Xiao Zhou

Project 5: Genetic Algorithms

Introduction

In this project, a basic genetic algorithm is implemented to simulate mating of individuals in a population over generations (with certain crossover and mutation probabilities) based on the individual fitness values calculated using a fitness function. The goal is to explore the effects of population size *N*, mutation probability p_m, crossover probability p_c, genetic string length *l*, and number of generations *G* have on the behavior of the genetic algorithm.

Method

The basic genetic algorithm is set to have a certain population size *N*, mutation probability p_m, crossover probability p_c, genetic string length *l*, and number of generations *G*. Each individual is represented by binary chars ('0's and '1's) of length *l*, and there are *N* individuals in each generation. The first generation is generated randomly to have '0's and '1's in its genetic bits. Then the fitness of each individual is calculated by first calculating the integer equivalent of its binary sequence (convert binary number represented by each individual to its base 10 equivalent), and then the integer value is passed in the following real-valued fitness **Equation (1)** in place of *x*, where *l* is the genetic string length. This function generates greater fitness values for binary strings that have 1's in the more significant digits (optimum genotype is all 1's).

$$F(s) = \left(\frac{x}{2^l}\right)^{10}, \qquad \textbf{Equation (1)}$$

A running sum of the total fitness is kept and used to calculate normalized fitness value for each individual by dividing the fitness value calculated from the equation above by the total fitness of the population. A running normalized fitness total is also kept to help with parent selection.

Two individuals are selected to be parents by randomly generating numbers between 0 and 1 and see which range the random numbers fall into based on the running normalized fitness total. Pick the individual that's represented by the lower bound of the random number for a parent. And this is done so that 2 parents are not the same individual during each mating (one cannot mate with itself).

During the mating of two parents, crossover or no crossover can take place. This is determined using a random number generated, and if the random number is less than the crossover probability (p_c) or if p_c=1, then crossover takes place (and vice versa). If crossover takes place, then the point of crossover is determined randomly and two children are produced by exchanging the bits of the parents at the point of crossover. If there's no crossover, the two children are simply copies of the parents. After that, there's possibility for mutation on each digit/char of the individual children. And this is determined by a random number generated between 0 and 1 and p_m (if p_m=1 or if random number < p_m).

Each couple mates to produce 2 children, who replace them in the new generation. This mating process is done *N/2* times so that the new generation is filled by children of old generation. After each generation, the average fitness, best fitness, and the number of correct bits (1's) in the fittest individual are calculated. This is done for *G* number of generations.

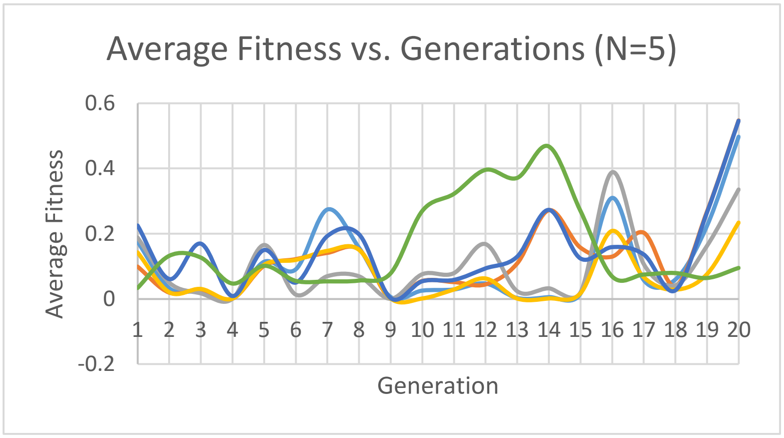
This genetic algorithm is implemented in C++ (see "ga.cpp" file) and takes the genetic string length *l*, population size *N*, number of generations *G*, mutation probability p_m, crossover probability p_c, and seed for random number generator by command line arguments, in this particular order. To use the program, simply compile with g++, and then run. The population in each generation, along with its statistics are displayed on standard output and the average fitness, best fitness, and number of correct bits in most fit individual for each generation will be written to a .csv (see sample run below).

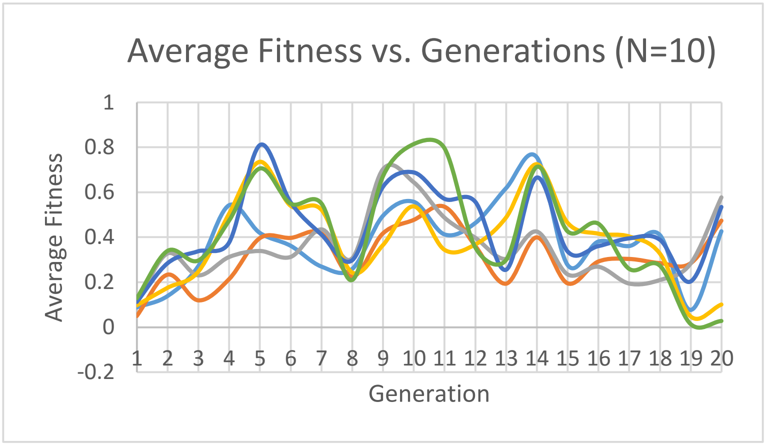
```
UNIX> g++ -o ga ga.cpp
UNIX> ./ga 20 30 20 0.033 0.6 0
Number of genes(bits) in genetic string: 20
Population size N: 30
Number of generations G: 20
Mutation probability pm: 0.033
Crossover probability pc: 0.6
Seed for random number generator: 0

1st Generation:
10111100110101100000
10110001111000111010
…
```

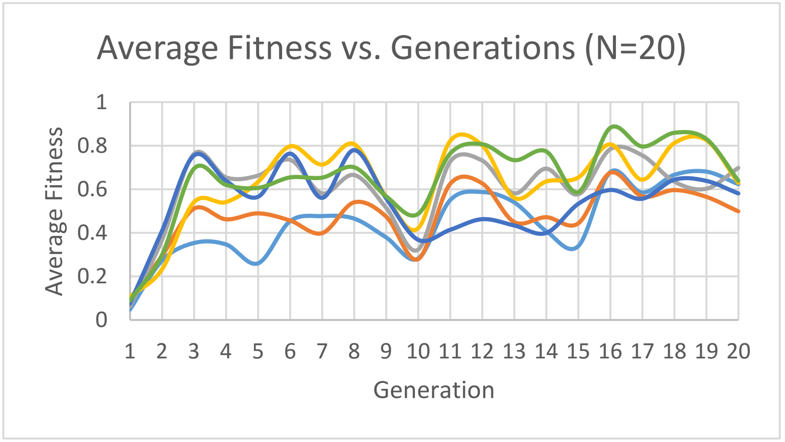
To test the effect of population size *N* on the behavior of the genetic algorithm, *N*=5, 10, 20, 40, and 80 are used while *l*=20, *G*=20, p_m=1/*N*, and p_c=0.6. To vary mutation probability, p_m=0, 0.03, 0.1, 0.5, and 1.0 are used while *l*=20, *N*=30, *G*=20, and p_c=0.6. To vary crossover probability, p_c=0, 0.2, 0.4, 0.8, and 1.0 are used while *l*=20, *N*=30, *G*=20, and p_m=0.033. To look at the effect of genetic string length, *l*=5, 10, 20, 30, and 40 are used while *N*=30, *G*=20, p_m=0.033 and p_c=0.6. And to vary number of generations, *G*=10, 20, 30, 40 and 50 are used while *l*=20, *N*=30, p_m=0.033 and p_c=0.6. Six runs were done for each parameter variation.

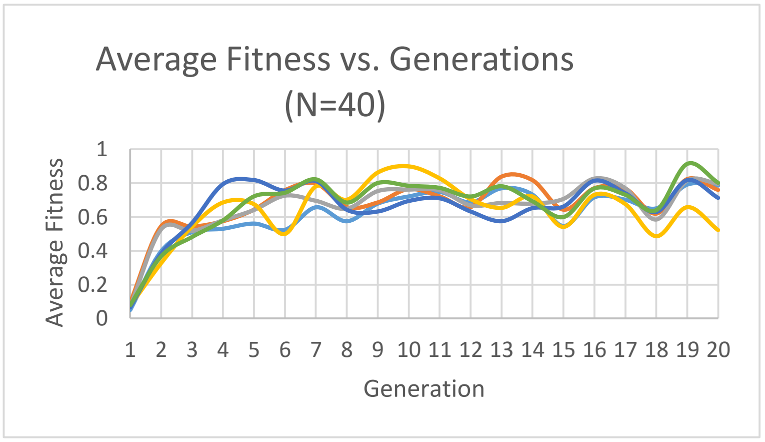
Data

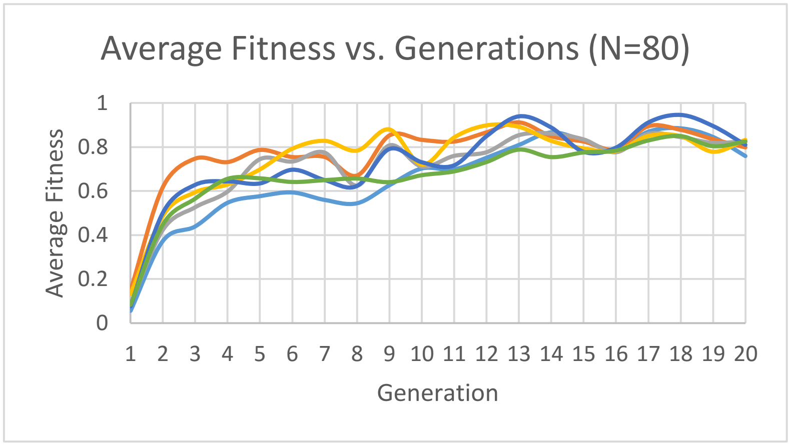
Varying population size N from 5, 10, 20, 40, and 80 gave the following data (average fitness, best fitness, and number of correct bits in most fit individual vs. time graphs that connected the data points, see **Figures 1**, **2**, and **3**):



