Project 5 Report: Genetic Algorithms

Patrick Elam

CS420

April 22, 2015

Introduction:

Genetic algorithms have been conceptualized since the very beginnings of computers. During the 1960’s, researchers began to investigate the notion of computationally finding solutions to problems via simulating an evolutionary process on potential solutions.

The way genetic algorithms work is as follows. First, one must have / produce some sort of objective formula (known as the fitness formula) by which potential solutions can be judged. With most problems to which computers are applied, this is takes the form of a mathematical formula. Then, a variety of solutions to the formula are randomly generated. The following is repeated until an optimal solution is found: Each member of the population (that is, each potential solution) is evaluated against the fitness formula and weighted depending on how well that solution fit the fitness formula. Then, two “parent” solutions are selected, and their properties duplicated and combined. The weight of each parent affects the likelihood that it is chosen, such that parent solutions that fared better (higher fitness score) are more likely to pass along their traits to successive generations. This process of selecting parents and generating children is repeated until there is the same number of children as there are parents. At this point, the children solutions replace the parents and are themselves evaluated on their fitness. The process then continues.

The advantage to this kind of problem solving is that many types of solutions are considered. This includes solutions that humans might not have before considered. This type of algorithm is best suited to finding solutions for problems that need to be optimized, and not necessarily for problems that have only an exact solution. One disadvantage to this type of solution finding is that the problem given must have some quantifiable way of determining fitness.

Simulator and Calculations:

Alex Chaloux and I wrote a program in C++ that would find the solution to a problem using a genetic algorithm. The problem we were instructed to solve was a simple function:

$$F(s) = (x/2^{\ell})^{10},$$

The function took one input: x as an integer. To be able to modify and come up with different iterations of solutions, we examined potential solutions as bit strings. Each bit string was evaluated as an integer and plugged into the function above to determine fitness. The astute will notice that the optimal solution is obvious, that all bits be set to 1, and thus give the highest output from the function; however, this solution is not as immediately obvious to a computer, and serves as a good example problem.

In determining run conditions for our genetic algorithm on this function, we altered five separate variables to determine their effect on the algorithm. The variables were: 1) The number of genes in the genetic string, or the number of bits in the integer 2) The population size, or the number of possible solutions 3) The mutation probability. This was the probability that a bit would be mutated from what its parents' bit was. 4) The crossover probability. This was the probability that some of the bits would be swapped in the string. And 5) The number of generations over which the problem was run.

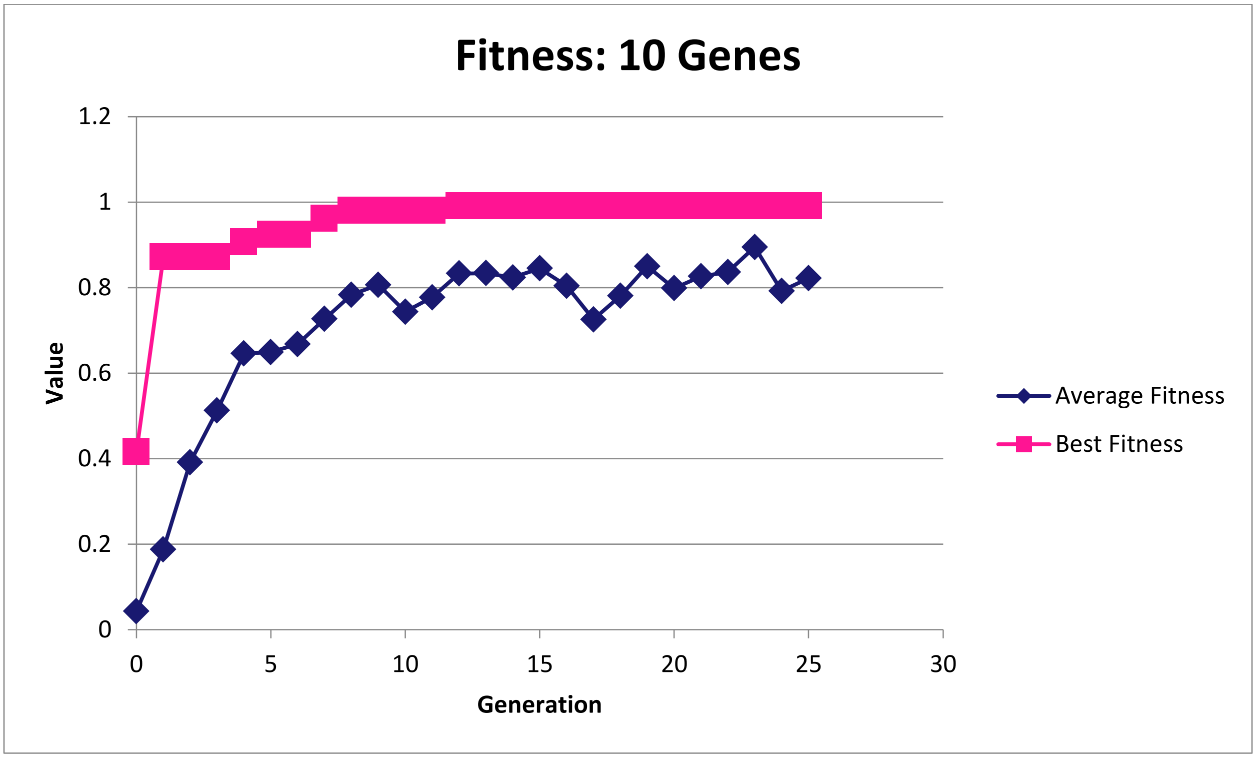
In running the genetic algorithm simulator, we took the 5 variables, as well as a number to seed the random generator. If the program was passed a -1, the number generator was seeded on time (more random). We wrote a configuration generator in C++ to write configuration files to automate simulation runs. The configuration generator generated 25 different variations of the 5 variables. For each variable, we had a configuration with 5 variations that that variable, while holding the other 4 variables constant. The static values were constant throughout all tests, and the values were as follows: 1) Number of genes: 20 2) Population size: 30 3) Mutation probability: 0.03 4) Crossover probability: 0.06 and 5) Number of generations: 25. For all configurations, we seeded the random number generator with -1.

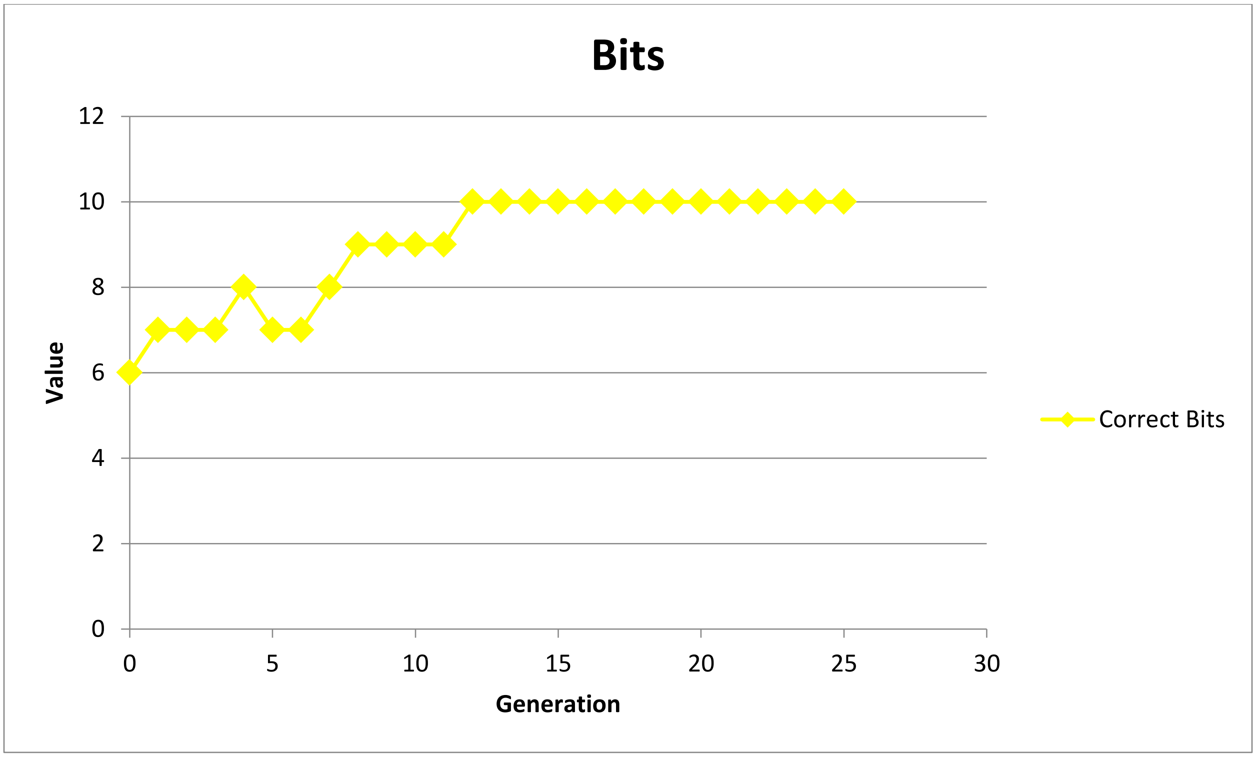
After running the 25 configurations, we graphed 3 points of data from each simulation. We graphed the both the average fitness of the population and the fitness of the best individual against time (generation number). We also plotted the number of correct bits in the most fit individual against time (generation number).

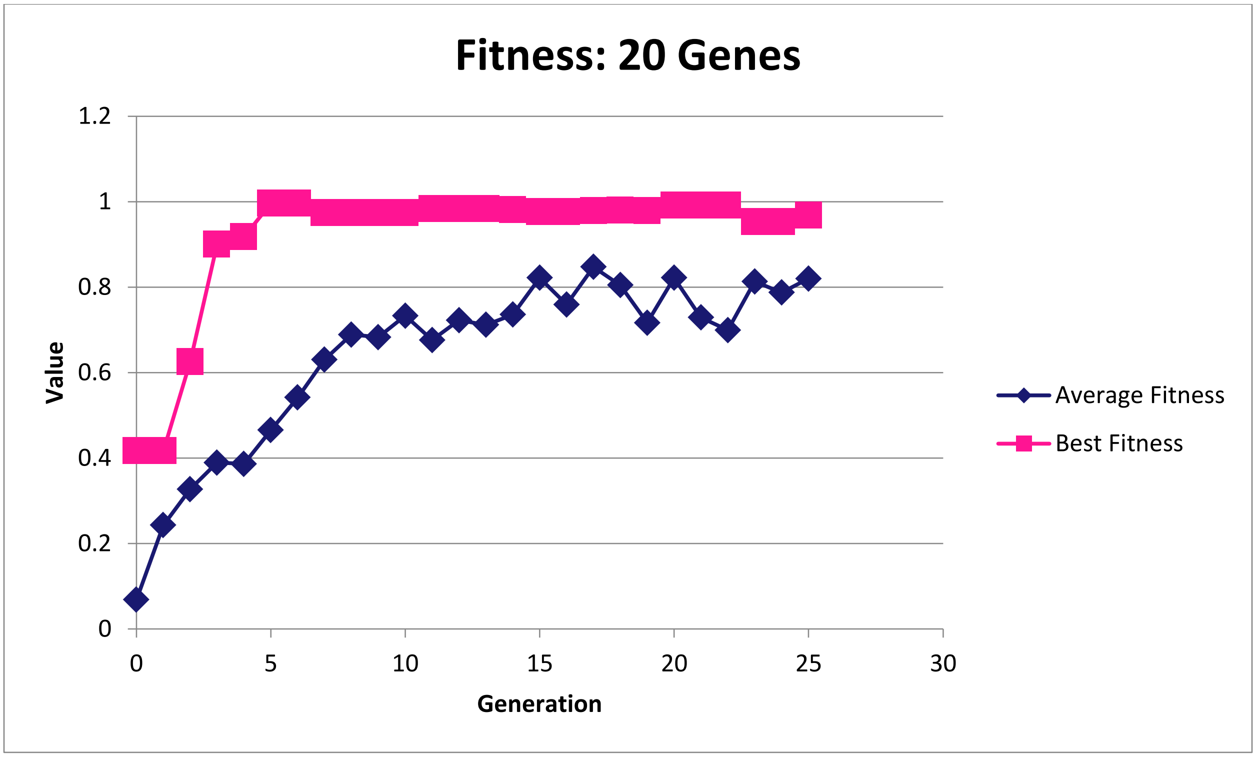
Results:

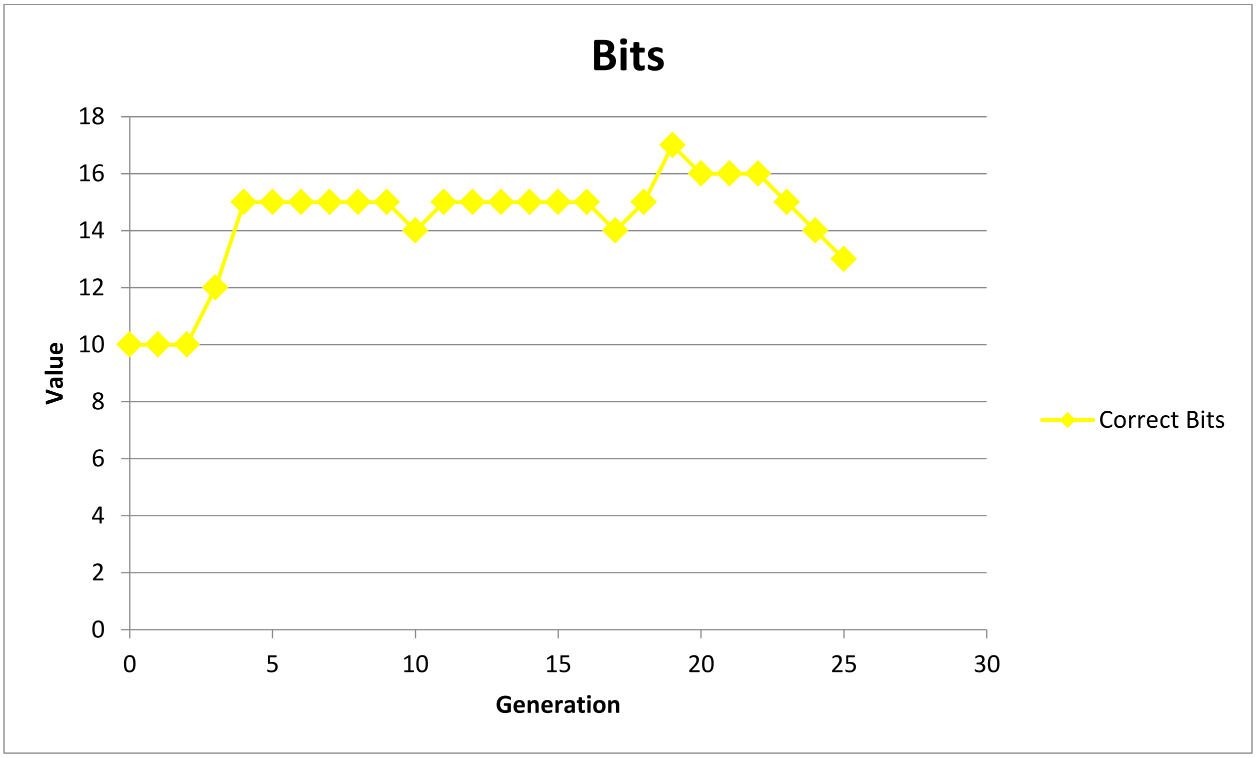
In the interest of being concise, I will only put the first, third and fifth iterations of each variable. I will show the Average Fitness and Best fitness graphs for these variations. At the end, I will include several other iterations of the Number of Correct bits vs generation graphs. This will be sufficient to show the patterns, but will limit the number of graphs.

