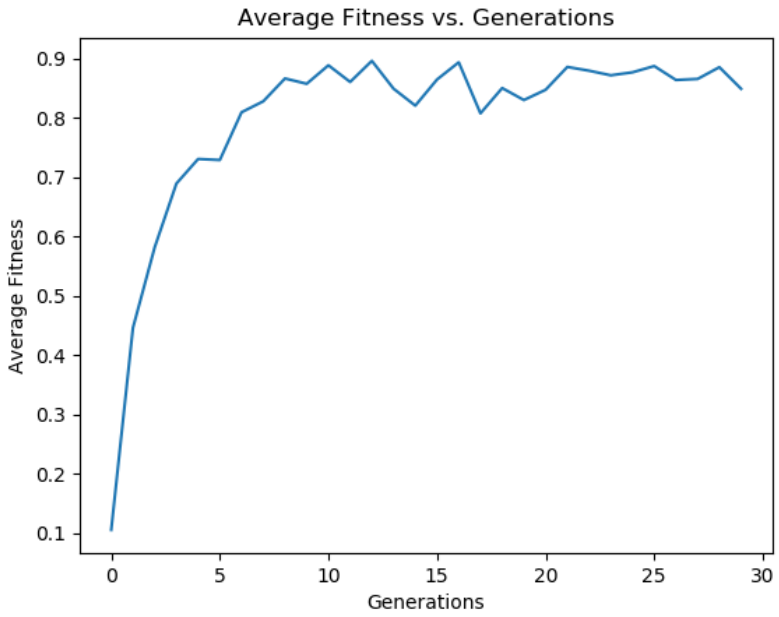

Figure 21: Average Fitness Graph for l=20, N=50, G=30, Pm=0.02, Pc=0.6

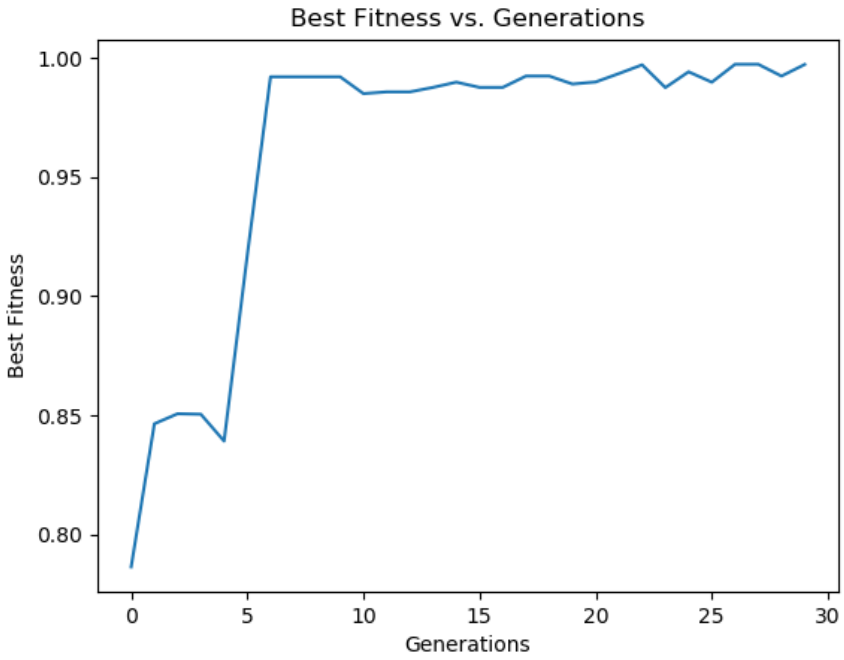

Figure 22: Best Fitness Graph for l=20, N=50, G=30, Pm=0.02, Pc=0.6

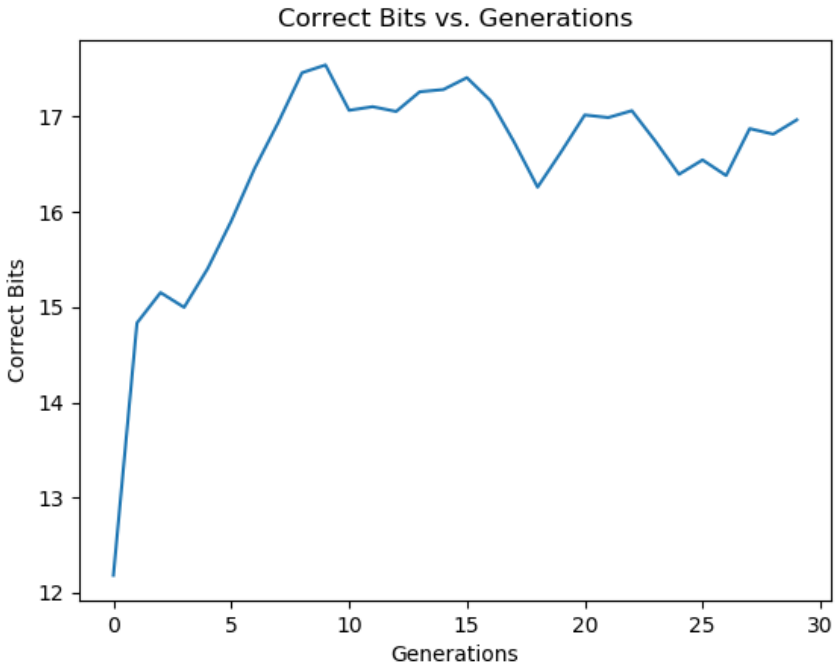

Figure 23: Average Correct Bits for l=20, N=50, G=30, Pm=0.02, Pc=0.6

The next section will provide discussion on the impact of these graphs and the effect the parameters have on this Genetic Algorithm.