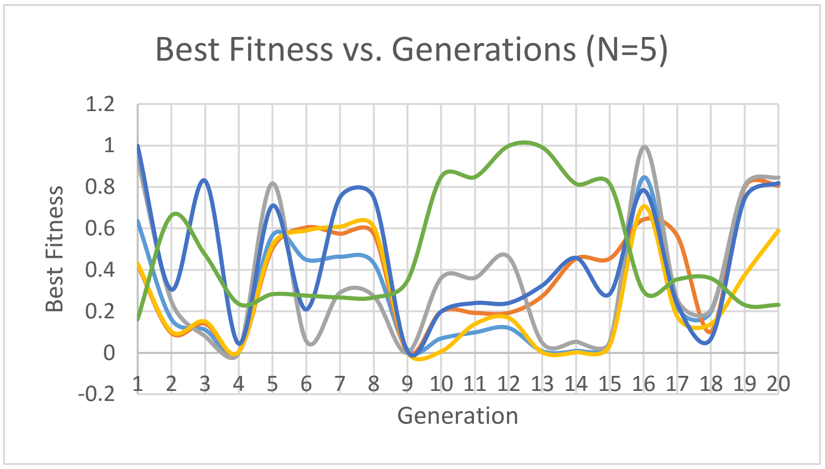


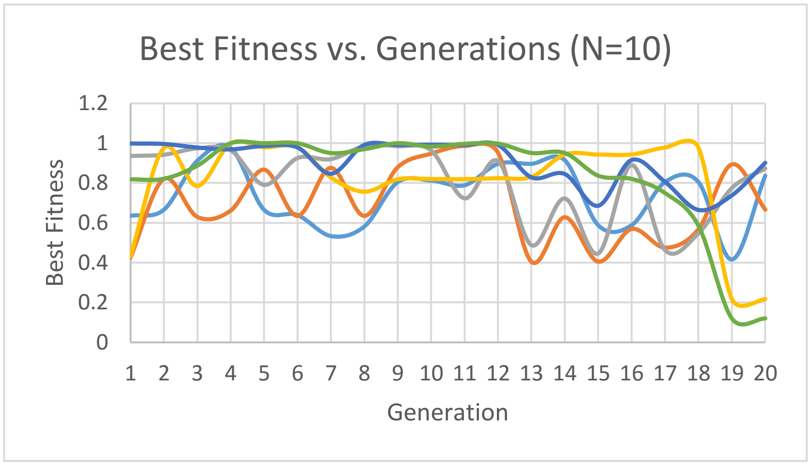


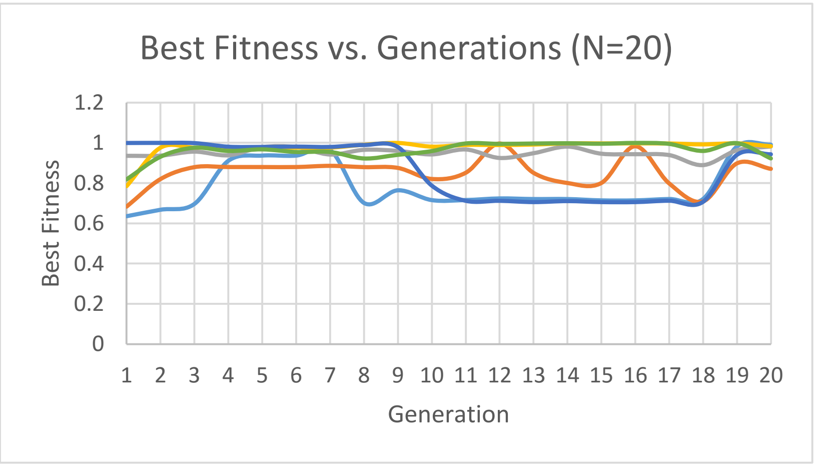


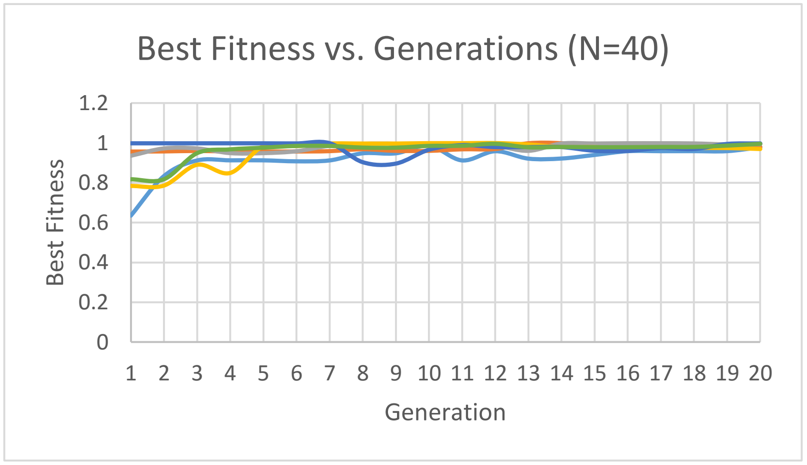


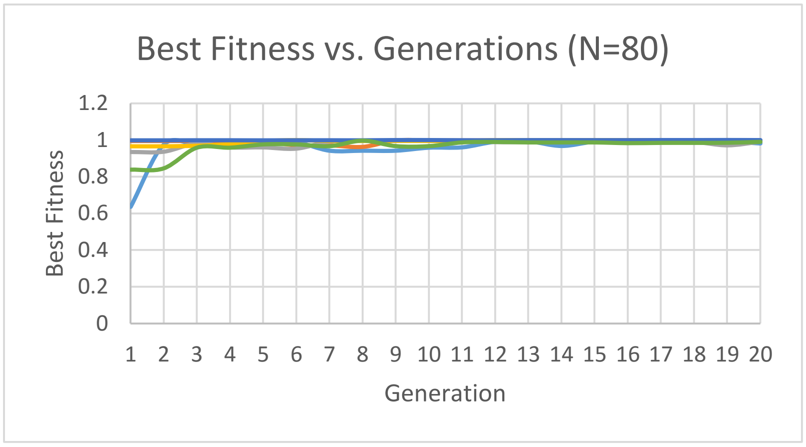
Figures 1: Average Fitness vs. Generations with Different Population Sizes



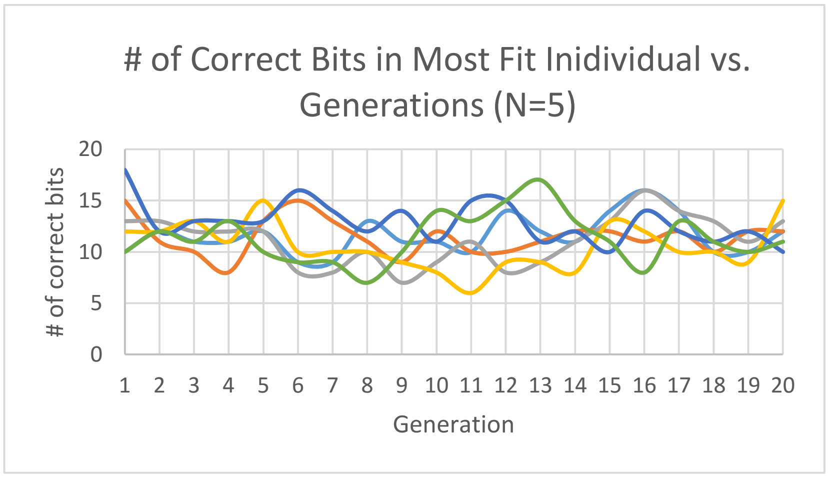


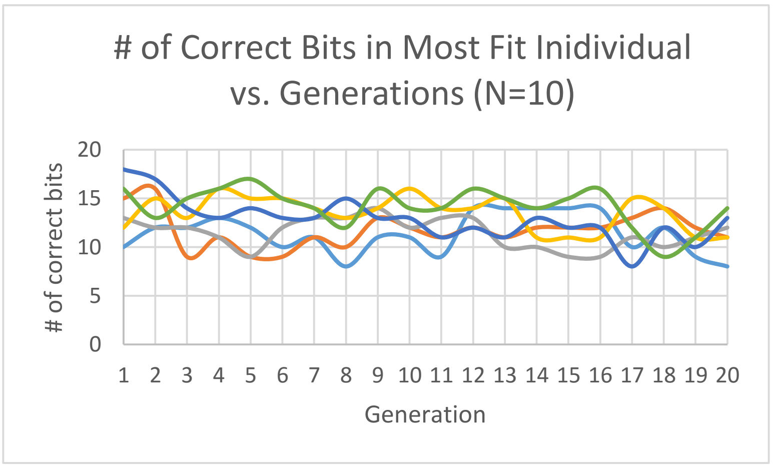


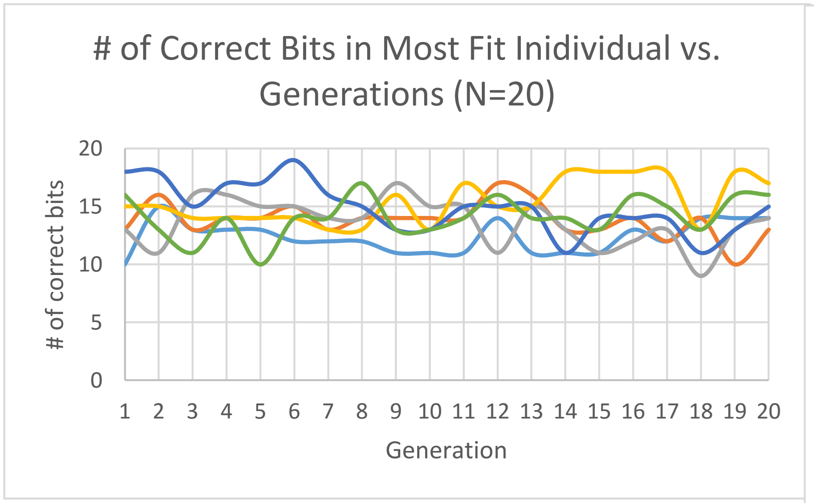


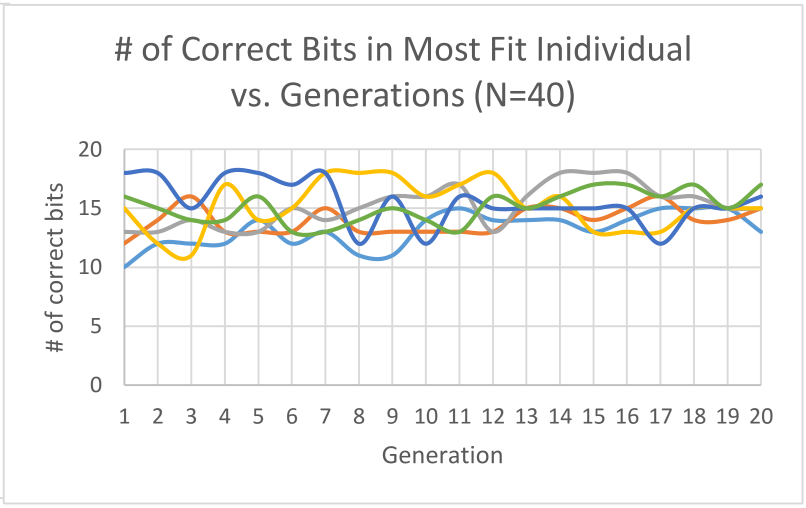


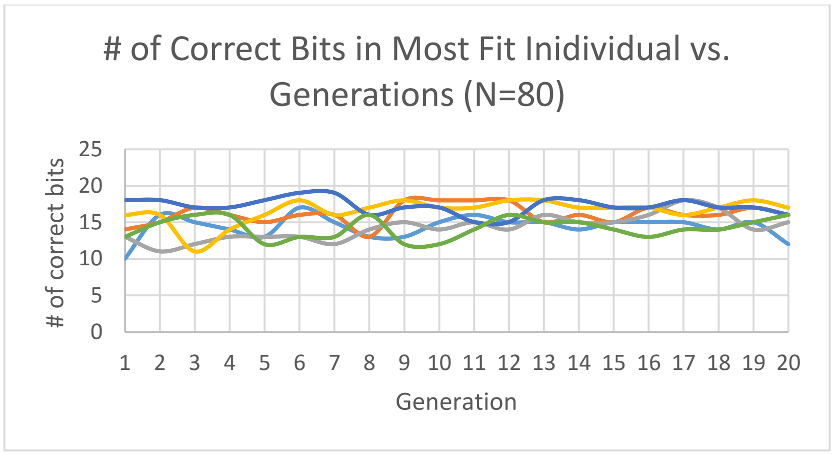
Figures 2: Best Fitness vs. Generation with Different Population Sizes