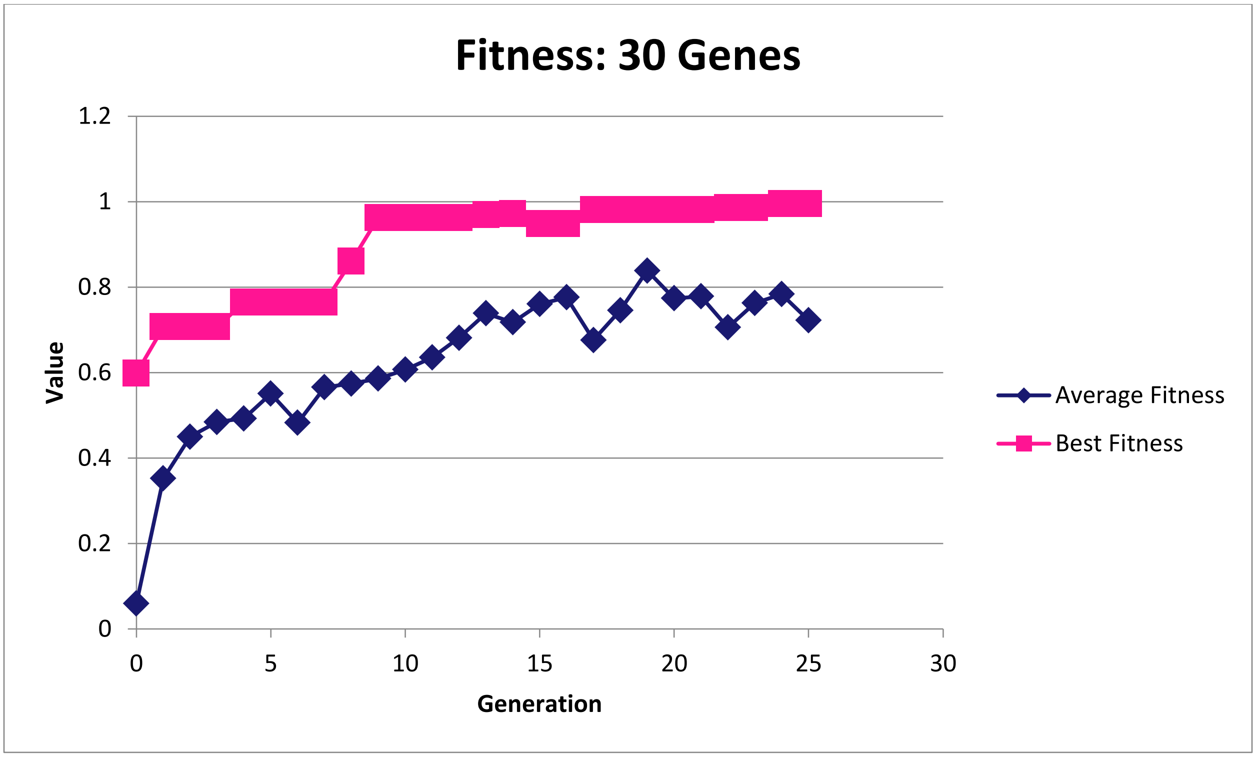
The following are the results when graphing the first, third and fifth variation of the number genes: 10, 20 and 30, respectively.

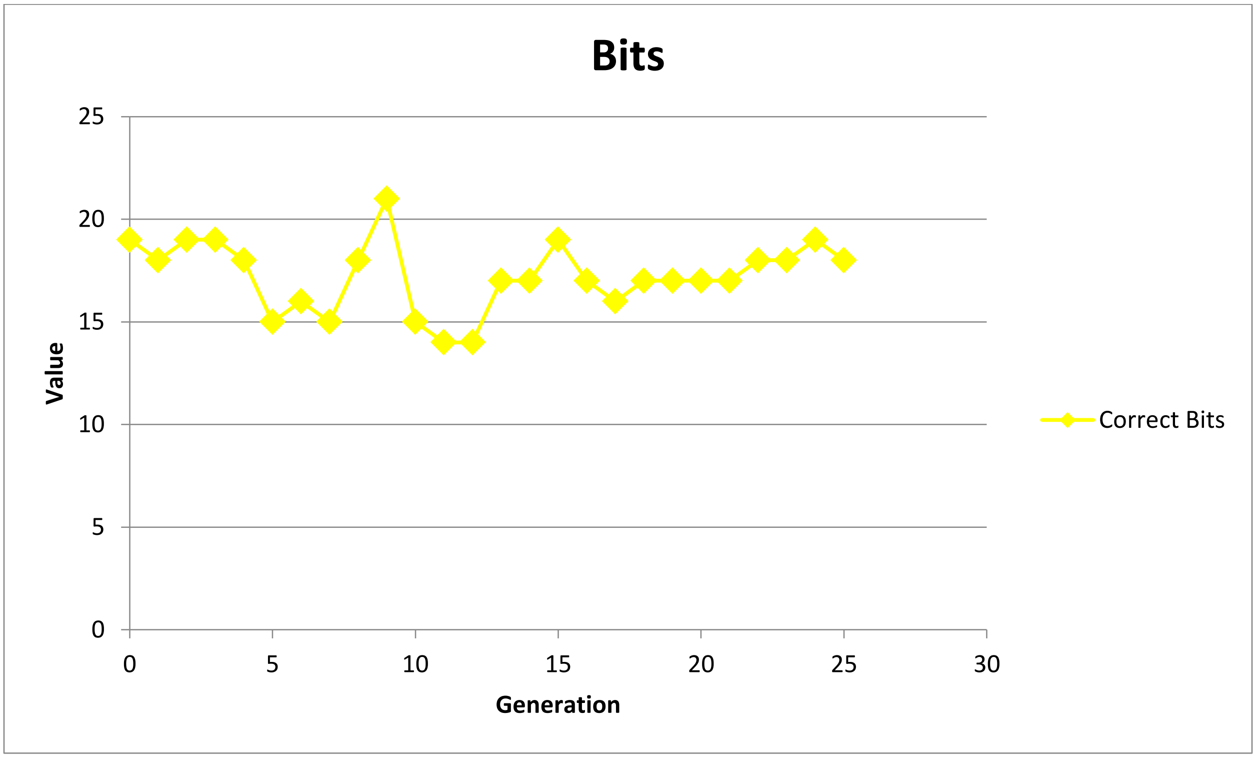




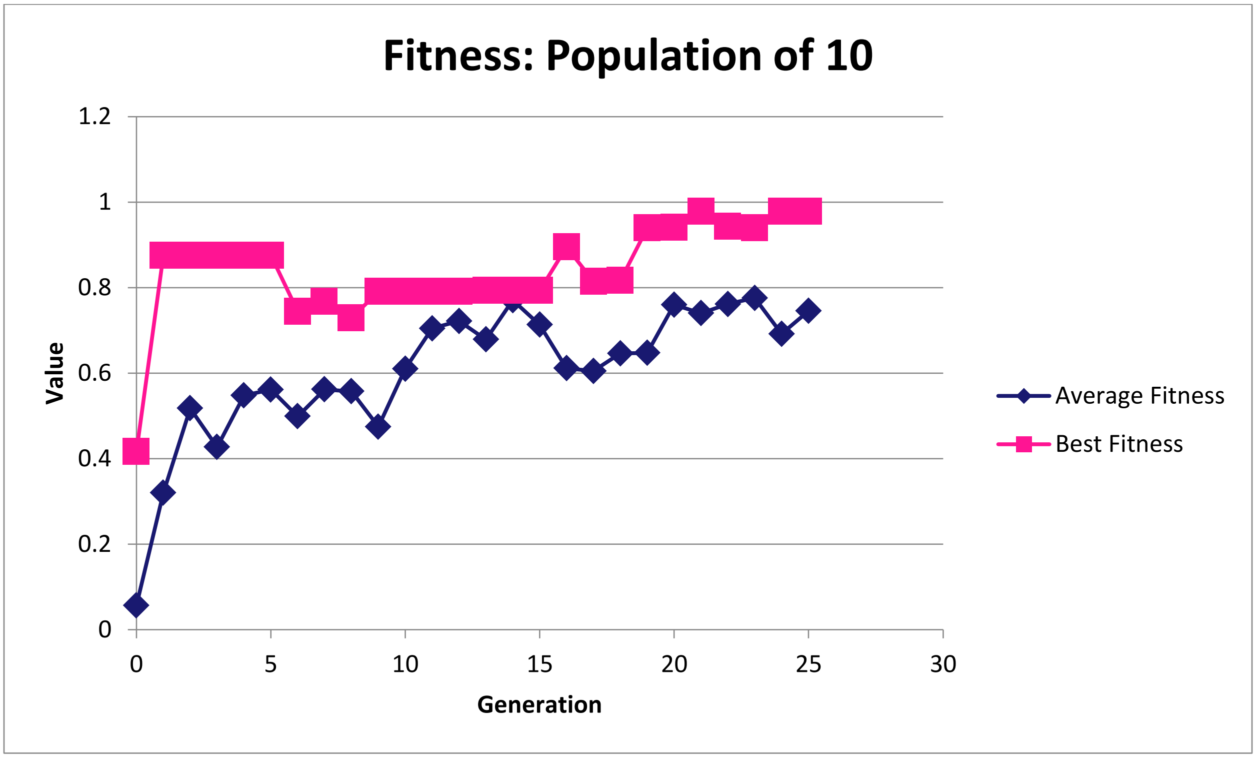

Fitness: 20 Genes
1.2
1
0.8
0.6
0.4
0.2
0
Value
0
5
10
15
20
25
30
Generation
Average Fitness
Best Fitness