Discussion:

What have the graphs above demonstrated about Genetic Algorithms? How do they help answer the questions asked in the Theory section above? Genetic Algorithms model evolution using natural selection. They can also solve some complex problems. The graphs above show relationships between the parameter inputs **(l, N, G, Pm, Pc)** and the optimum solution. They show how this solution evolves with each passing generation and give an idea of what the best parameter values might be to achieve this solution. So, how do the parameters affect the outcome?

The bit string length does seem to play a role in how well the average fitness turns out. The average fitness in **Figure 6** is higher than the average fitness in **Figure 9** and the string lengths are 0.8 and 0.6 respectively. In almost every Best Fitness graph, the results climb to near 1 quickly and mostly hang out there. So, string length does not seem to affect the best fitness. However, the average correct bits do seem to be affected. In Figure 8, the average correct bits are around 13 even though it has a smaller string length than Figure 11 which has around 9 average correct bits. A possible reason for a lower string length performing better is that larger strings might be more difficult to evolve as they have more incorrect bits to work with.

The size of the population affects the behavior of this Genetic Algorithm. Larger populations perform much better than smaller populations. In **Figures 18, 19, and 20,** the population size is 50, the average fitness reaches 0.9, and the average correct bits is around 18. In **Figures 15, 16, and 17**, the population size is 10, the average fitness is 0.6, and the average correct bits is around 3. That is a significant difference. One possible reason for this is that larger populations will have more individuals with better fitness scores to choose from. A higher class of parents selected will produce better offspring. Looking at **Figure 1,** it is very probable that the parents have correct bits in different locations, so single point crossover allows for the chance to inherit more correct bits.

More generations that occur also improve the average fitness score for the population. The higher number of generations spanned allows for selection and mating of higher quality parents. In **Figures 12, 13, and 14,** the number of generations spanned is 70, the average fitness is around 0.85, and the average correct bits is around 11. In **Figures 3, 4, and 5**, the number of generations spanned is 10, the average fitness is around 0.75, and the average correct bits is around 10. The performance improvement doesn't seem significant, but there is improvement. Mutation could play a role in the higher generations not significantly outperforming less generations as this would make it possible to still wind up with lesser fitness scores.

Now that the role each parameter plays has been determined, it is time to look at the graphs and answer the questions in the Theory section above.

What will happen to the average fitness score of a population with each new generation? As a whole, the graphs show that the average fitness score will climb with each new passing generation. However, the affects of mutation and the randomness of selection do allow for slightly lesser average fitness scores to be produced occasionally.

What fitness score will the best individual have? The graphs show that the best individual fitness will quickly climb towards the optimum result which is one. This happens because it is very likely that with each passing generation, at least one individual will have a fitness score near one.

What affect will there be on the number of correct bits in a population when the best individuals are selected more often to produce offspring? The number of average correct bits seems to have a direct relationship with the average fitness according to the graphs above. However, it is possible to have more correct bits and still have a lower fitness score. There is a small optimum window to attain high fitness values. It takes higher order bits to achieve high fitness values according to **Equation 1** which is the fitness function for this Genetic Algorithm. A single 1 in the highest order will have a higher fitness score than a 0 in the highest order followed by all ones. This is the exception.

Given the knowledge above, the graphs in Figures 21, 22, and 23, were an attempt to optimize the solution. The length was set to 20, the population size was set to 50, and the number of generations was set to 30. These parameters were chosen because of their individual performances. Though, they did not achieve all 1's, the average fitness score was between 0.9 and 1.0, and the average correct number of bits was between 17 and 18. The results are pretty good.

Conclusion:

Genetic Algorithms are a good way to solve problems using natural selection. This experiment used a python program to simulate the Genetic Algorithm. The program calculated and used a fitness function as the basis for selecting parents. The parents produced two offspring, either a copy of each parent, or a crossover of the two. Many generations were spanned to evolve the average fitness score of the population. The optimum result would be 1. Graphs were produced to show the effect that individual length, population size, and generations spanned have on the average fitness, best fitness, and average correct bits of a population. The biggest factor that improved the average fitness score was population size. The higher the population, the better the average fitness. The number of generations spanned also played a role but were less significant. Mutation was included so that a result was not achieved to quickly. This allowed for the algorithm to explore more possible solutions. The fitness function can be used to map real world problems and a Genetic Algorithm can optimize that function to find fast and optimal solutions.