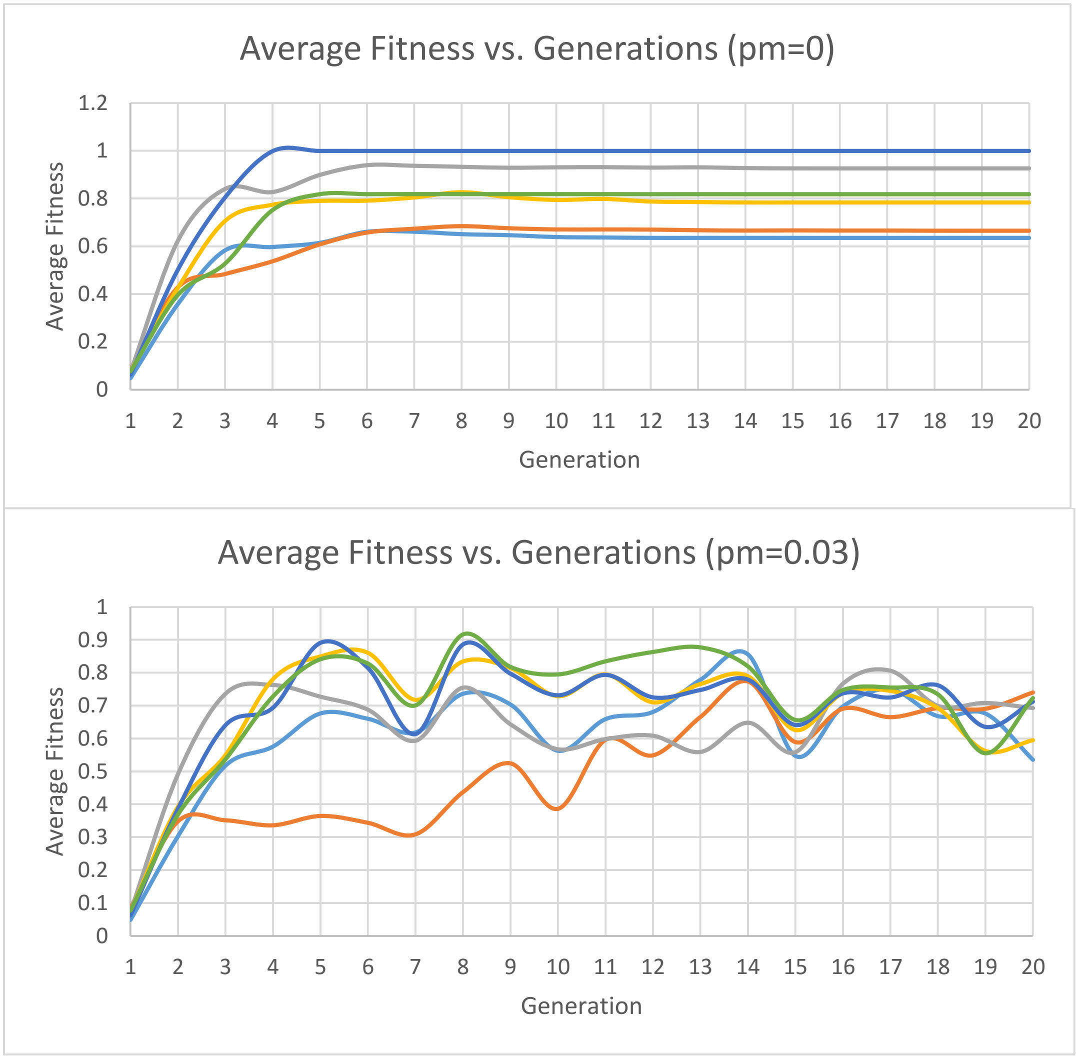


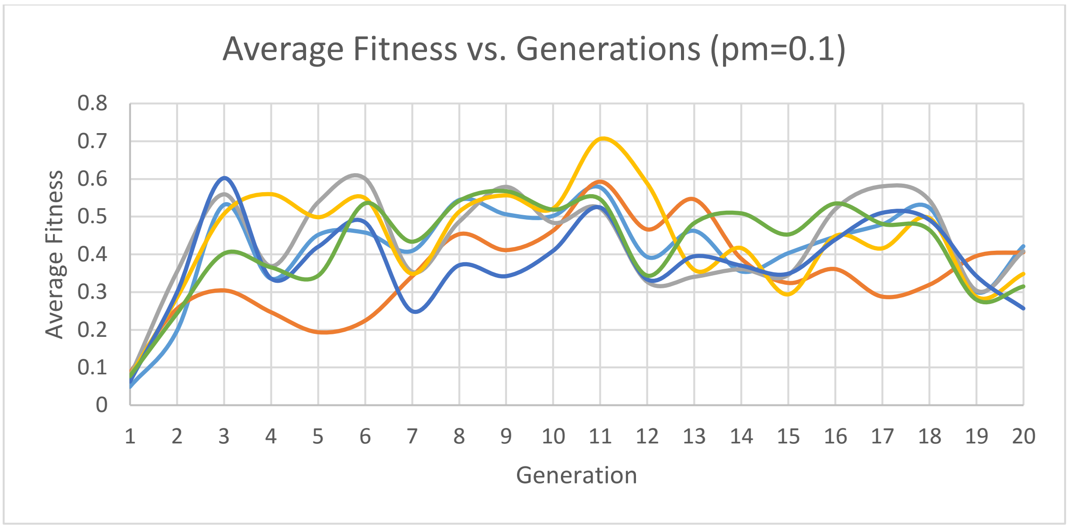


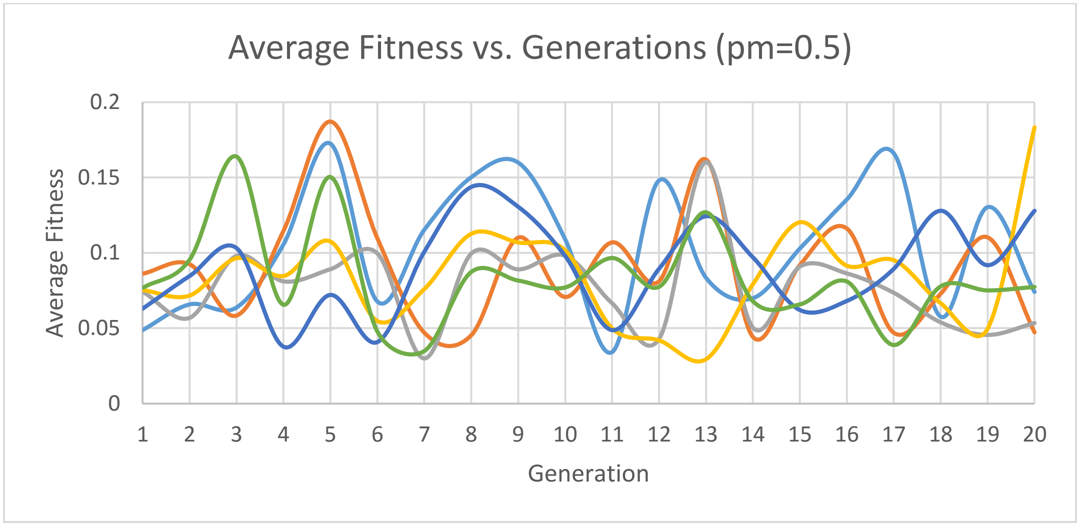


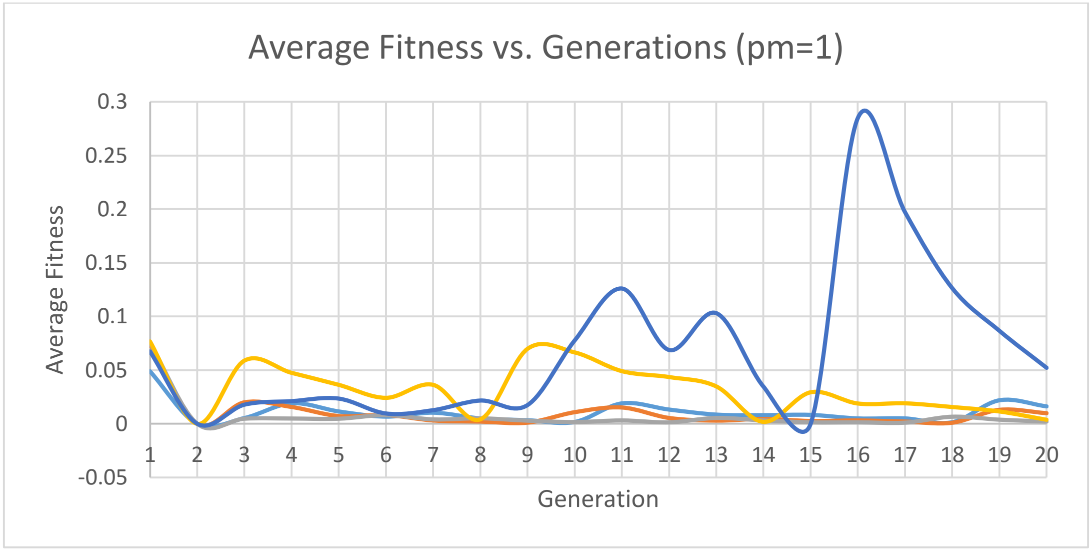
Figures 3: # of correct bits in most fit individual vs. generation with different population sizes

Figures 4, **5**, and **6** are data collected when mutation probability is varied (see method section for full parameter list) from 0 to 0.03 to 0.1 to 0.5 to 1.