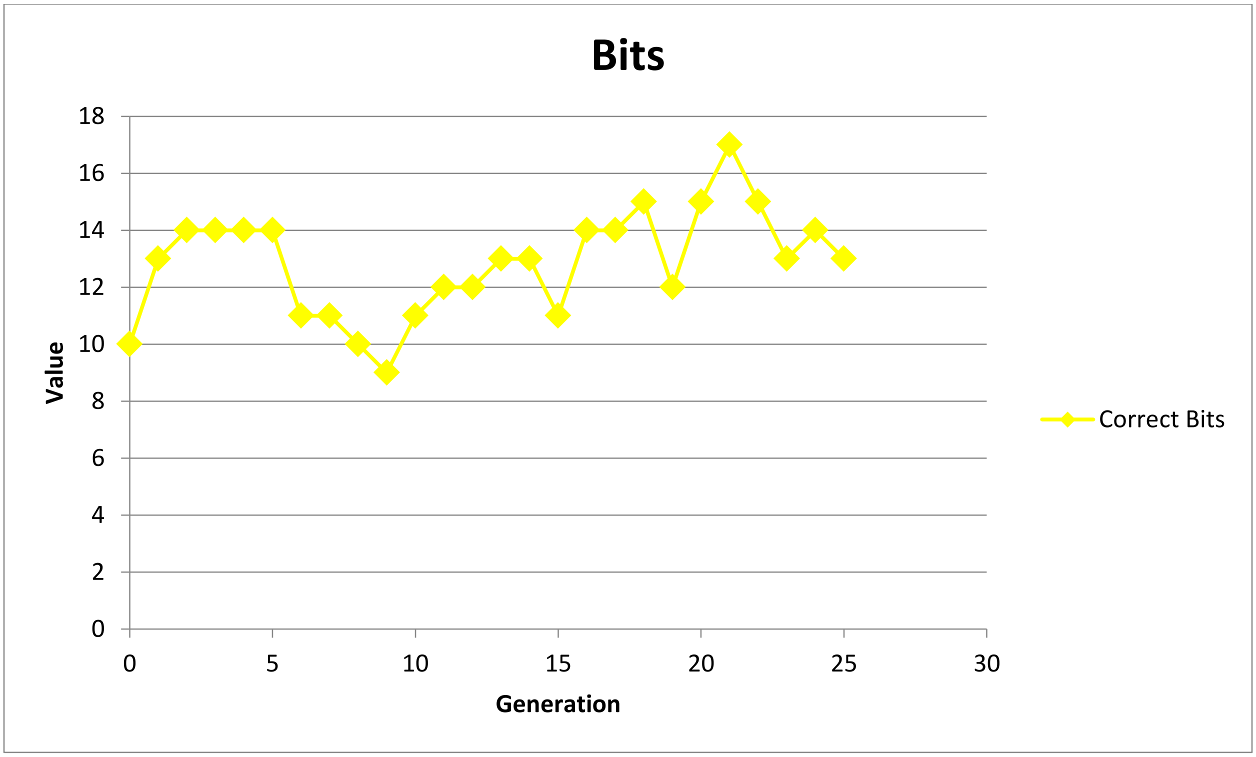

Bits
18
16
14
12
10
8
6
4
2
0
Value
0
5
10
15
20
25
30
Generation
Correct Bits

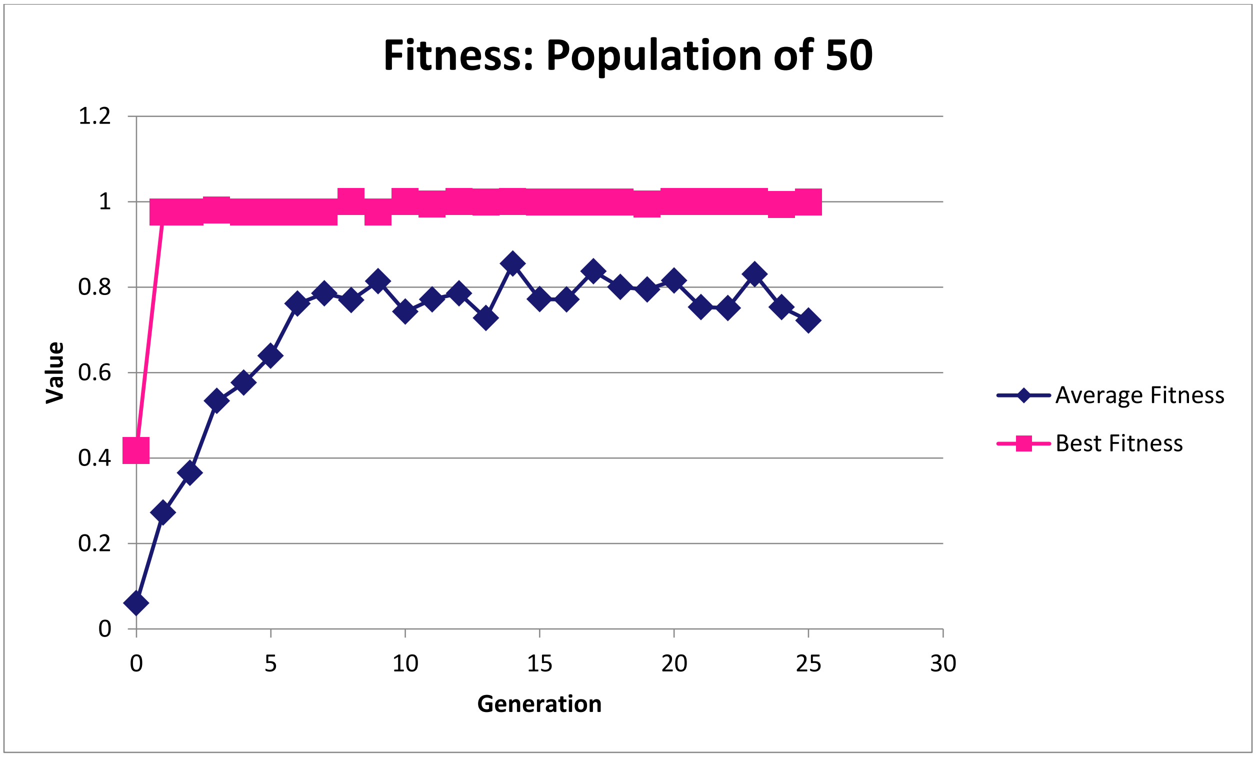

Fitness: 30 Genes
Value
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
30
Generation
Average Fitness
Best Fitness

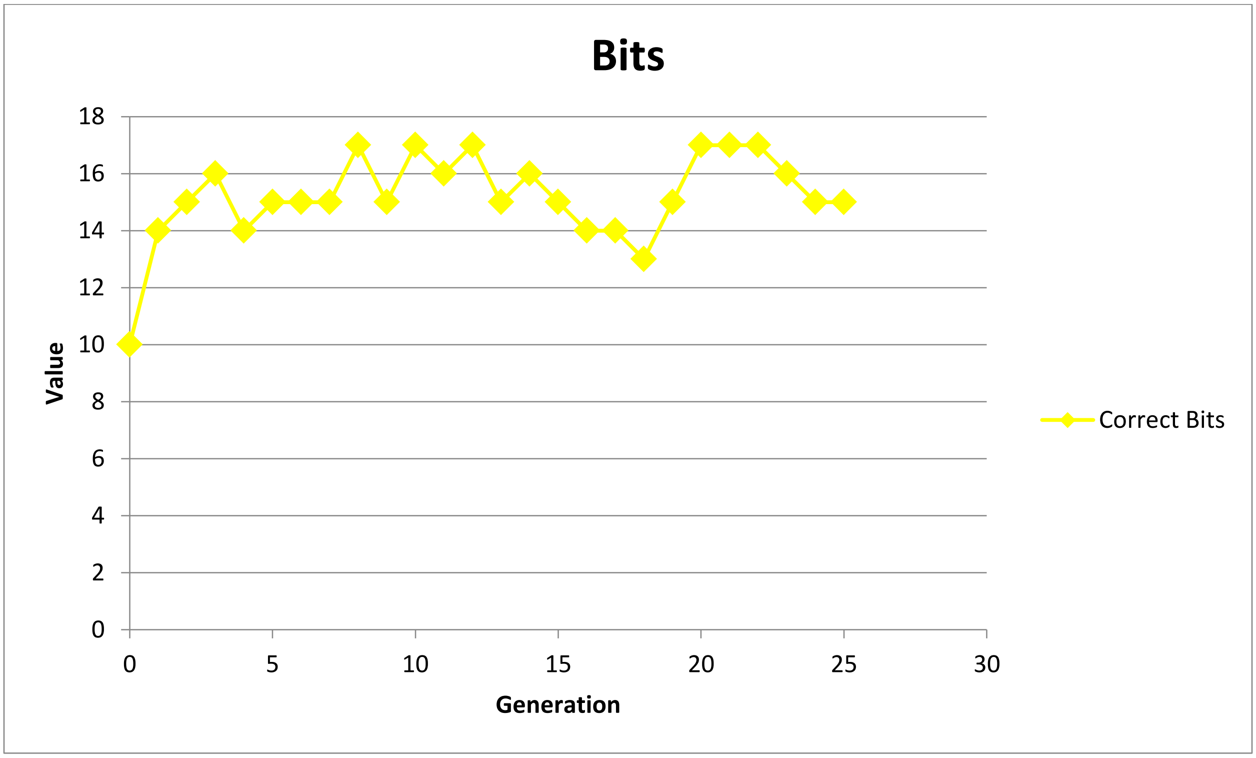

Bits
Value
25
20
15
10
5
0
0
5
10
15
20
25
30
Generation
Correct Bits

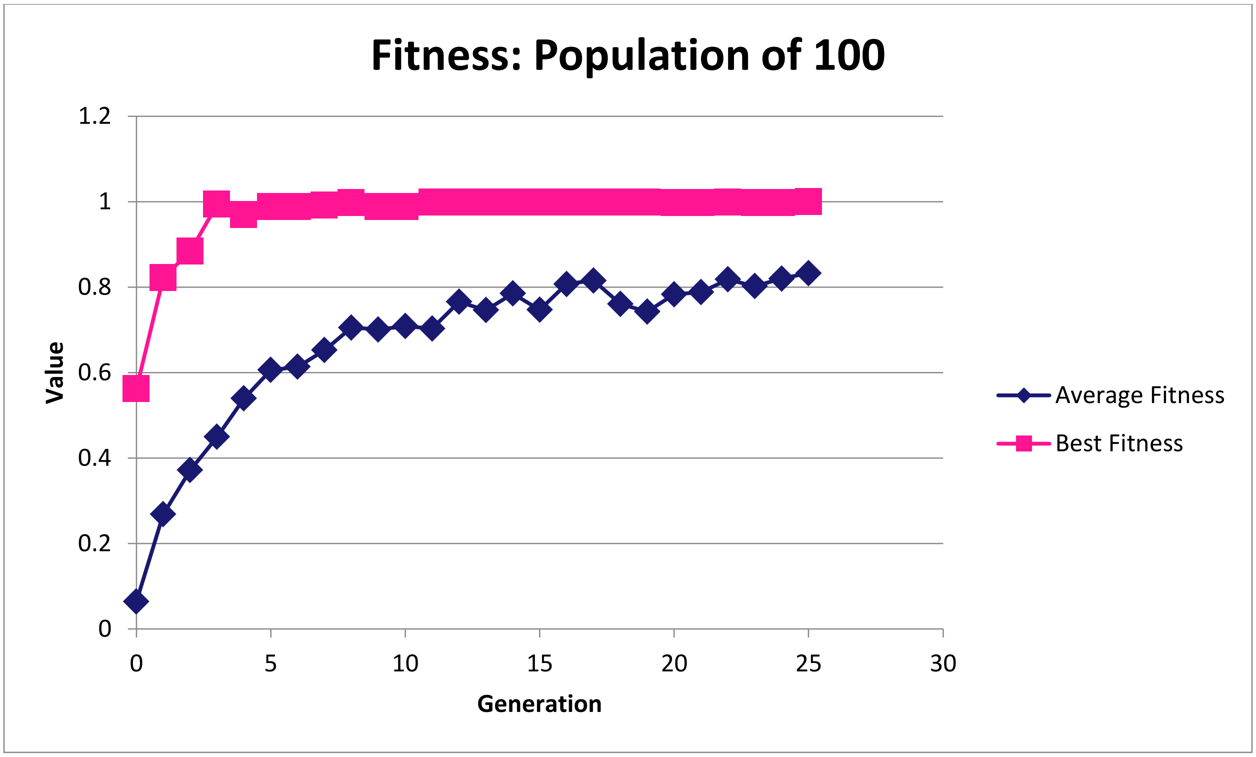
The following are the results when graphing the first, third and fourth variation of the population size: 10, 50 and 100, respectively.

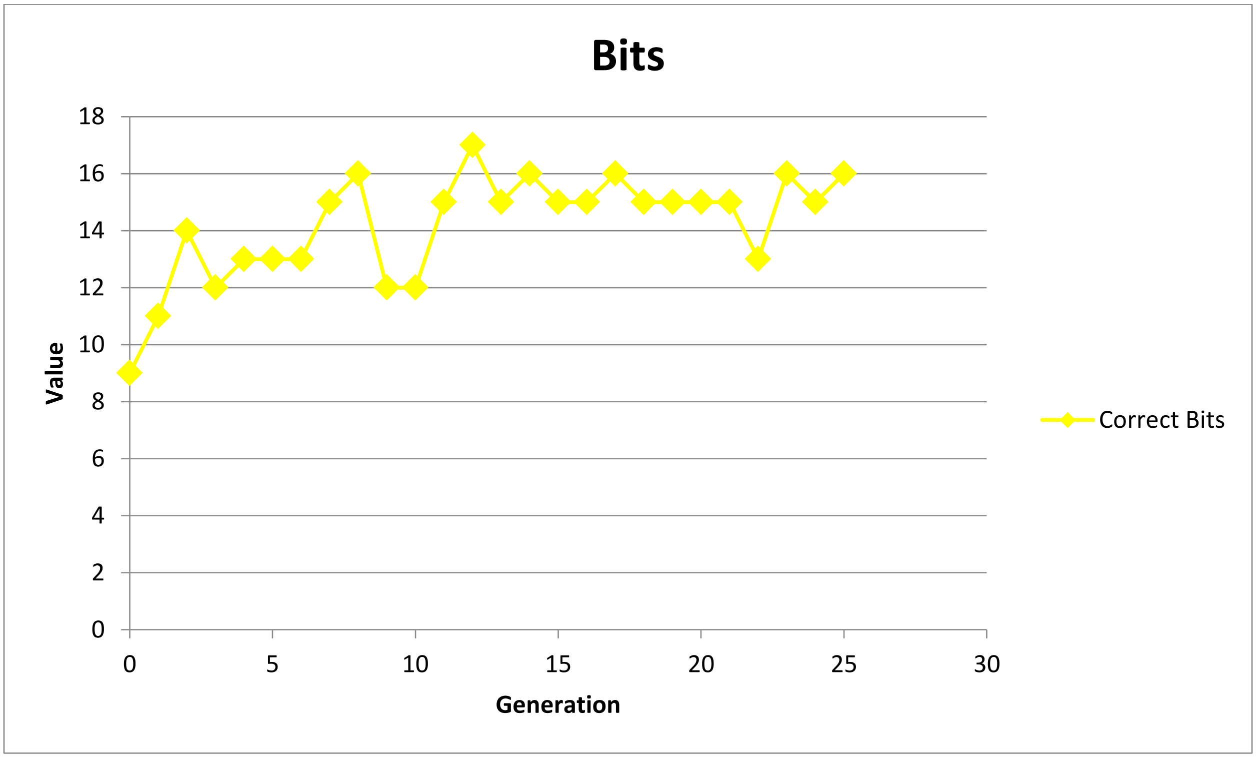




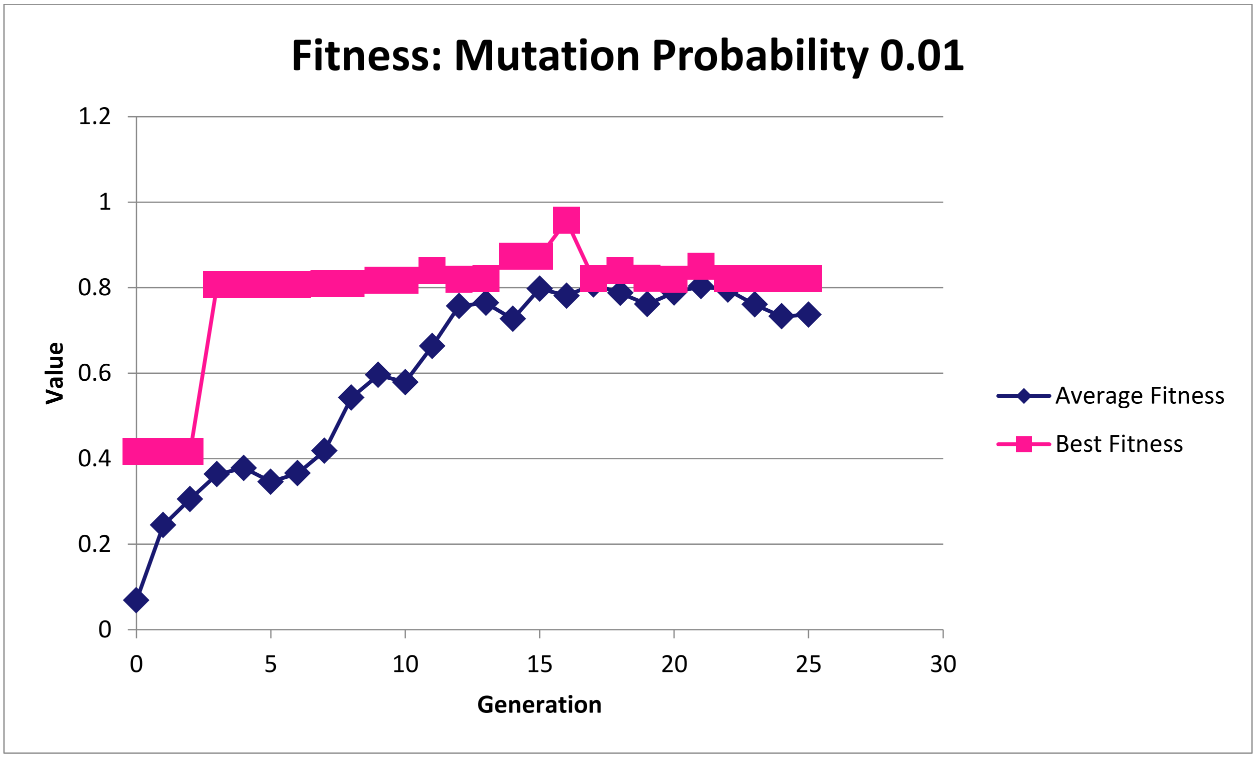

Fitness: Population of 50
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
30
Value
Generation
Average Fitness
Best Fitness

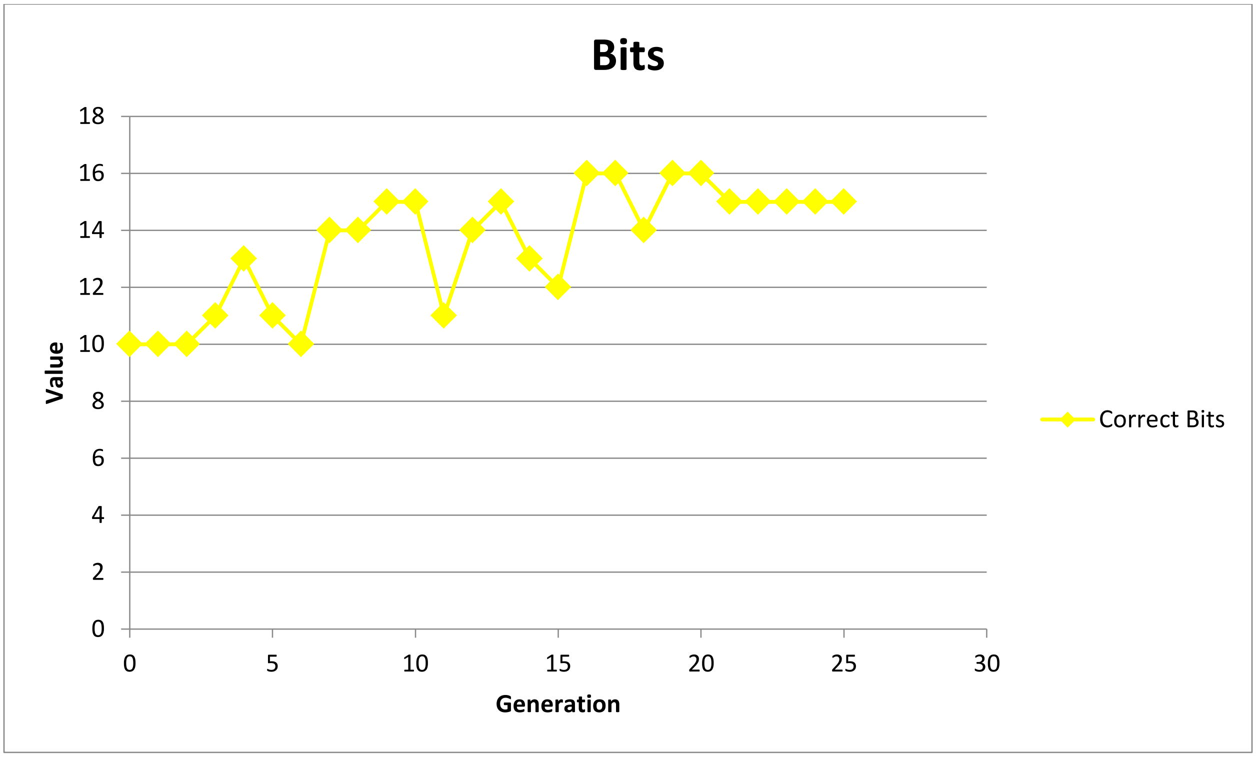

Bits
18
16
14
12
10
8
6
4
2
0
0
5
10
15
20
25
30
Value
Generation
Correct Bits

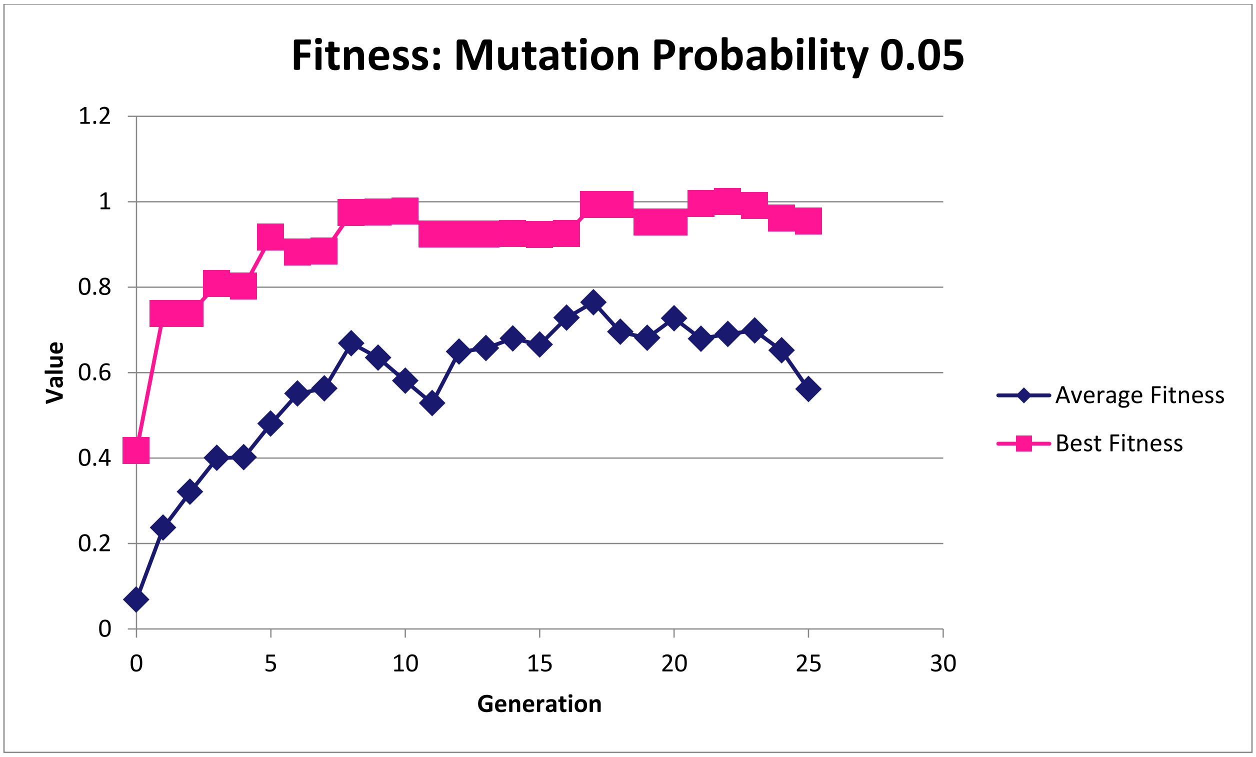

Fitness: Population of 100
1.2
1
0.8
0.6
0.4
0.2
0
Value
0
5
10
15
20
25
30
Generation
Average Fitness
Best Fitness

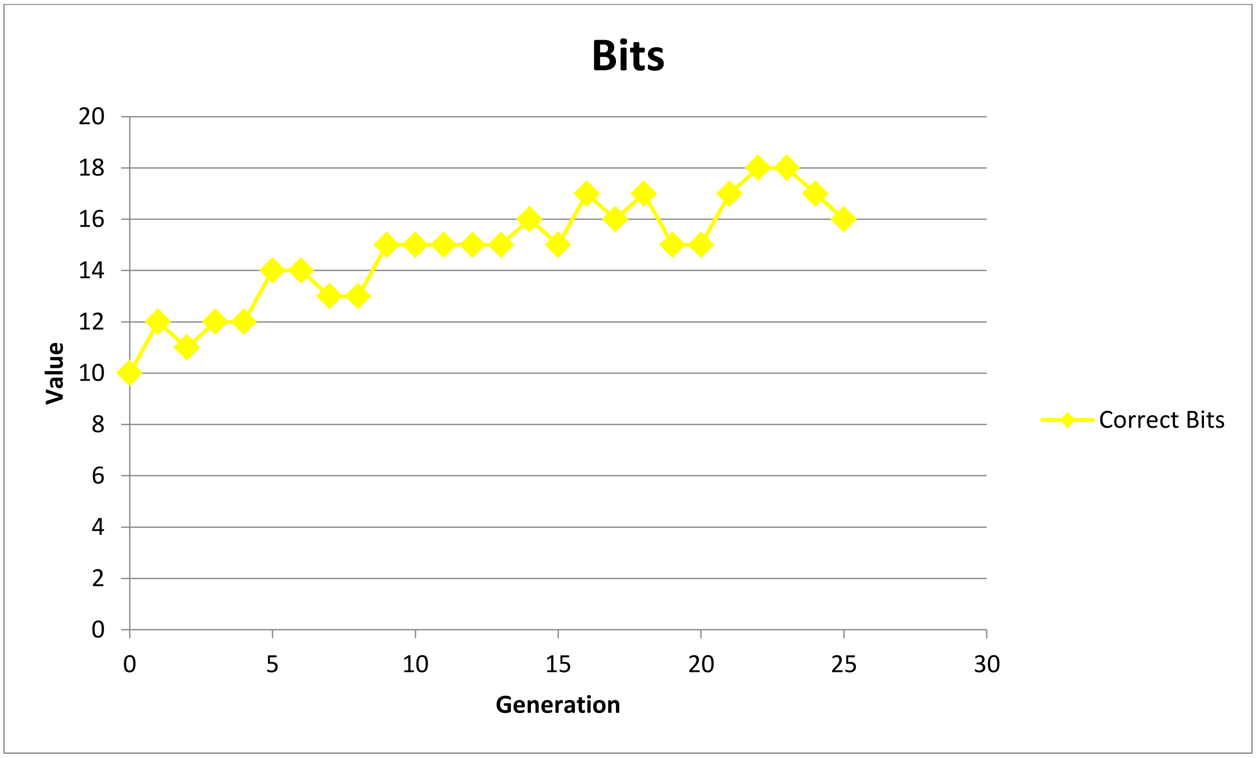

Bits
18
16
14
12
10
8
6
4
2
0
Value
0
5
10
15
20
25
30
Generation
Correct Bits

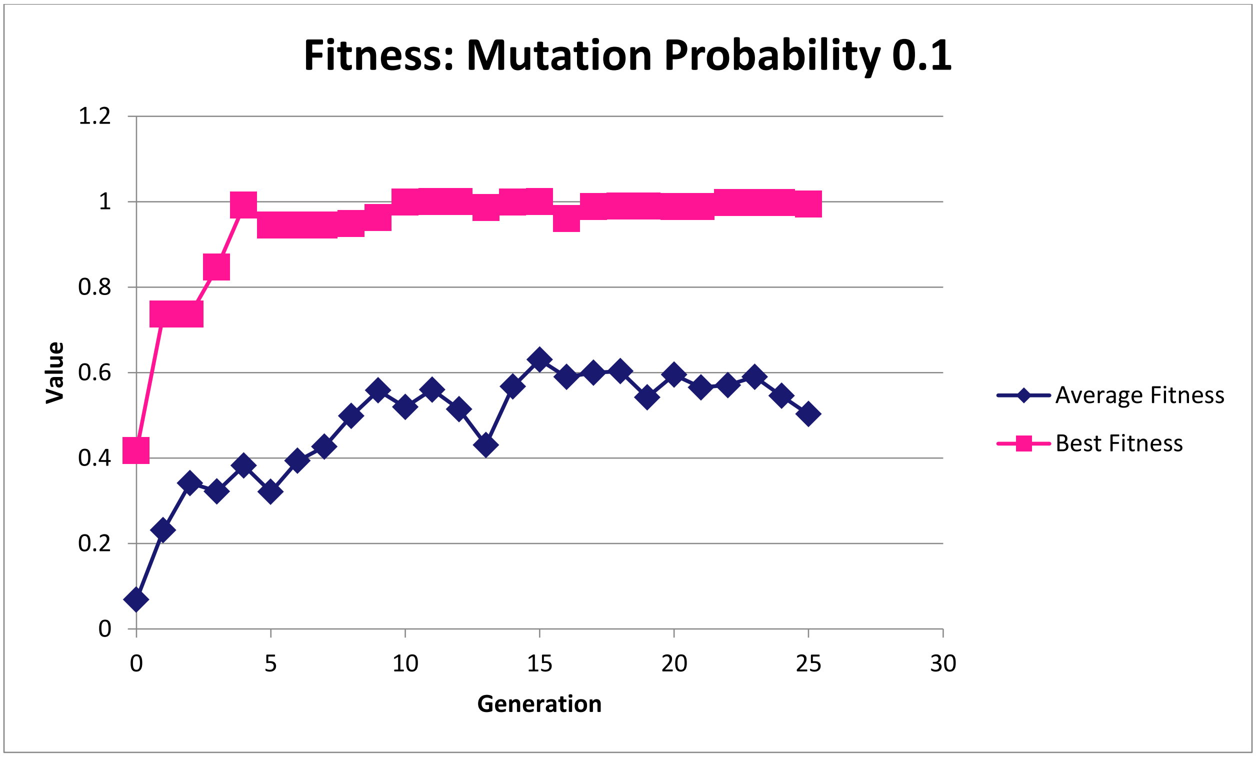
The following are the results when graphing the first, third, fourth and fifth variations of the mutation probability: 0.01, 0.05, 0.1, and 0.2, respectively.