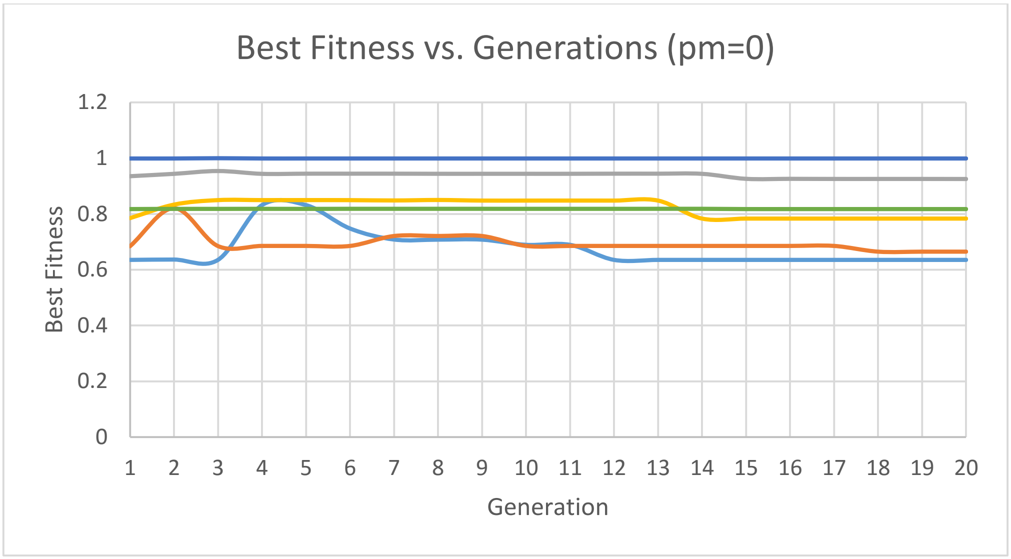




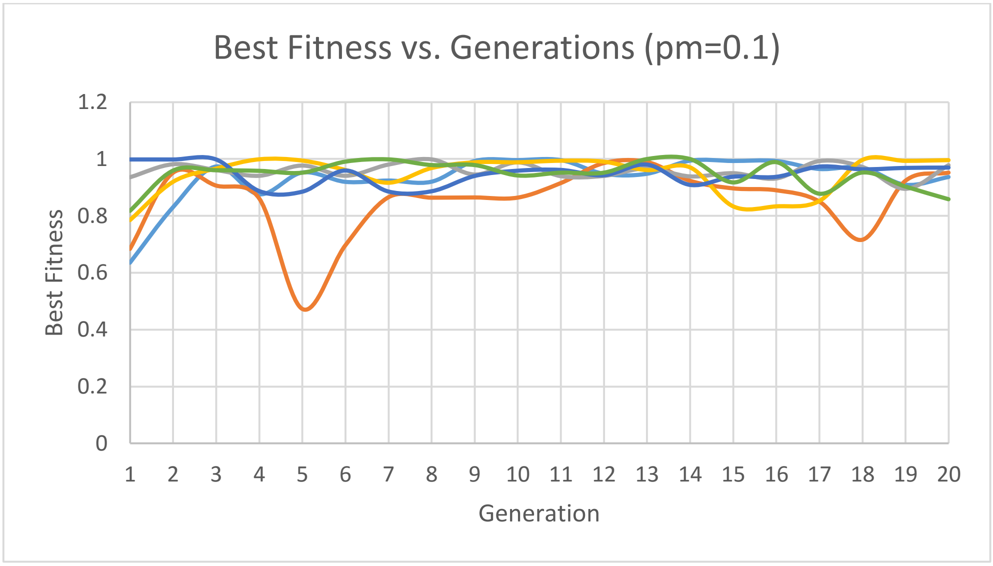




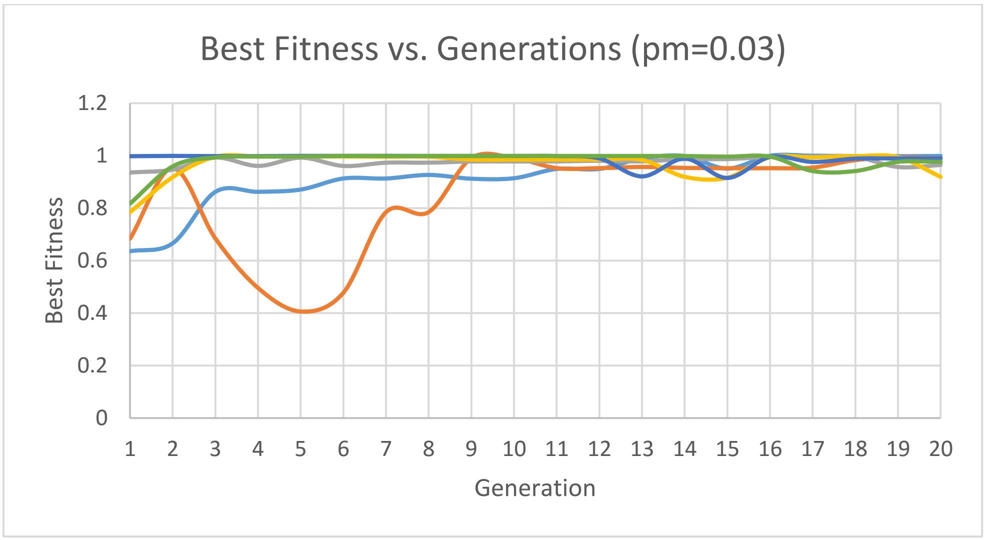
Figures 4: average fitness vs. generation for different mutation probabilities



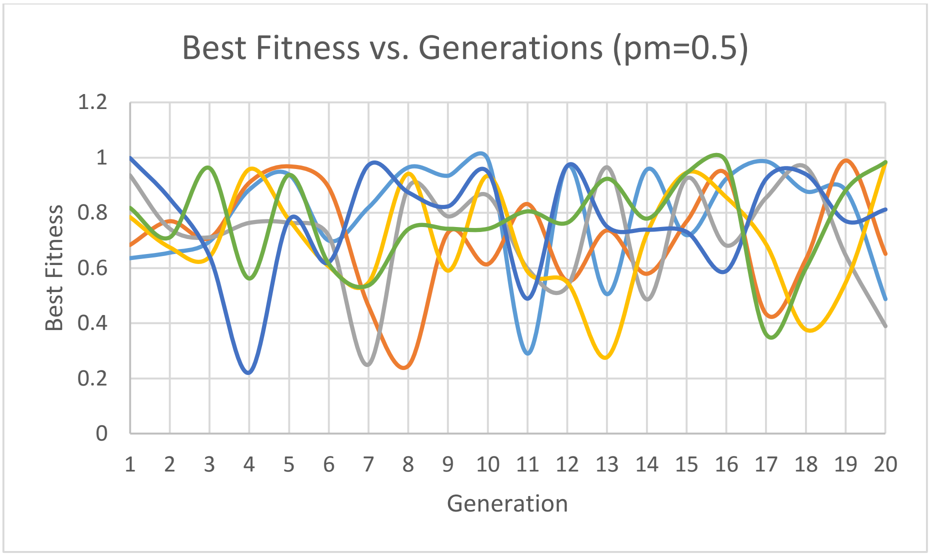
Best Fitness vs. Generations (pm=0)
Best Fitness
Generation

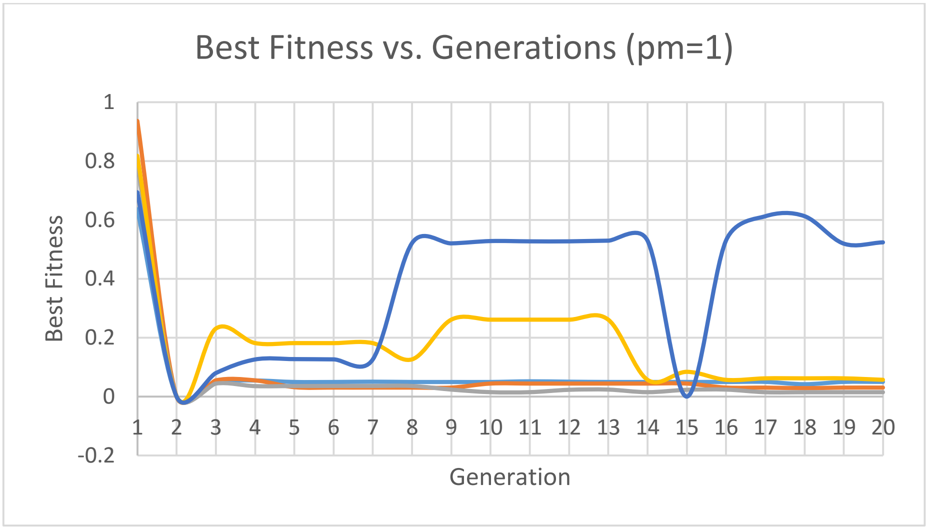


Best Fitness vs. Generations (pm=0.1)
Best Fitness
Generation

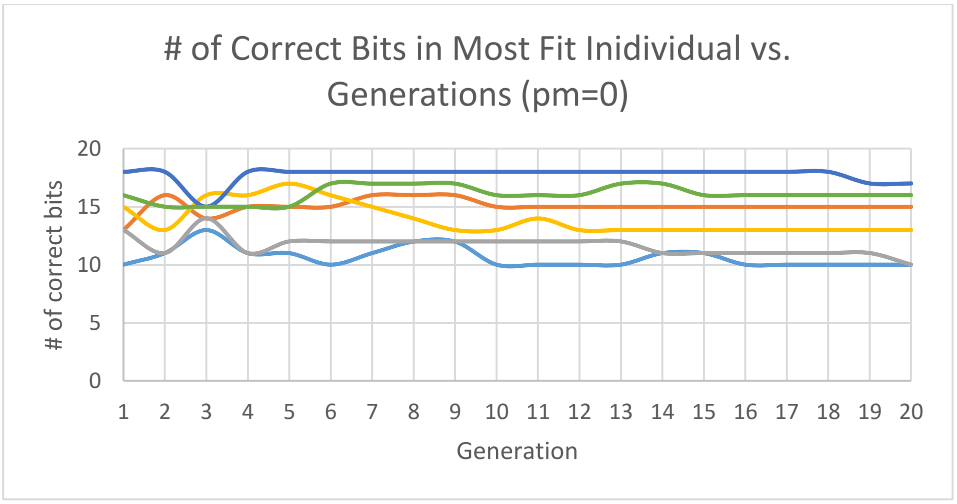


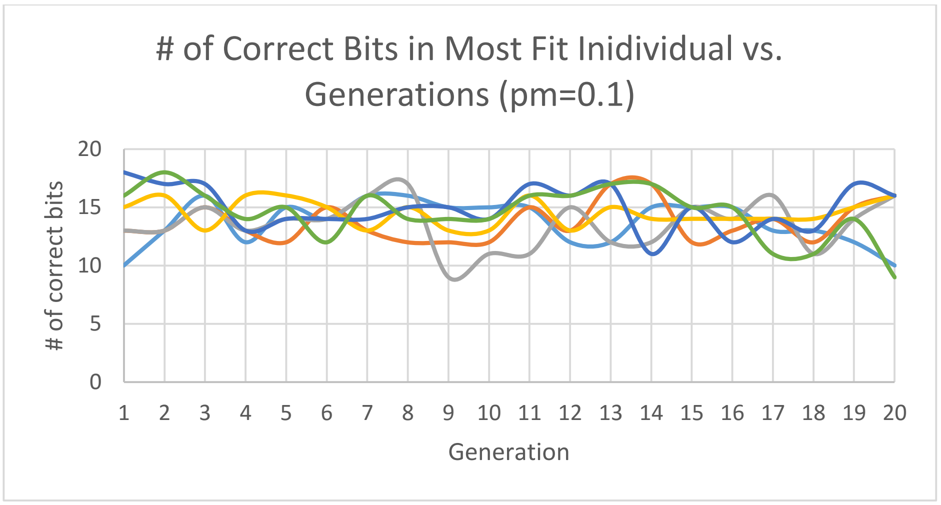
Best Fitness vs. Generations (pm=0.03)
Best Fitness
Generation



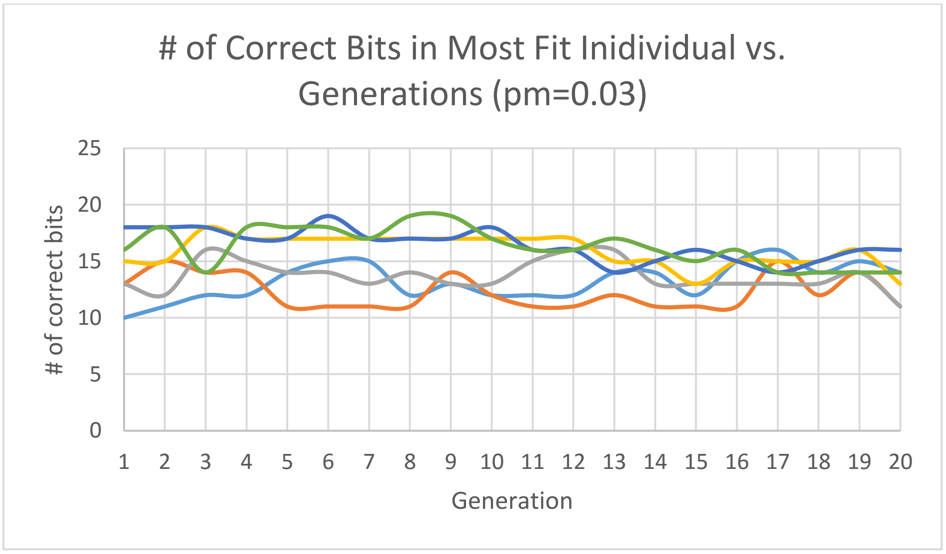


Figures 5: best fitness vs. generation for different mutation probabilities

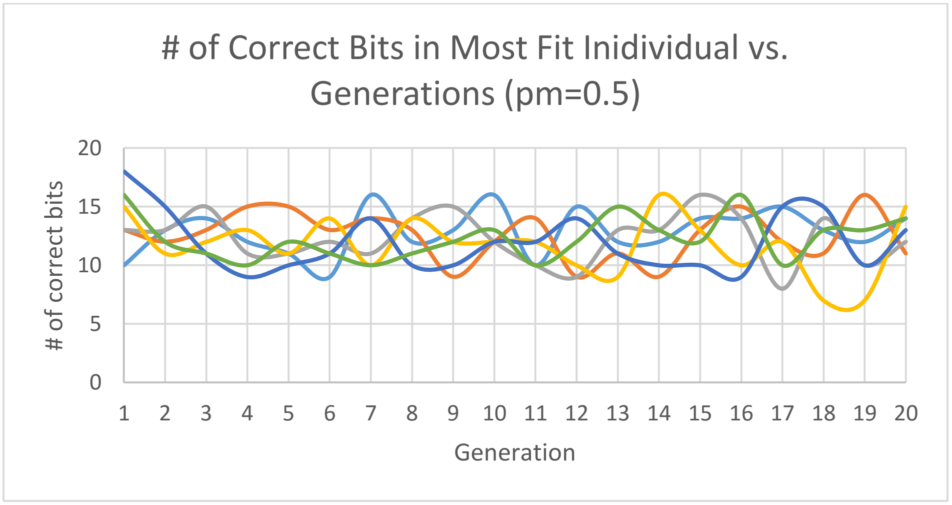




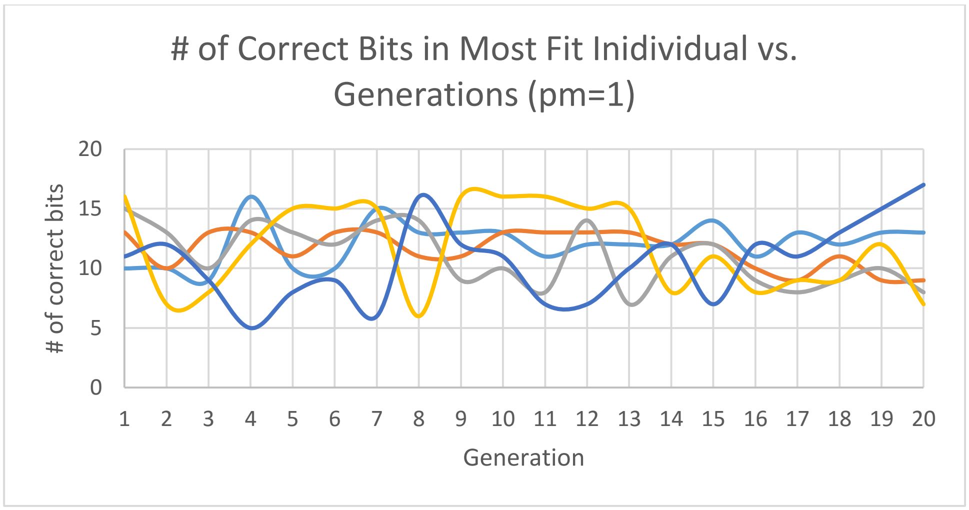
# of Correct Bits in Most Fit Inidividual vs. Generations (pm=0.1)
# of correct bits
Generation
0
5
10
15
20
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20



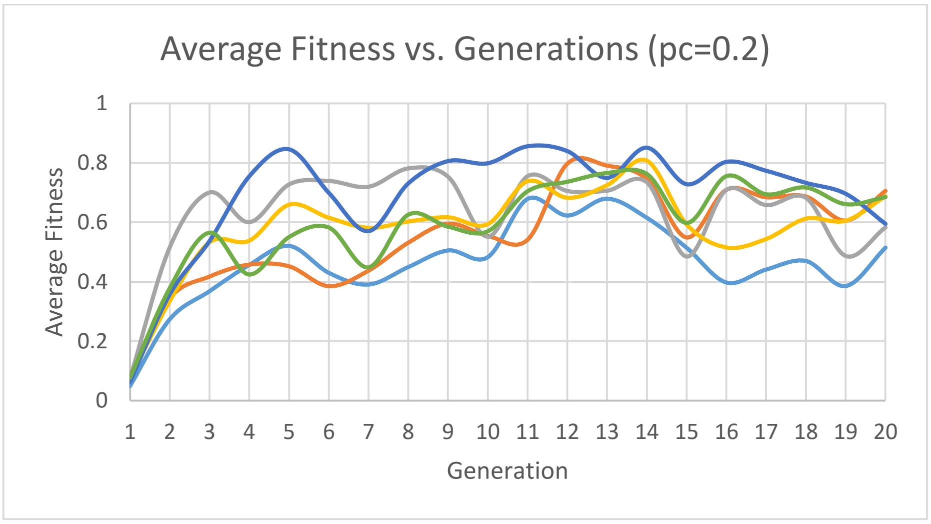
# of Correct Bits in Most Fit Inidividual vs. Generations (pm=0.03)
# of correct bits
Generation
0
5
10
15
20
25
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

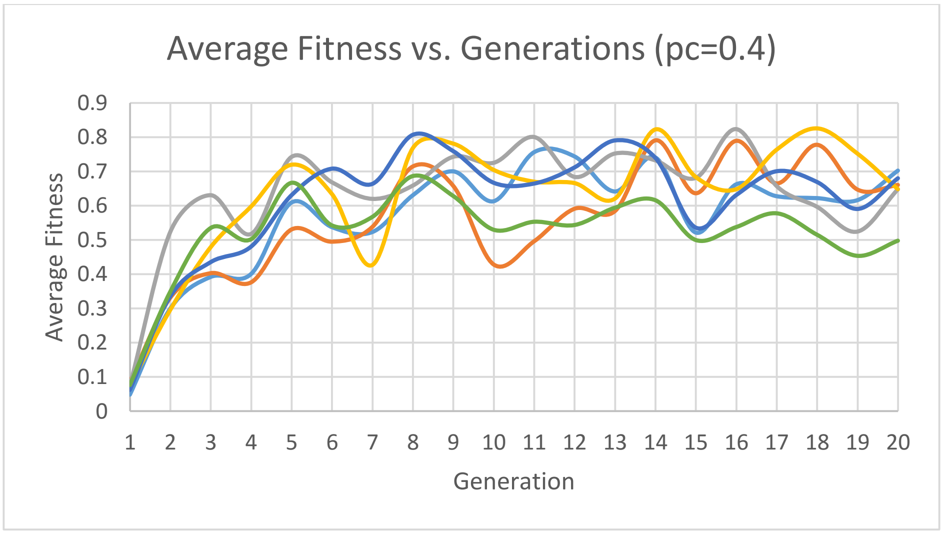


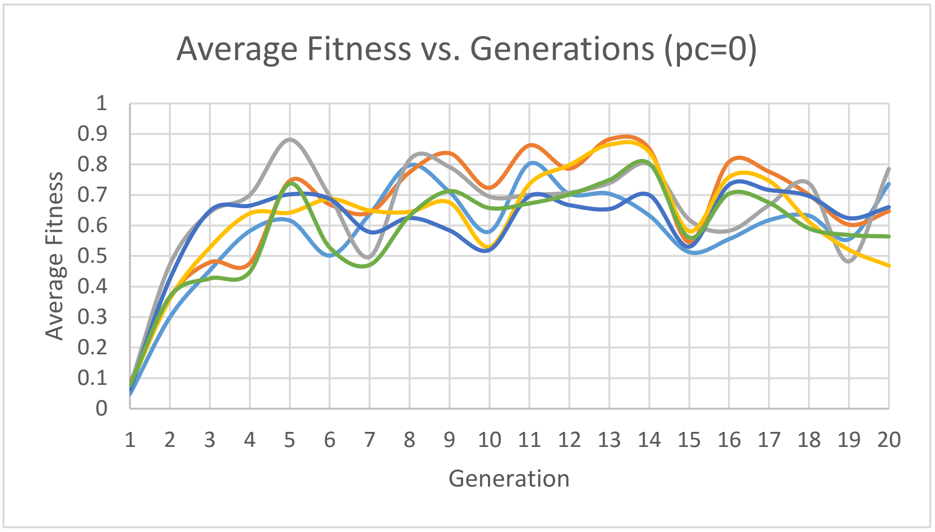
# of Correct Bits in Most Fit Inidividual vs. Generations (pm=0.5)
# of correct bits
Generation
0
5
10
15
20
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