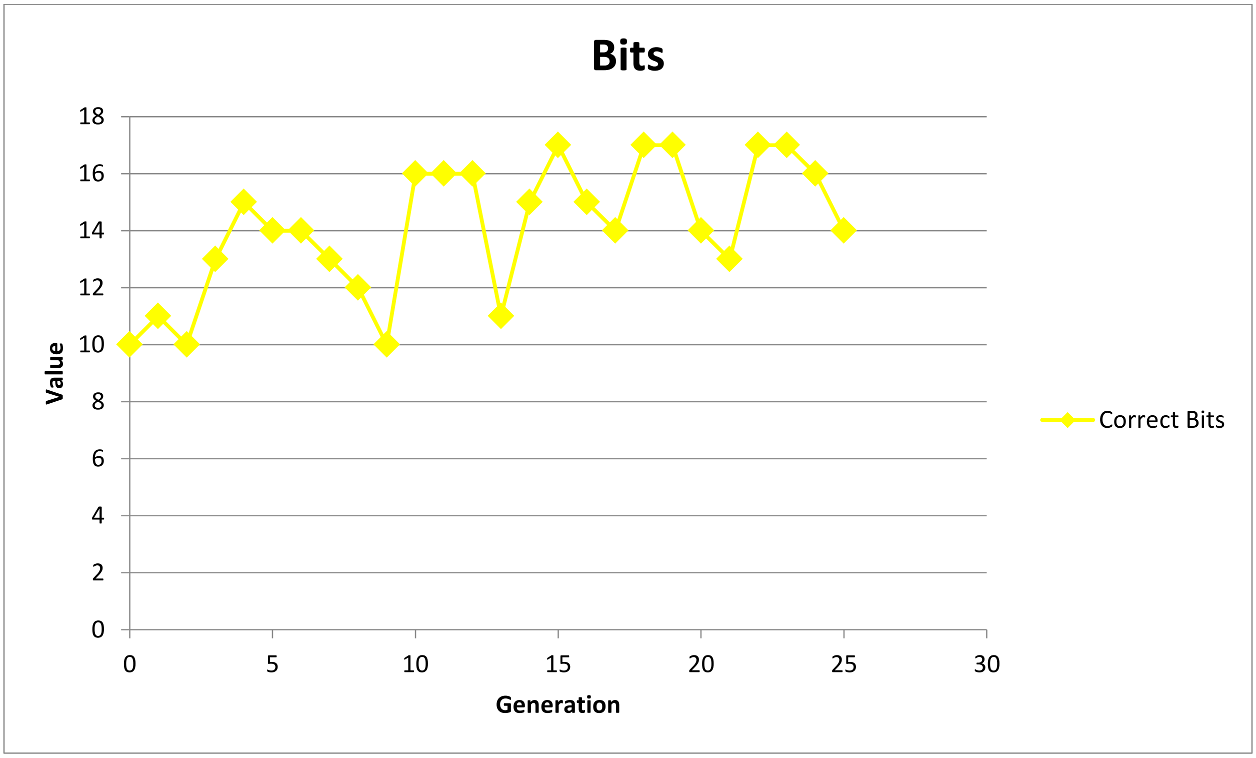


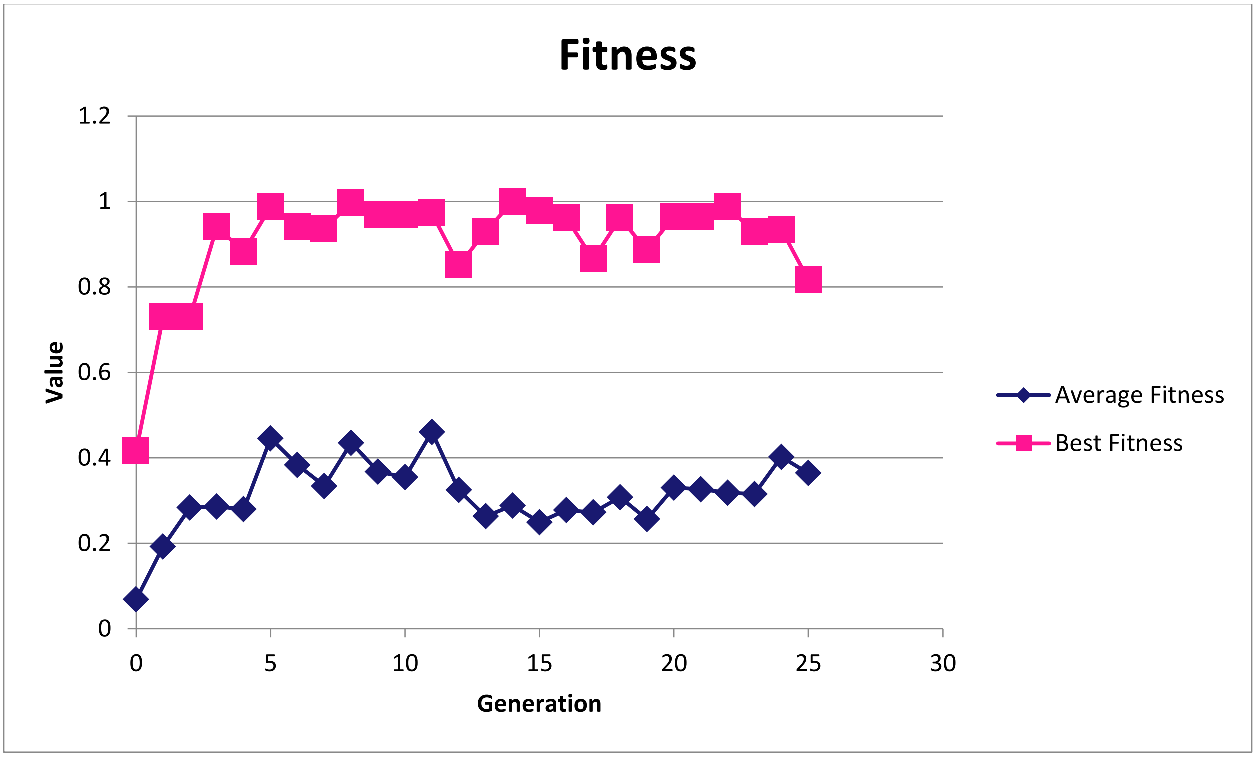


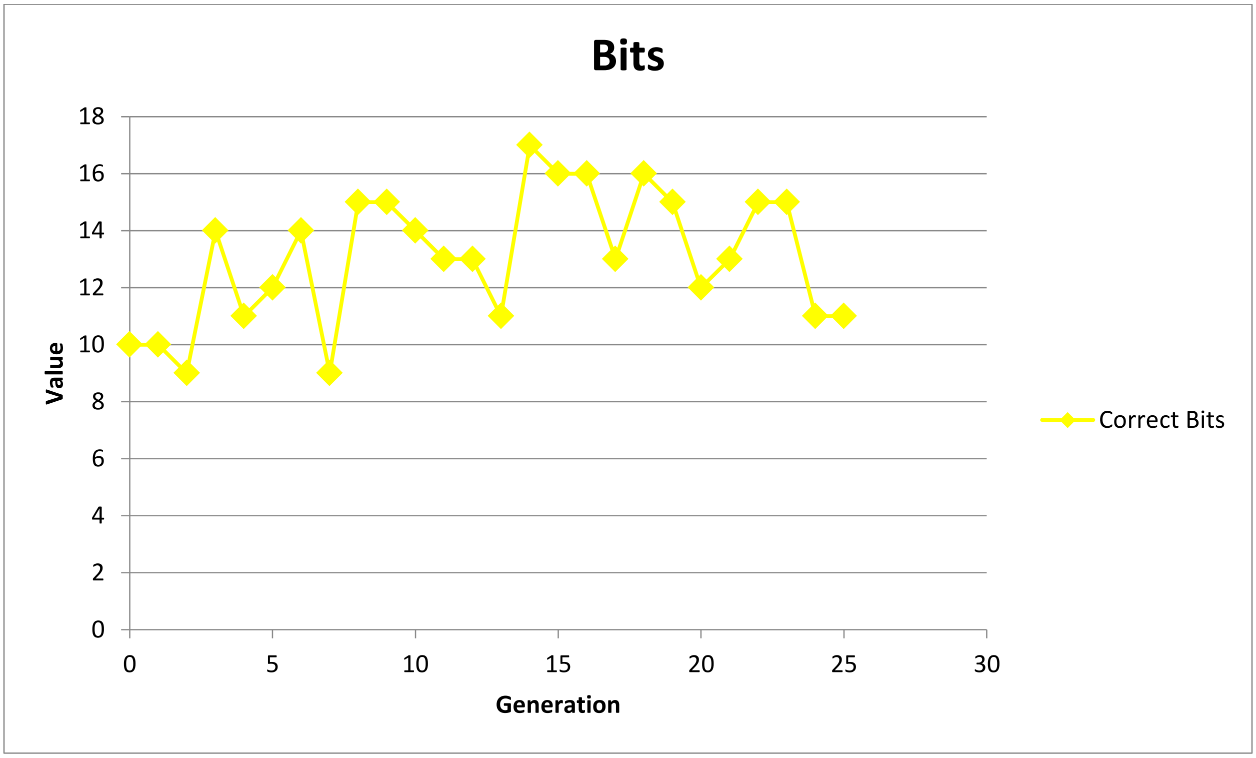

Fitness: Mutation Probability 0.05
Value
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
30
Generation
Average Fitness
Best Fitness


Bits
Value
20
18
16
14
12
10
8
6
4
2
0
0
5
10
15
20
25
30
Generation
Correct Bits

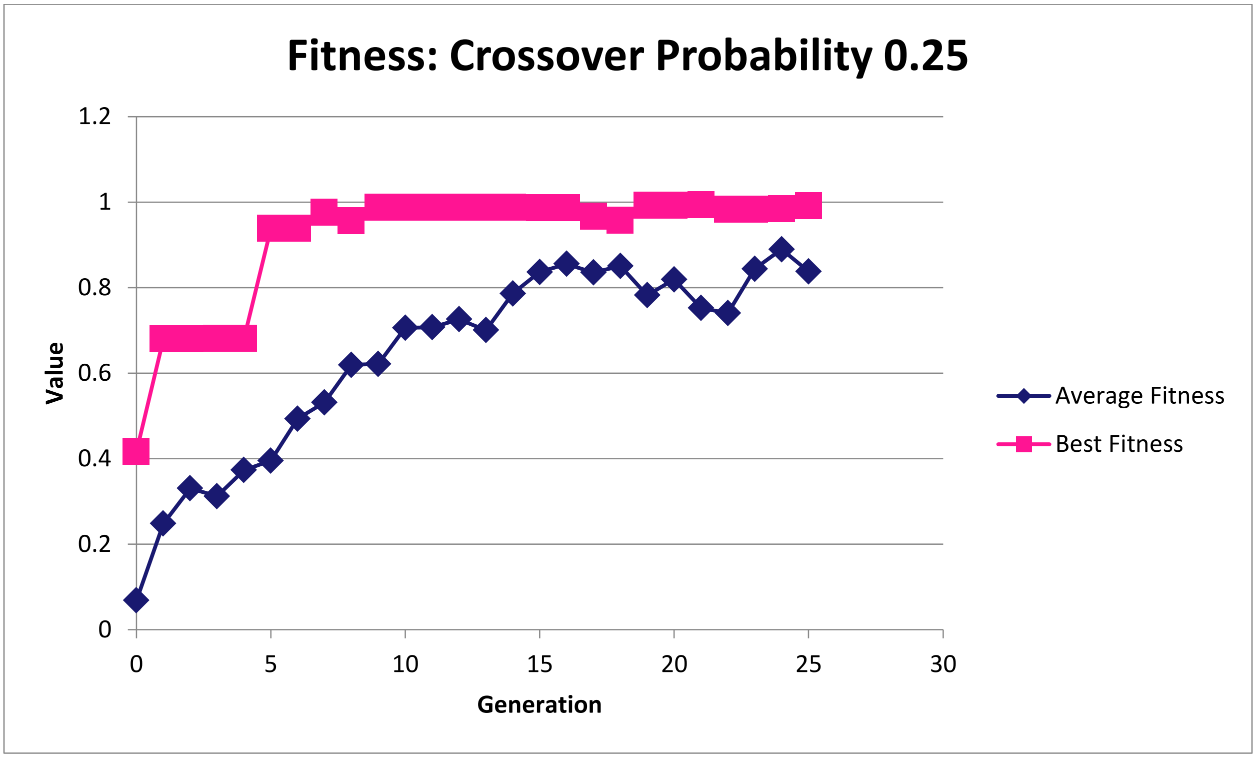

Fitness: Mutation Probability 0.1
1.2
1
0.8
0.6
0.4
0.2
0
Value
0
5
10
15
20
25
30
Generation
Average Fitness
Best Fitness