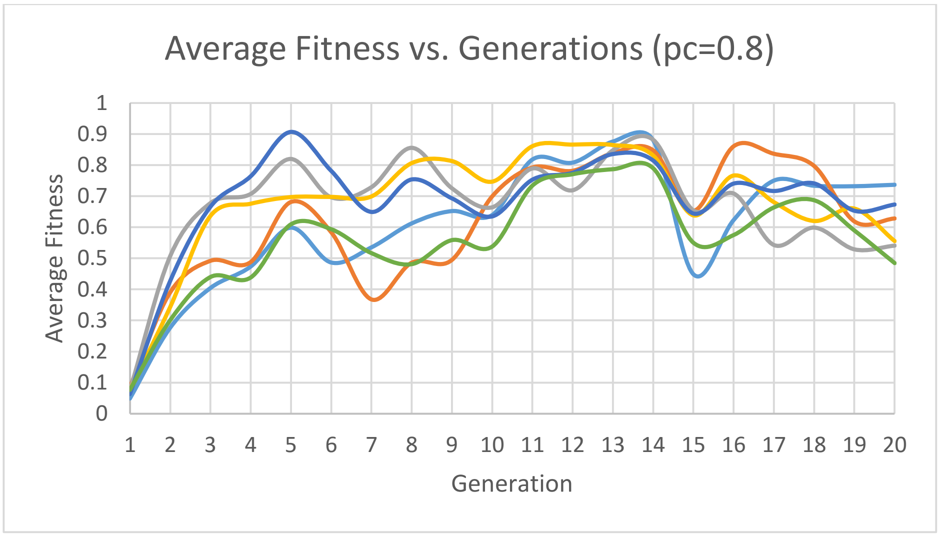


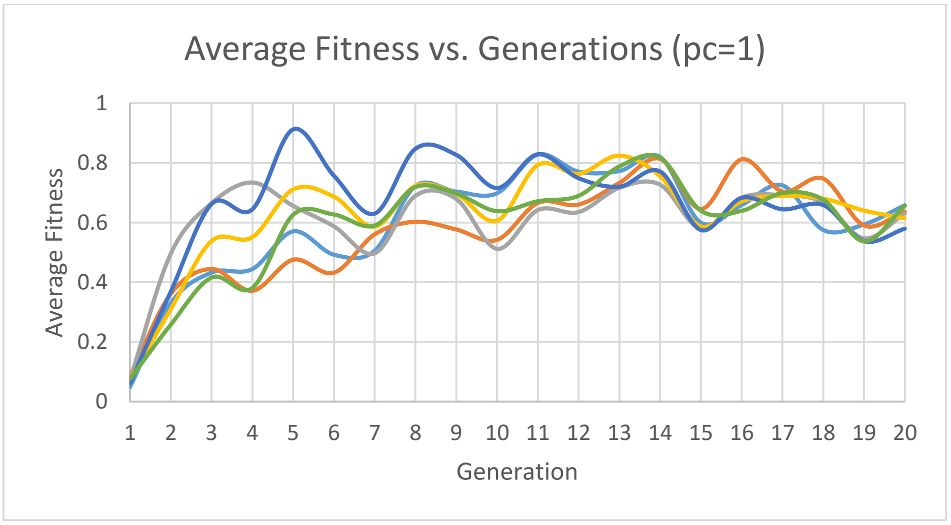
Figures 6: # of correct bits in most fit vs. generation for different mutation probabilities



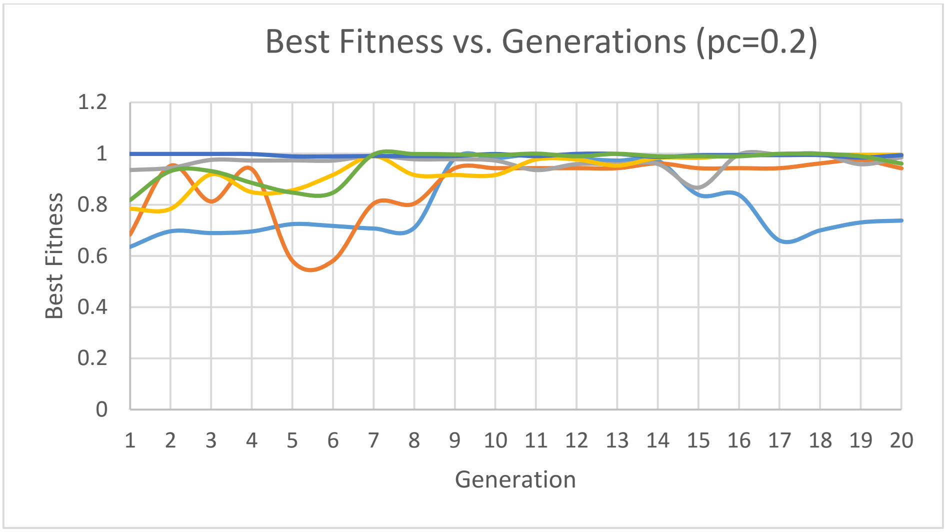


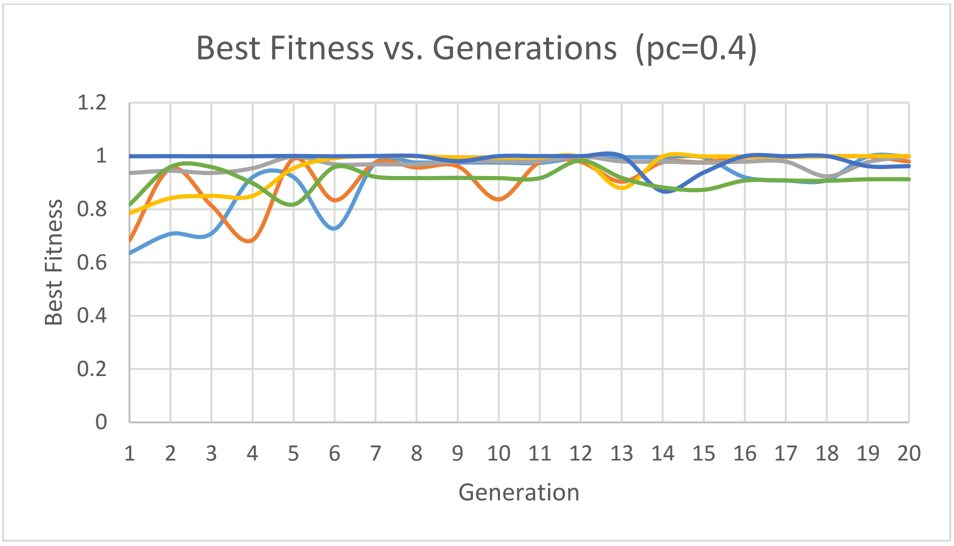


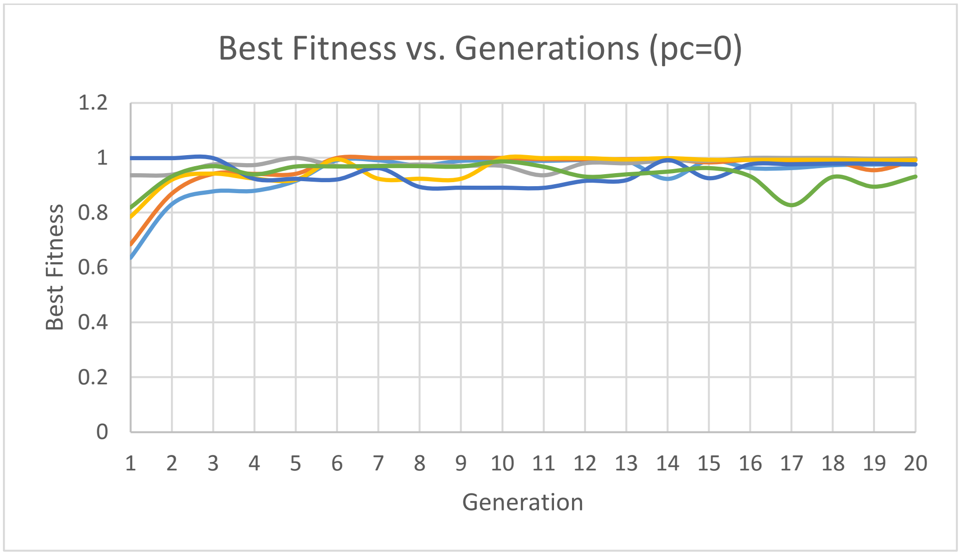


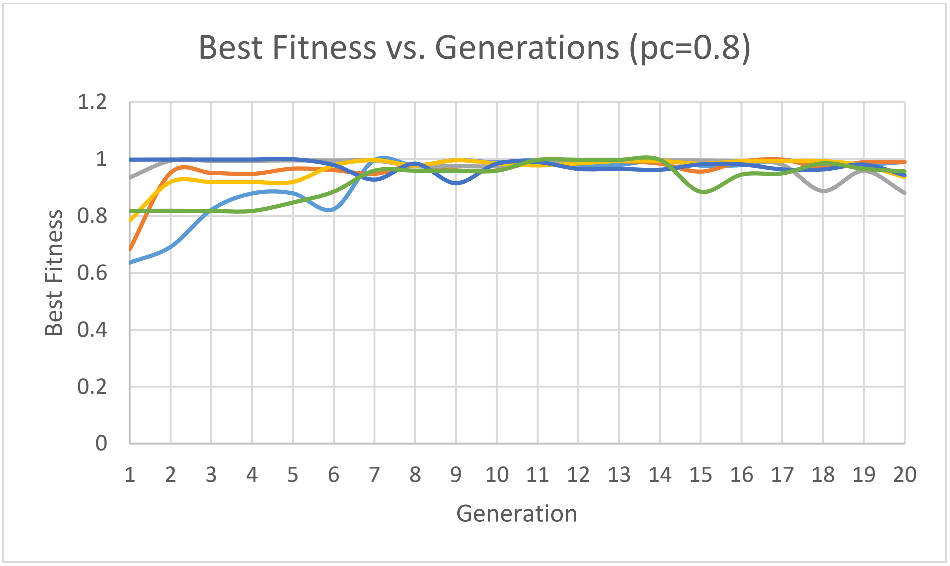


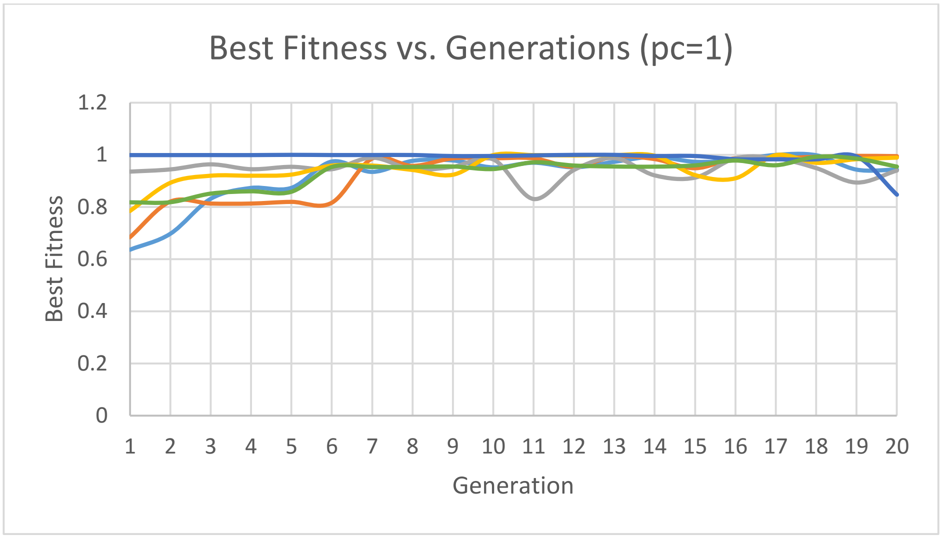
Figures 7: average fitness vs. generations with different crossover probabilities


Best Fitness vs. Generations (pc=0.2)
Best Fitness
Generation
0
0.2
0.4
0.6
0.8
1
1.2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