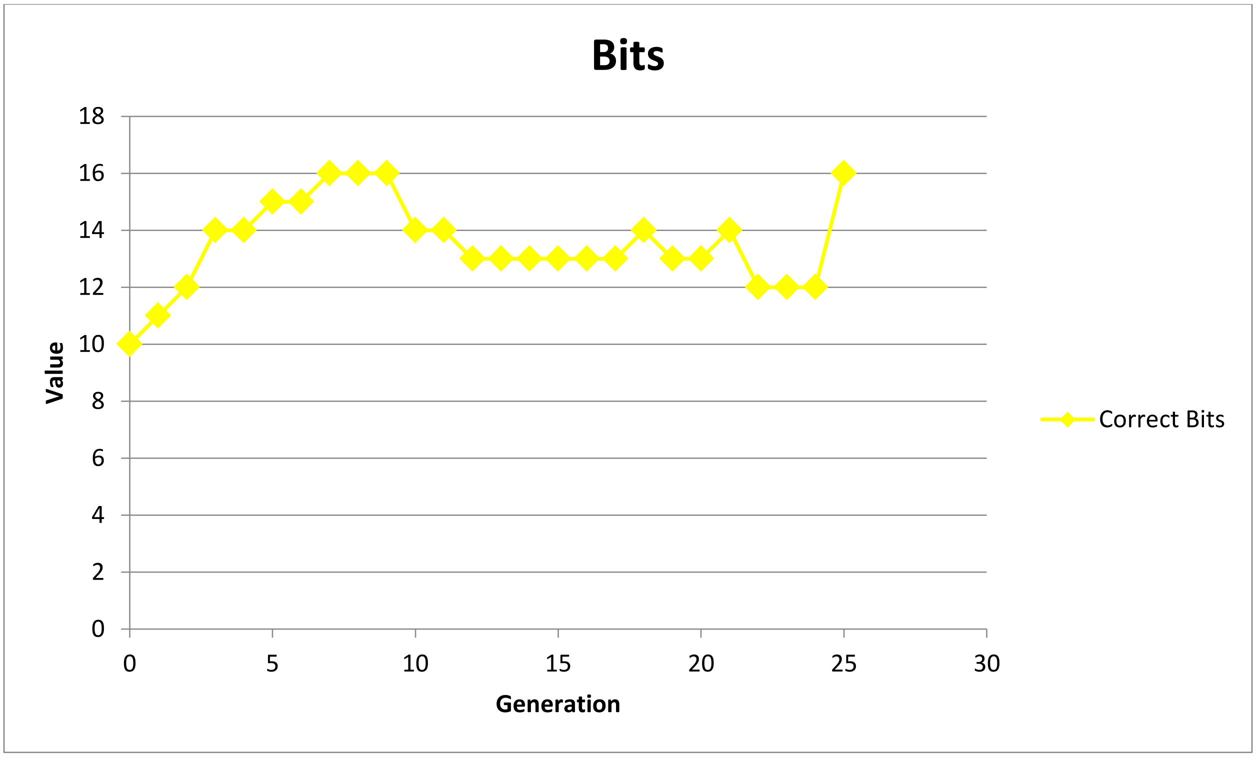

Bits
18
16
14
12
10
8
6
4
2
0
Value
0
5
10
15
20
25
30
Generation
Correct Bits

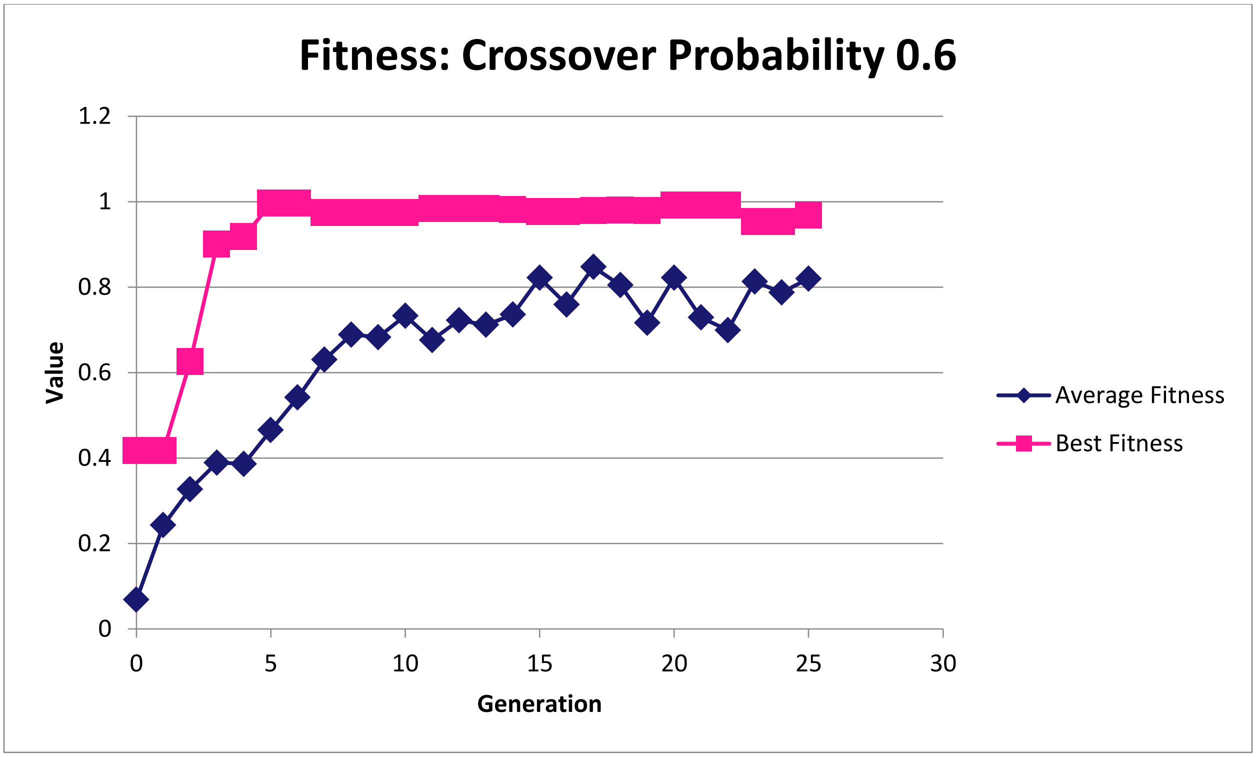

Fitness
1.2
1
0.8
0.6
0.4
0.2
0
Value
0
5
10
15
20
25
30
Generation
Average Fitness
Best Fitness

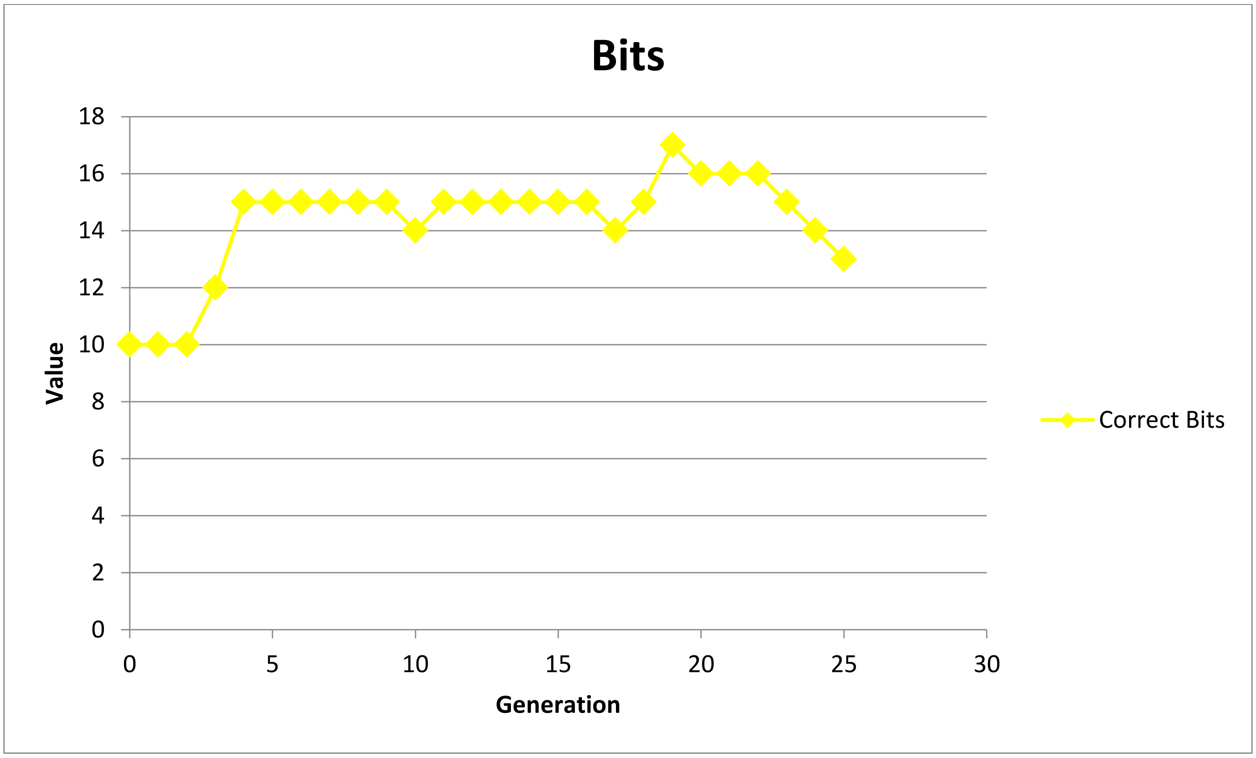

Bits
18
16
14
12
10
8
6
4
2
0
Value
0
5
10
15
20
25
30
Generation
Correct Bits

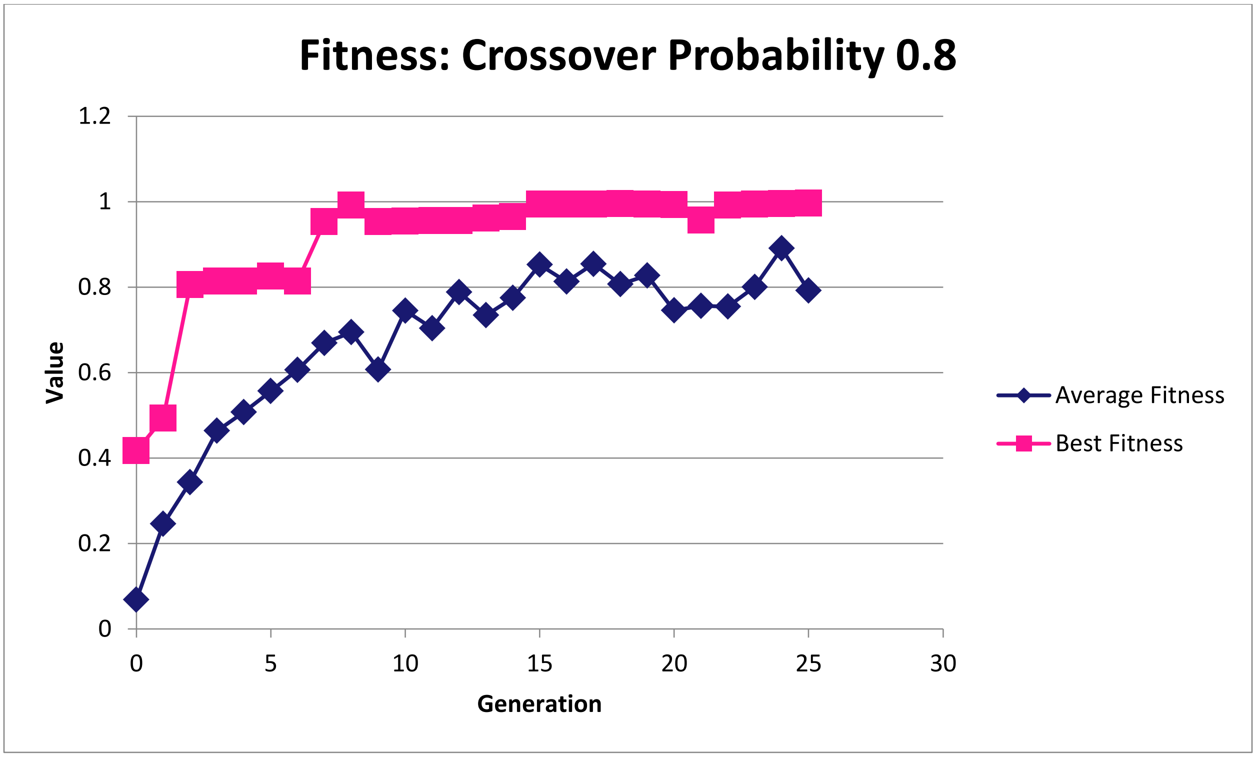
The following are the results when graphing the first, third and fifth variation of the mutation probability: 0.25, 0.6 and 0.8, respectively.