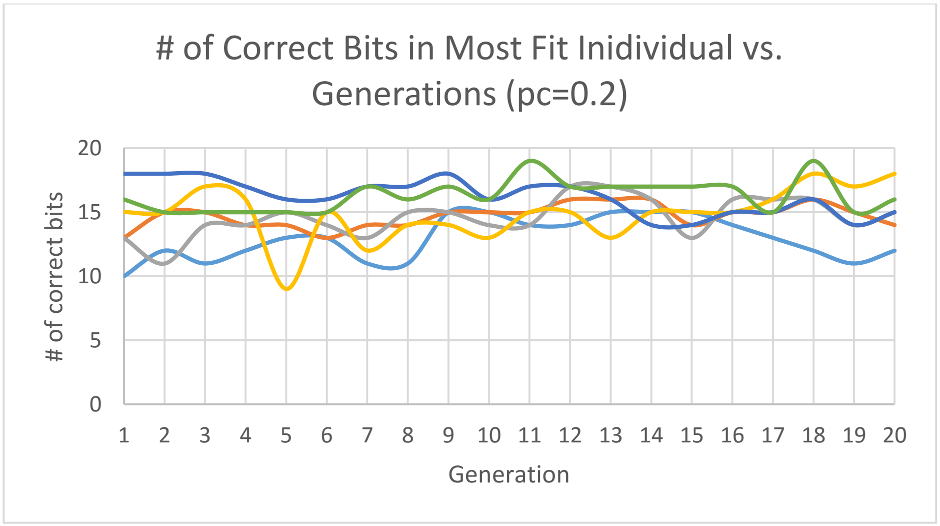

Best Fitness vs. Generations (pc=0.4)
Best Fitness
Generation
0
0.2
0.4
0.6
0.8
1
1.2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

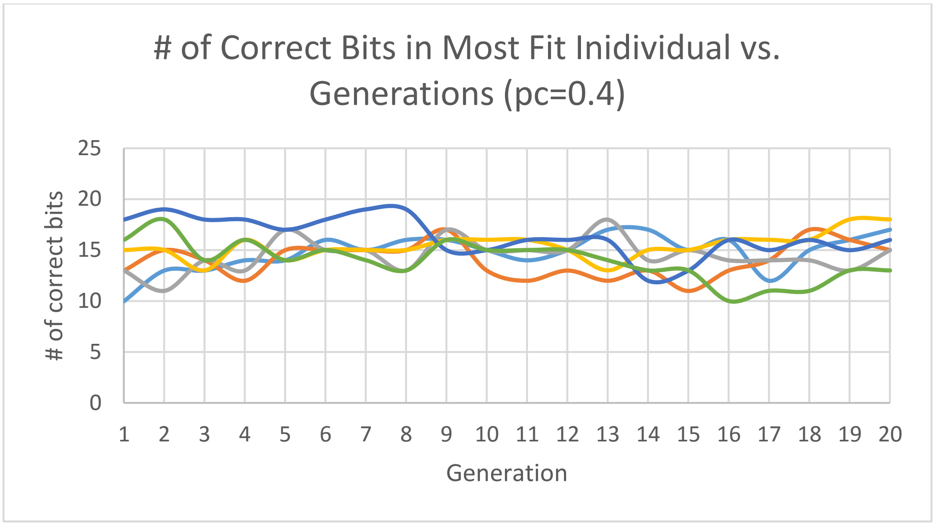

Best Fitness vs. Generations (pc=0)
Best Fitness
Generation
0
0.2
0.4
0.6
0.8
1
1.2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20



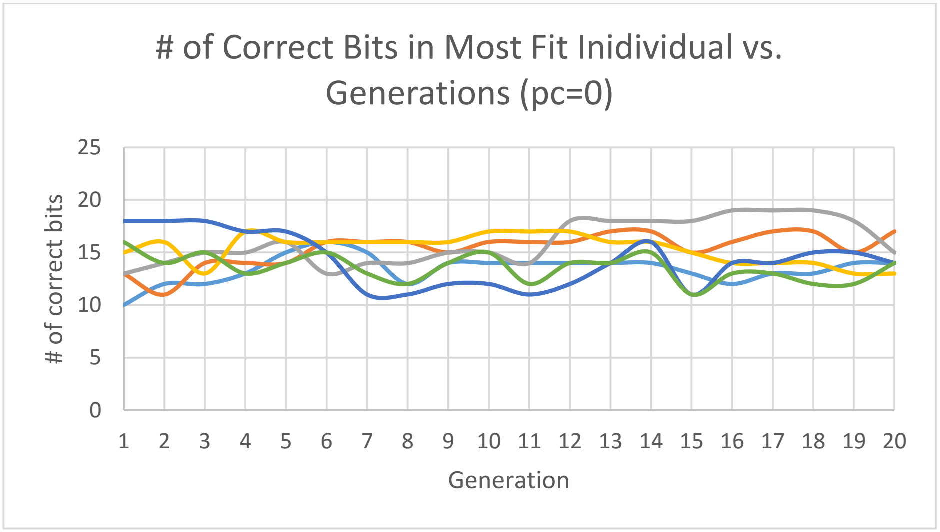


Figures 8: best fitness vs. generations with different crossover probabilities

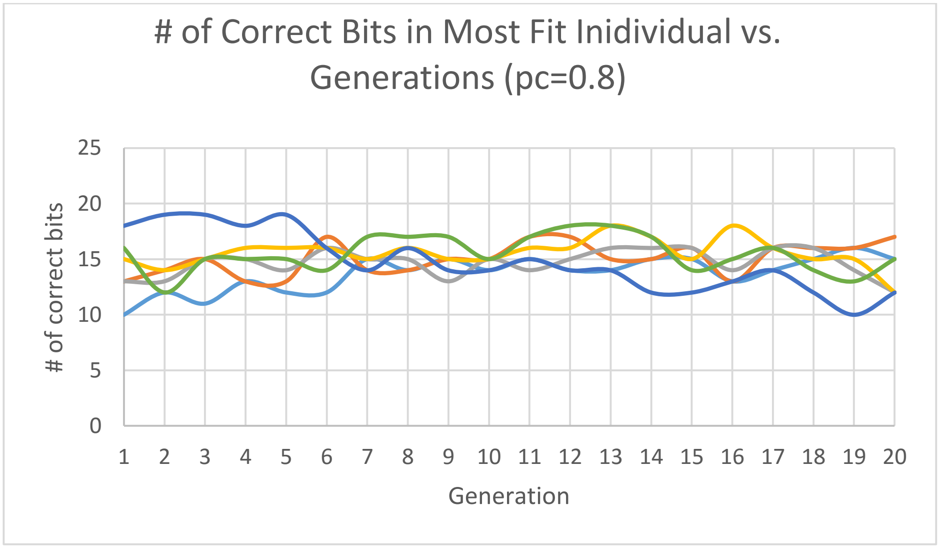




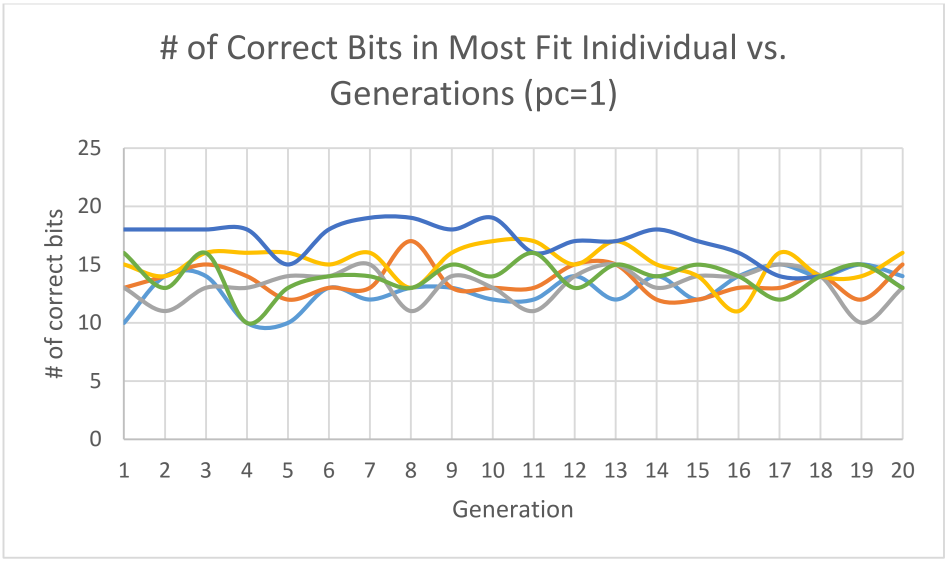
# of Correct Bits in Most Fit Inidividual vs. Generations (pc=0.4)
# of correct bits
Generation



# of Correct Bits in Most Fit Inidividual vs. Generations (pc=0)
# of correct bits
Generation

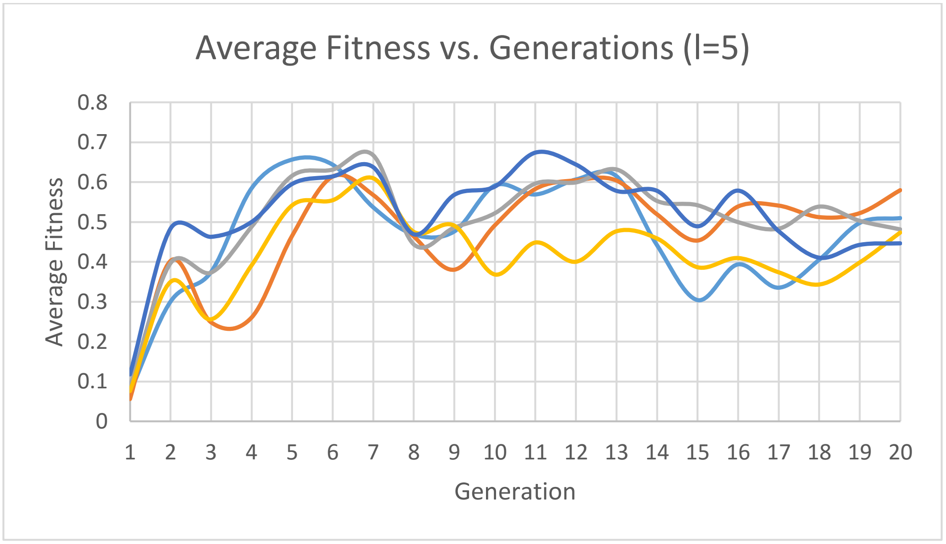


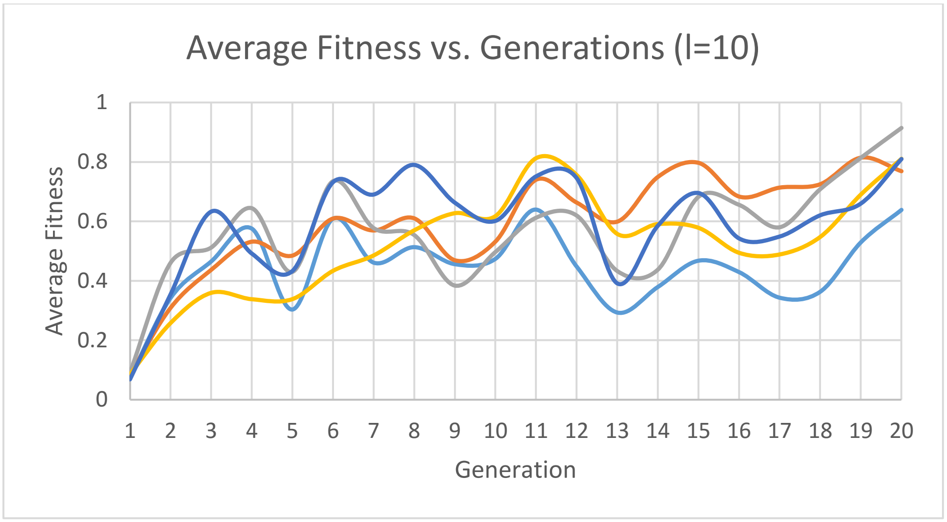
# of Correct Bits in Most Fit Inidividual vs. Generations (pc=0.8)
# of correct bits
Generation