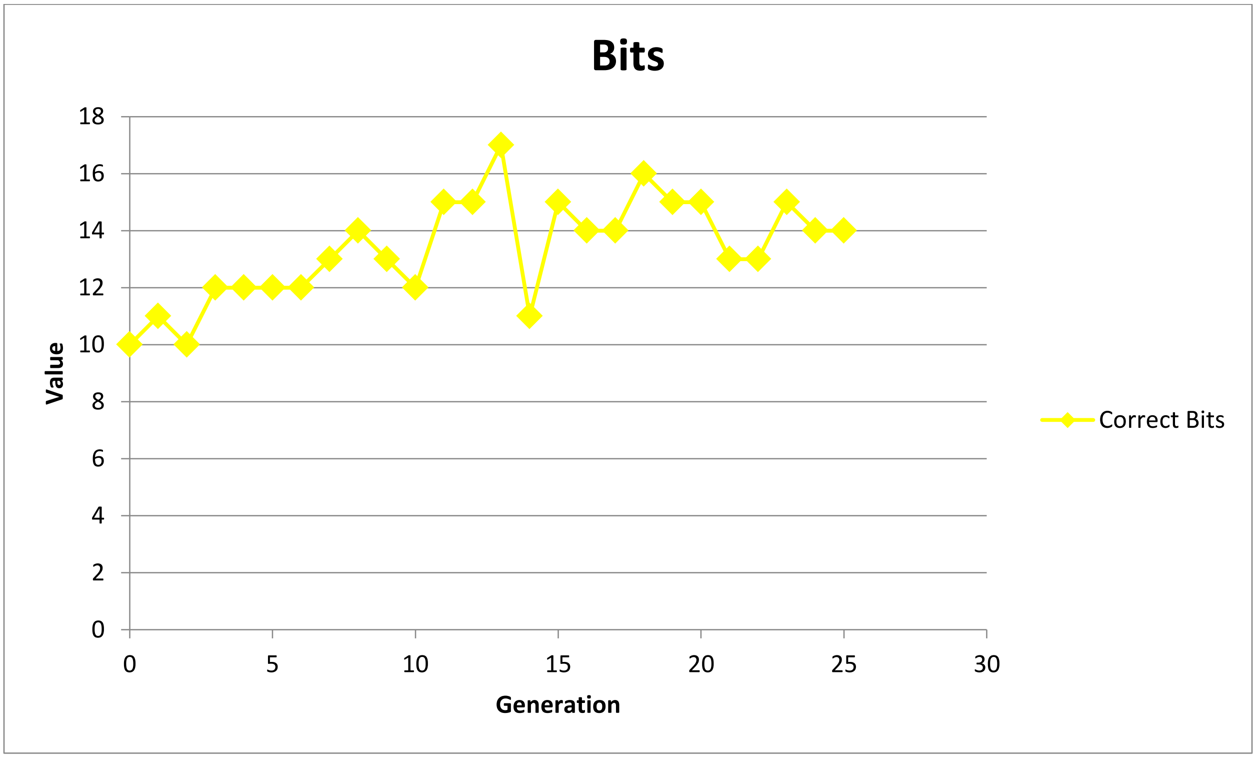




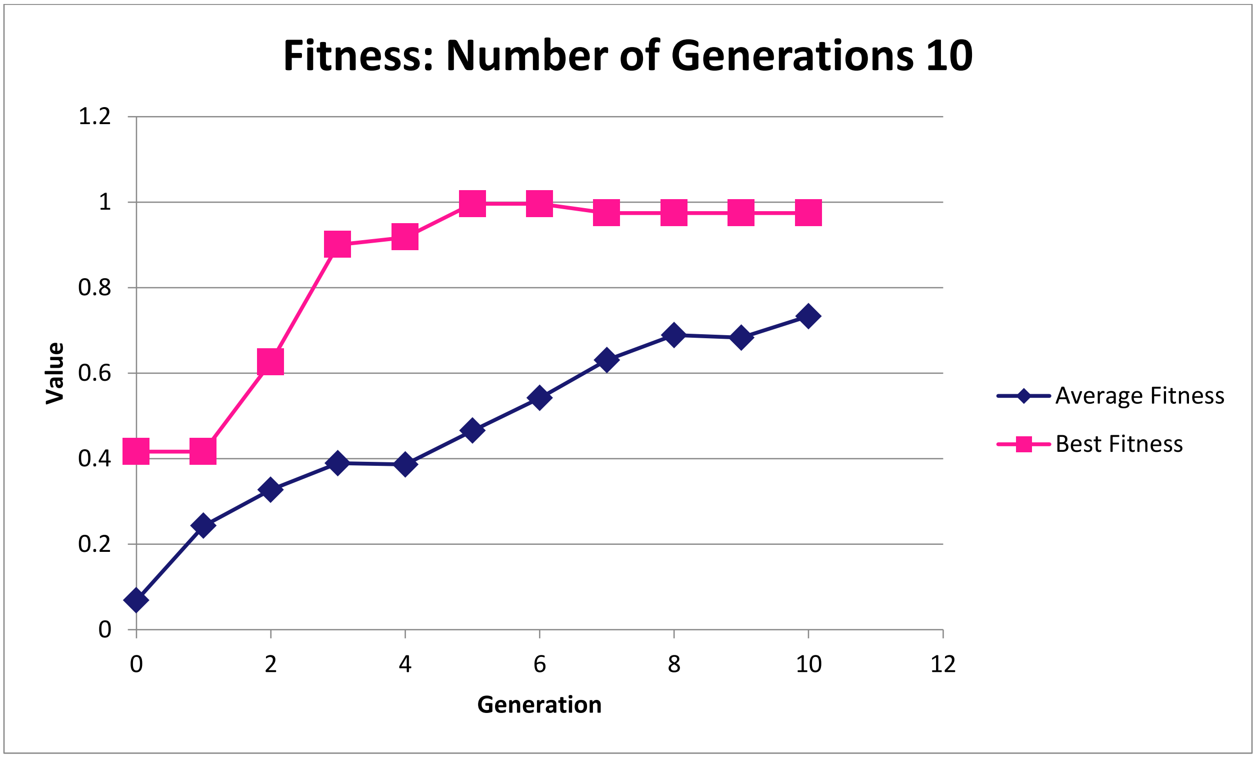

Fitness: Crossover Probability 0.6
1.2
1
0.8
0.6
0.4
0.2
0
Value
0
5
10
15
20
25
30
Generation
Average Fitness
Best Fitness

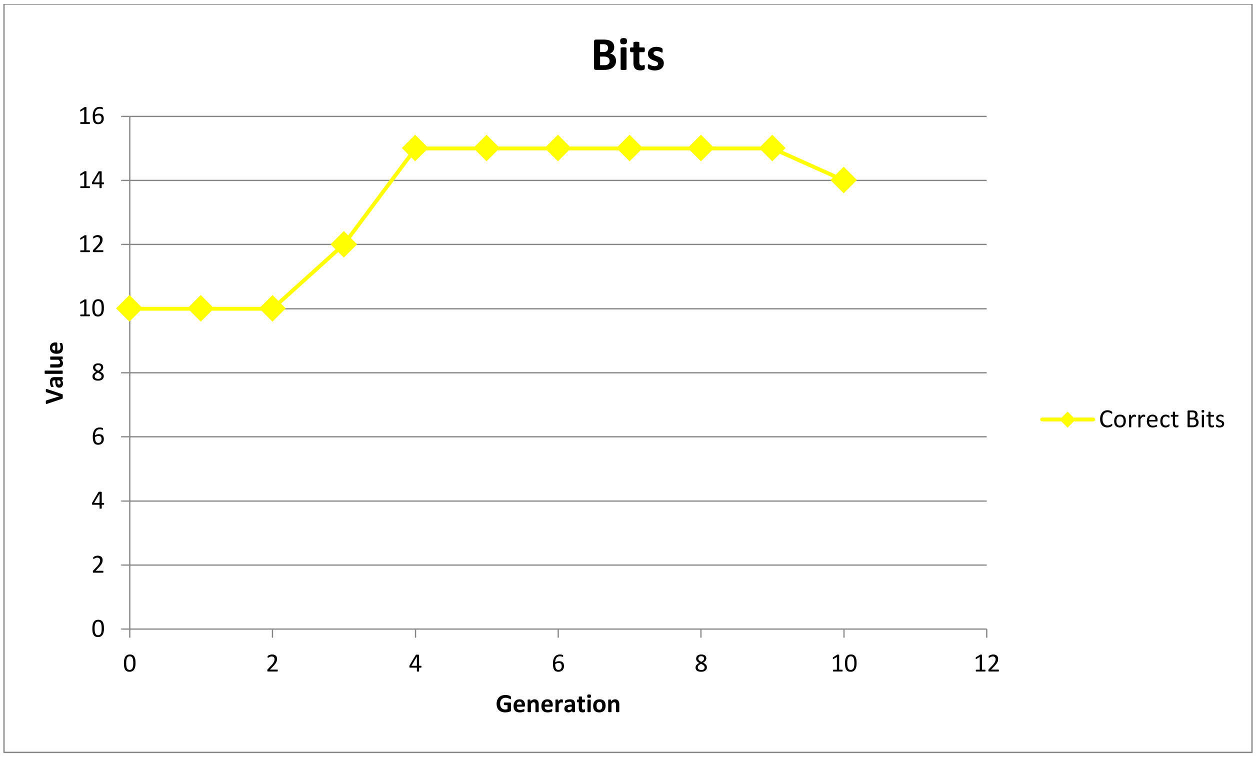

Bits
18
16
14
12
10
8
6
4
2
0
Value
0
5
10
15
20
25
30
Generation
Correct Bits

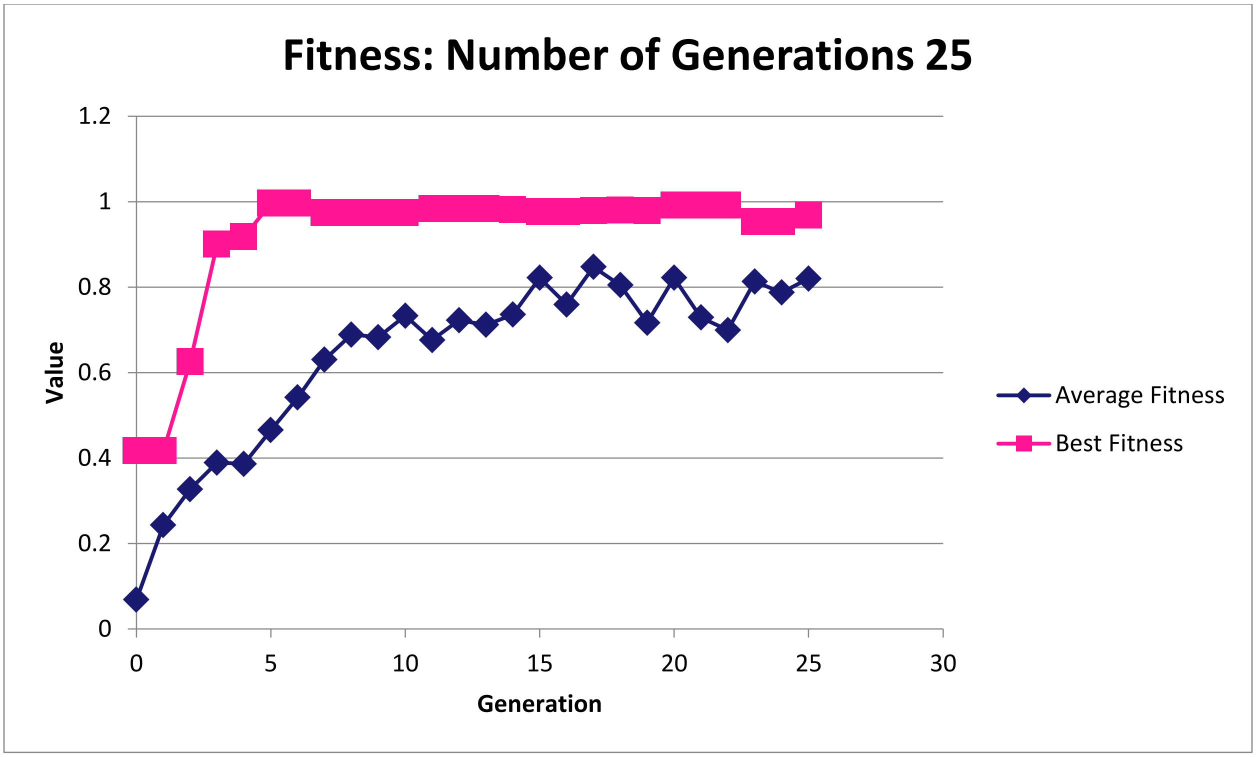

Fitness: Crossover Probability 0.8
1.2
1
0.8
0.6
0.4
0.2
0
Value
0
5
10
15
20
25
30
Generation
Average Fitness
Best Fitness

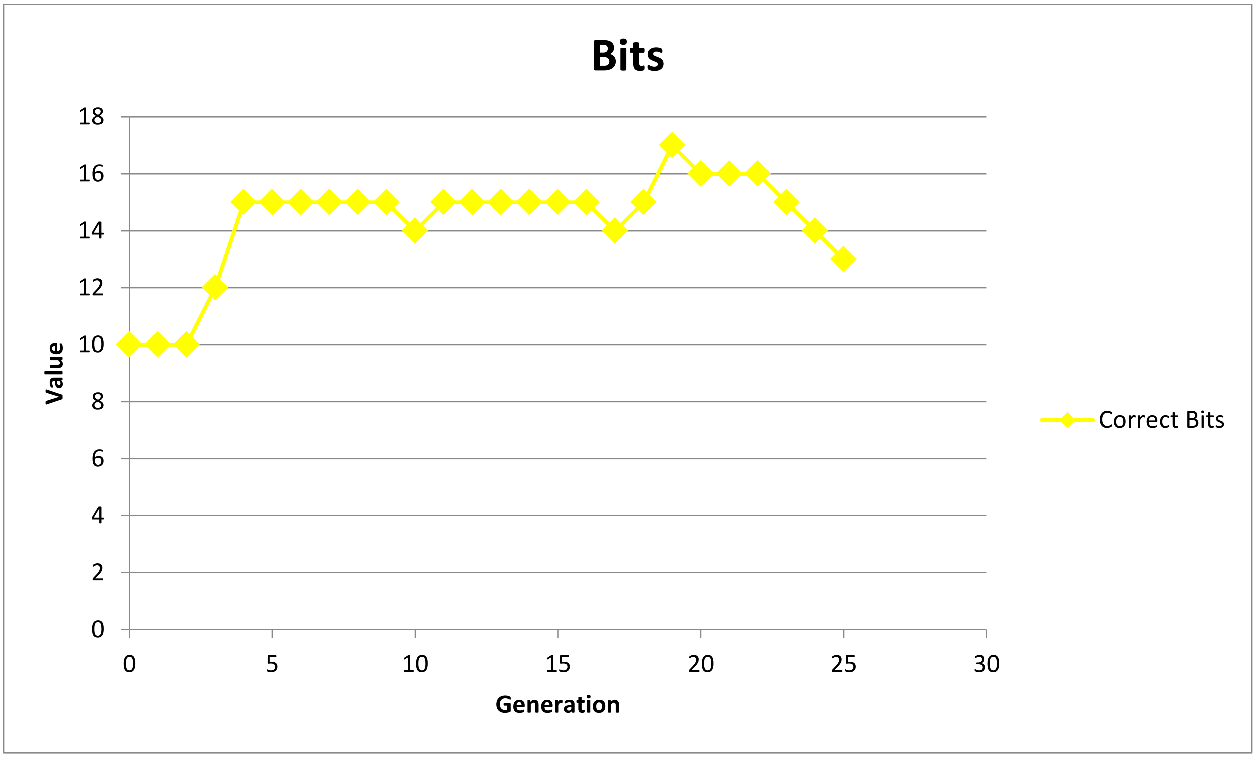

Bits
18
16
14
12
10
8
6
4
2
0
Value
0
5
10
15
20
25
30
Generation
Correct Bits

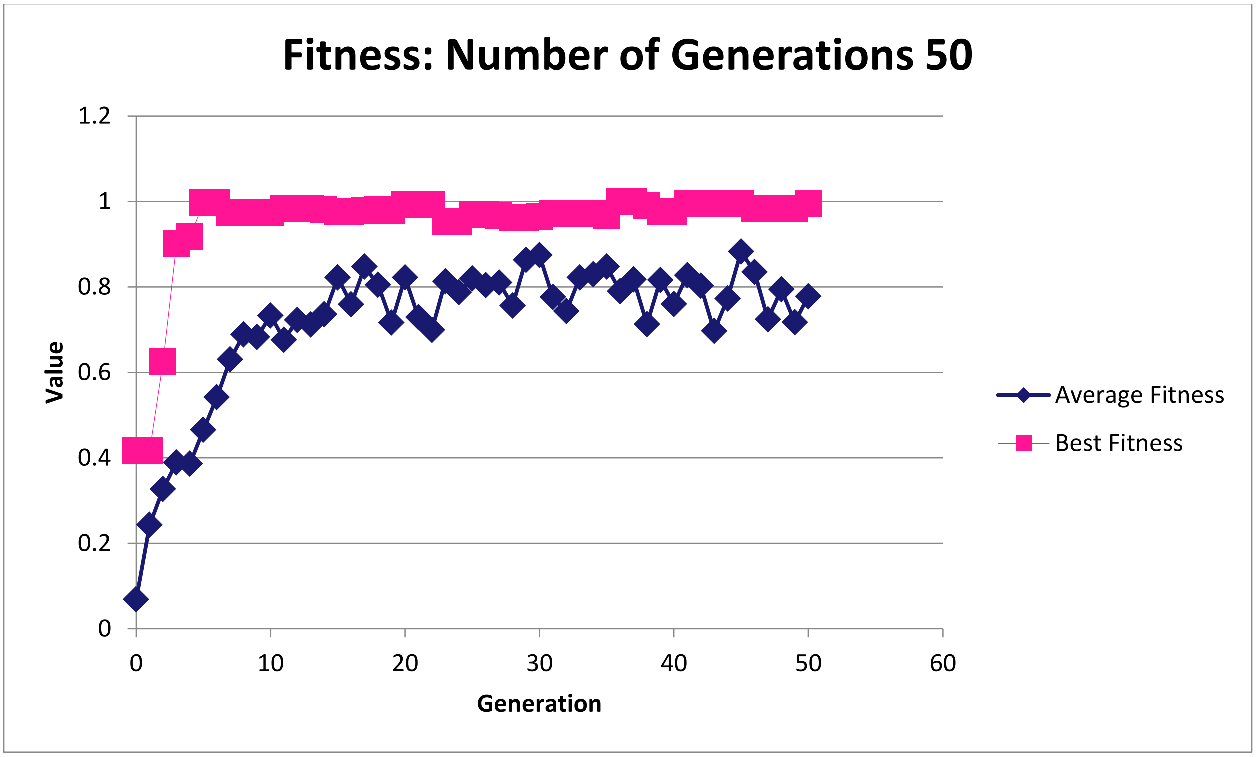
The following are the results when graphing the first, second, third and fifth variation of the mutation probability: 10, 25, 50, and 1000, respectively.

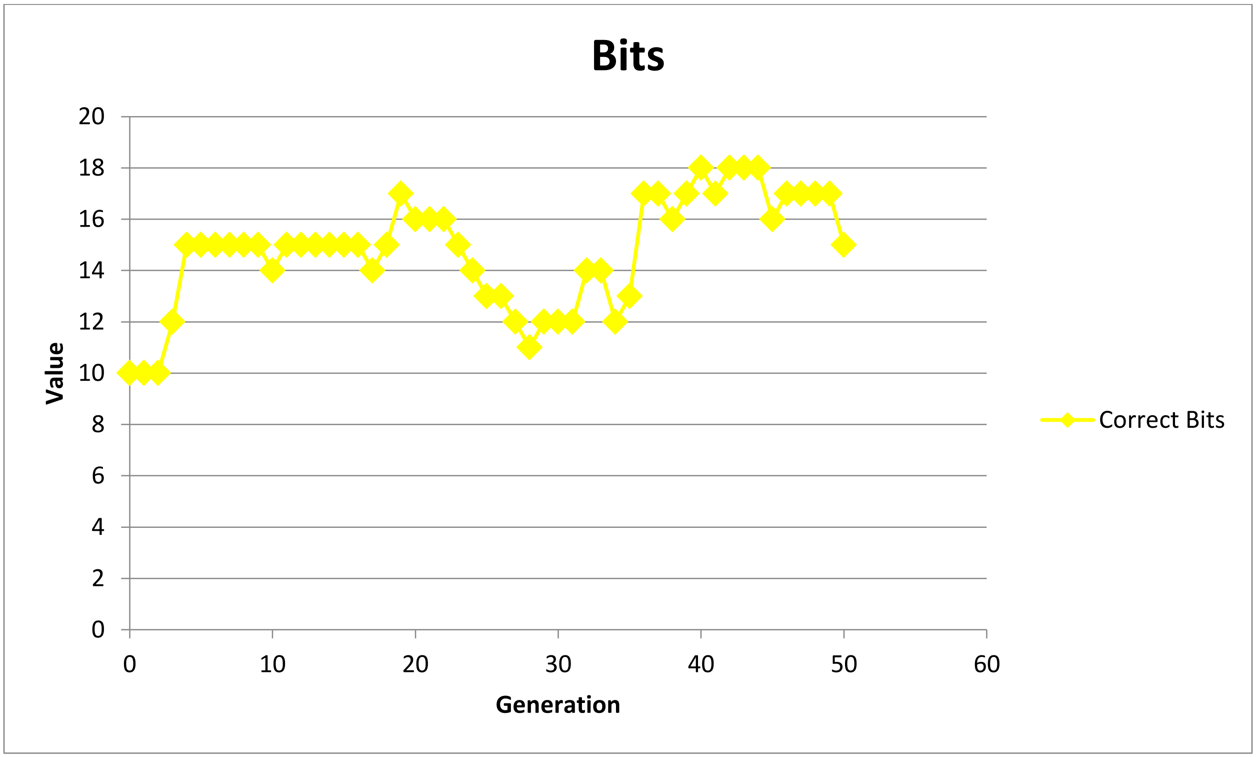


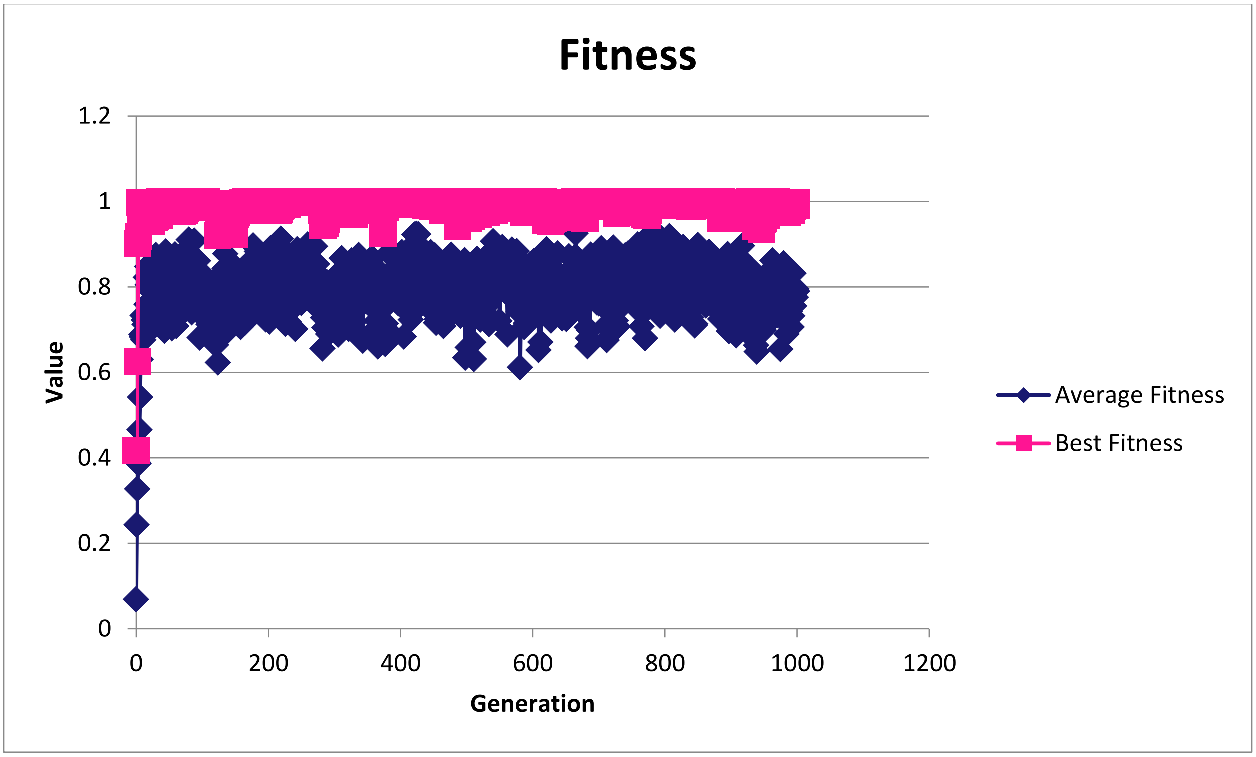


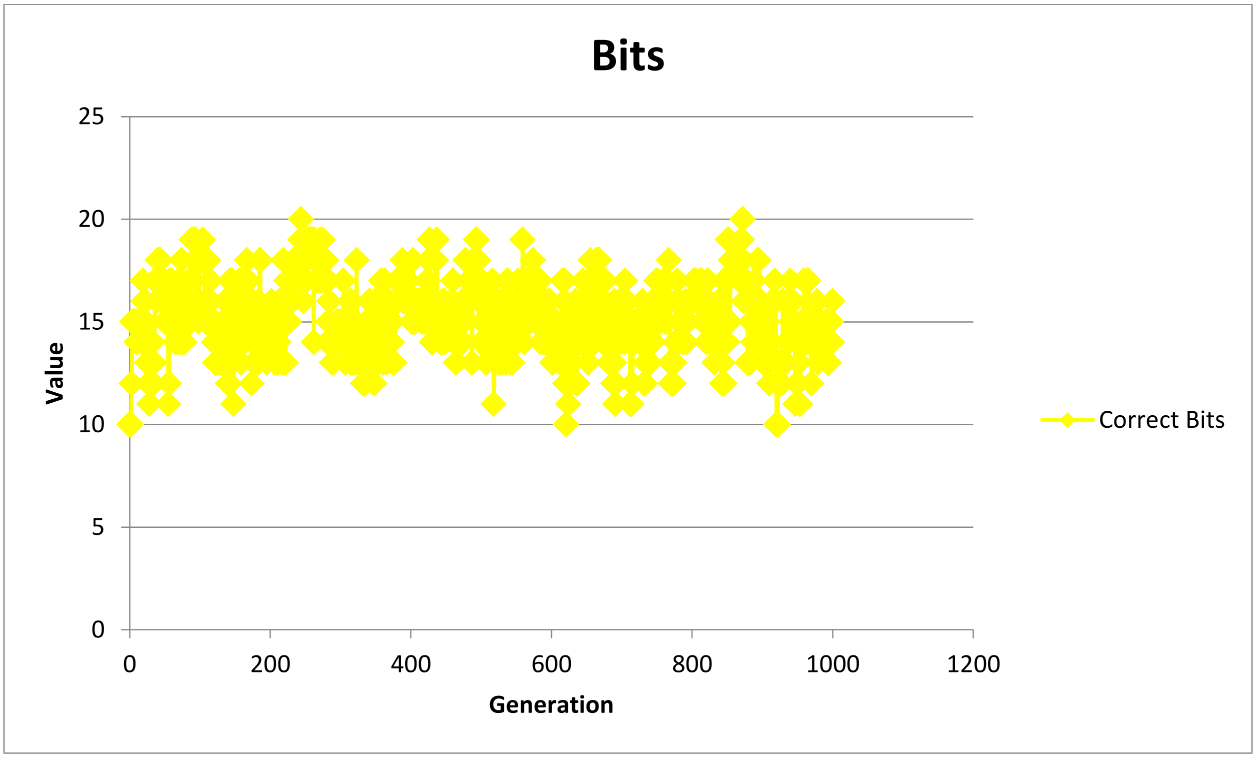

Fitness: Number of Generations 25
1.2
1
0.8
0.6
0.4
0.2
0
Value
0
5
10
15
20
25
30
Generation
Average Fitness
Best Fitness


Bits
18
16
14
12
10
8
6
4
2
0
Value
0
5
10
15
20
25
30
Generation
Correct Bits


Fitness: Number of Generations 50
Value
1.2
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
60
Generation
Average Fitness
Best Fitness


Bits
Value
20
18
16
14
12
10
8
6
4
2
0
0
10
20
30
40
50
60
Generation
Correct Bits


Fitness
1.2
1
0.8
0.6
0.4
0.2
0
Value
0
200
400
600
800
1000
1200
Generation
Average Fitness
Best Fitness


Bits
25
20
15
10
5
0
Value
0
200
400
600
800
1000
1200
Generation
Correct Bits

Discussion and Conclusions:

Below is a discussion on each variable and how it affects the fitness of a solution population over time.

Number of Genes: It can be seen from the graphs that a lower population size seems to encourage a population to increase its average fitness more quickly. This may be because with fewer genes, fewer mutations need to be made to find a better solution.

Population Size: It can be seen from the graphs that a higher population size seems to allow a population to become more fit. With more individual solutions, it is more likely that a better solution might be found via mutations.

Mutation Probability: It can be seen from the graphs that, generally, with a higher mutation probability, a population can achieve a more fit population more quickly. However, it is also seen that if the Mutation probability gets too high, the population's fitness can suffer. This variable thus must be balanced. Too little mutational probability and not enough mutations occur to increase the population's average fitness over time. Too much mutational probability, however, and a population might achieve a high level of fitness and then loose it, because the successful genes got mutated back out.

Crossover Probability: It can be seen from the graphs that crossover probability affected the populations in a strange way. When a child string has its genes crossed over, the less important genes are swapped with the most important genes. Thus it can be seen that when this happened, populations took sudden jumps. If these jumps occurred toward the beginning, they were more influential, as the two ends of the string of genes were less similar. As time went on and the population had more "correct" bits overall, the crossover probability was less influential.

Number of Generations: It can be seen from the graphs that as the number of generations grew, the effect of each generation became less and less. The fitness of the best individual hit above 0.9 after about 5 or 6 generations, and the average fitness of the population continued to grow at a steady rate until about after 25 to 30 generations. After that point, it can be seen that the average fitness of the population does not increase much, if at all, and continues to hold an average of about 0.75. It is important to note here that our random numbers were seeded by time, and that each simulation had a (theoretically) different seed. Thus the ten generations of the first configuration and the first ten generations of the fifth configuration are not inherently the same, but still behave similarly, meaning that the behavior of these simulations was regular.

As for the quality of this type of algorithm, it can be seen that it is well suited for optimization problems, as solution finesses trend upward over time. It must be noted here something regarding the correct bits of the most fit individual over time. It can be seen from most plots that this number tends to trend upward over time, but this is not a sure thing. This behavior is, in fact, indicative of the poor quality of the fitness function that was used for this inquiry. The nature of the way we evaluated individual solutions was not particularly optimal for this type of algorithm. Because the fitness of a string was based on the integer interpretation of that string's binary genes, each successive gene was doubly as important in terms of fitness as the one before it. This meant that a mutation of a gene near the one end of a string was much more influential than a mutation at the other end. When a solution string was crossed over (with a higher crossover probability), if the one side of the string was significantly more or less fit than the other side, the effect of the crossover was much more significant. Returning to the bits in the correct individual, it is seen that even though the strongest individual was trending upward, the number of correct bits might not be. This is because the Number of Correct Bits metric does not take into account the position of the correct bits. Thus it can be seen that this metric does not provide very much helpful information.