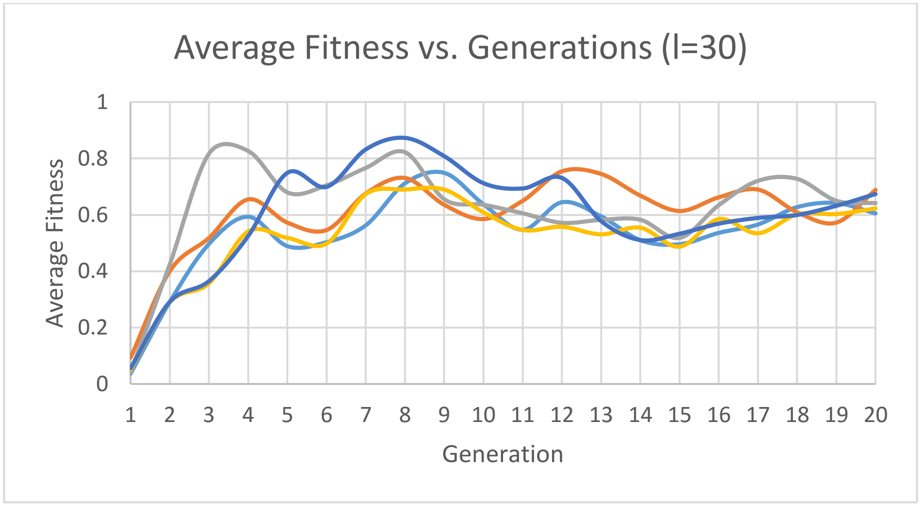


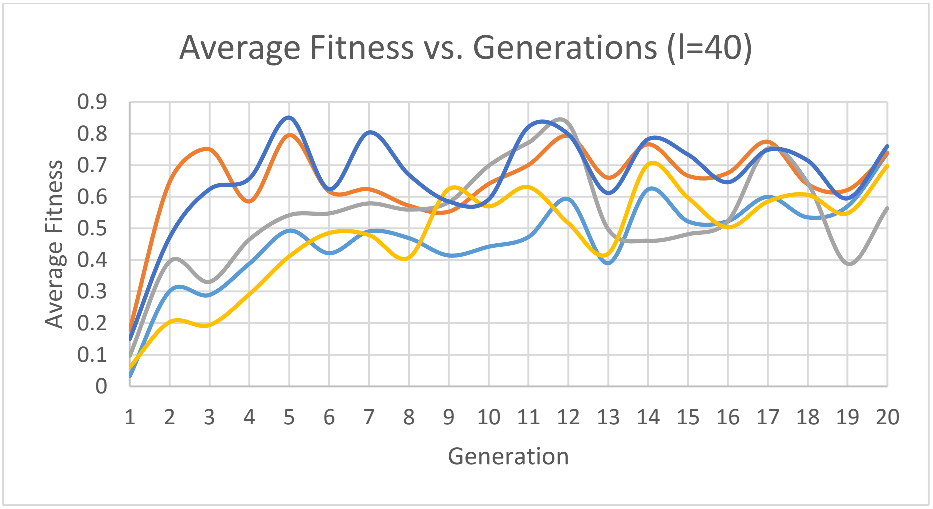
Figures 9: number of correct bits in most fit vs. generations with different crossover probabilities

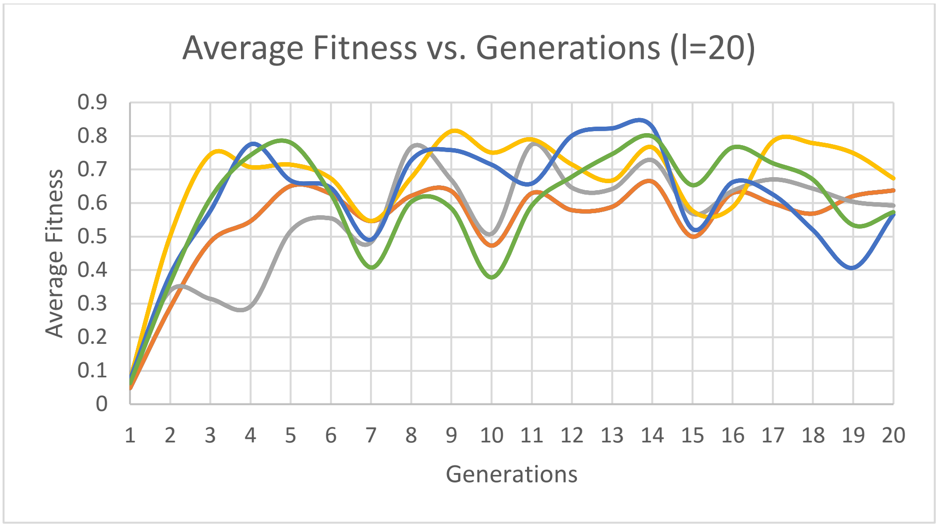
Figures 10, **11**, and **12** are results of varying the length of genetic string (l=5, 10, 20, 30, and 40) and are presented in graphs of the average fitness, best fitness and number of correct bits in the fittest individual with respect to time/generation.



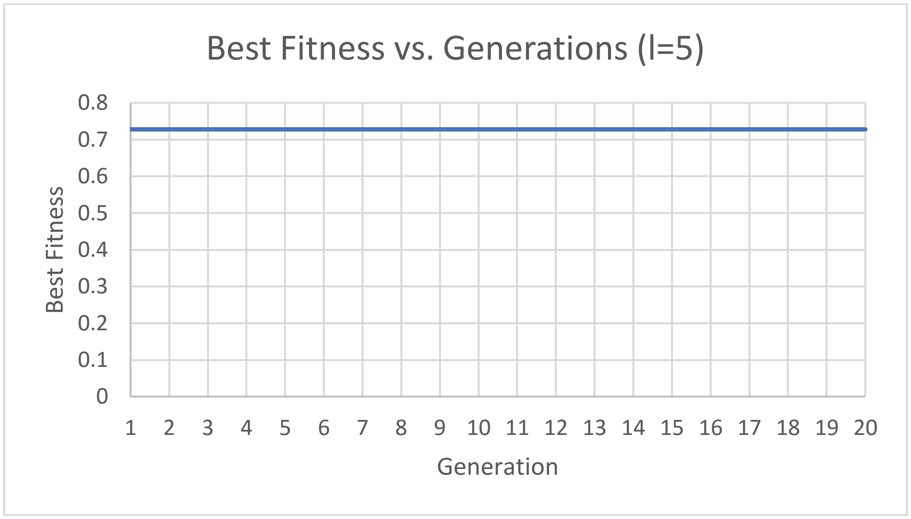

Average Fitness vs. Generations (l=10)
Average Fitness
Generation
0
0.2
0.4
0.6
0.8
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

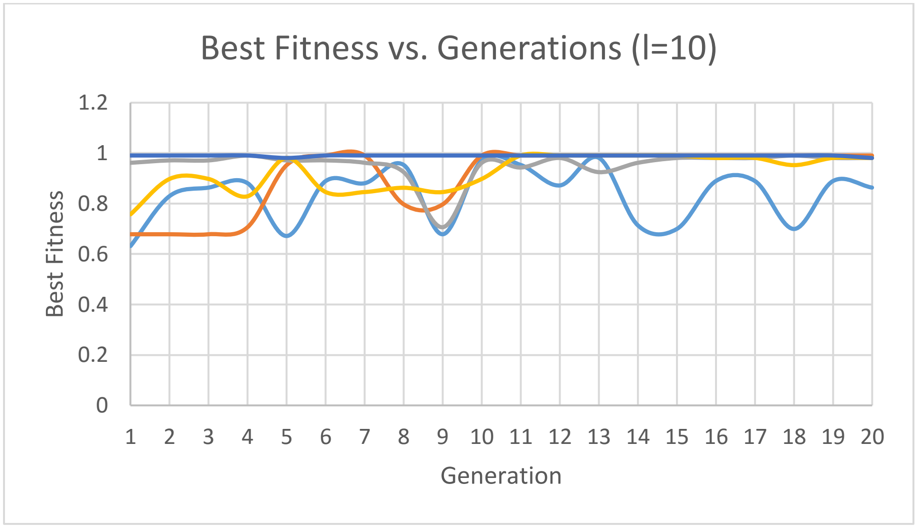

Average Fitness vs. Generations (l=30)
Average Fitness
Generation
0
0.2
0.4
0.6
0.8
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

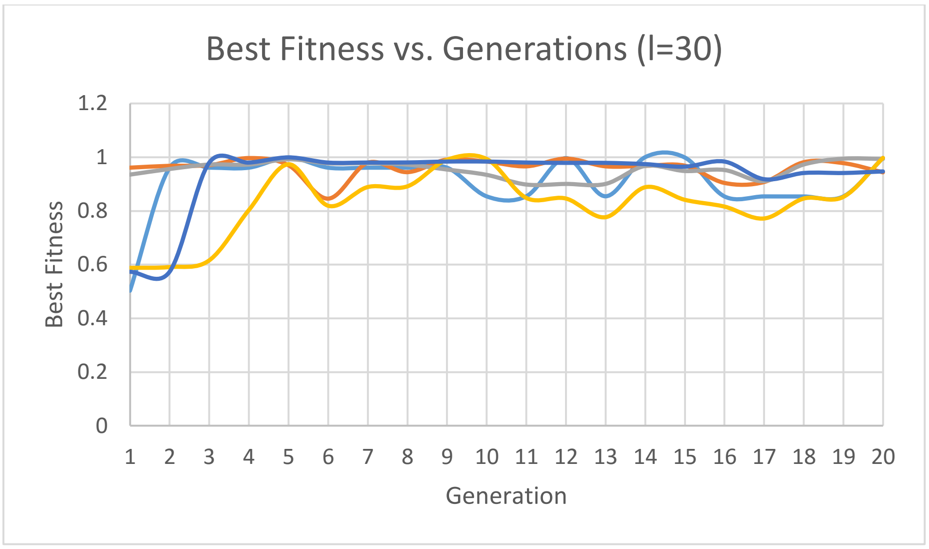

Average Fitness vs. Generations (l=40)
Average Fitness
Generation
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

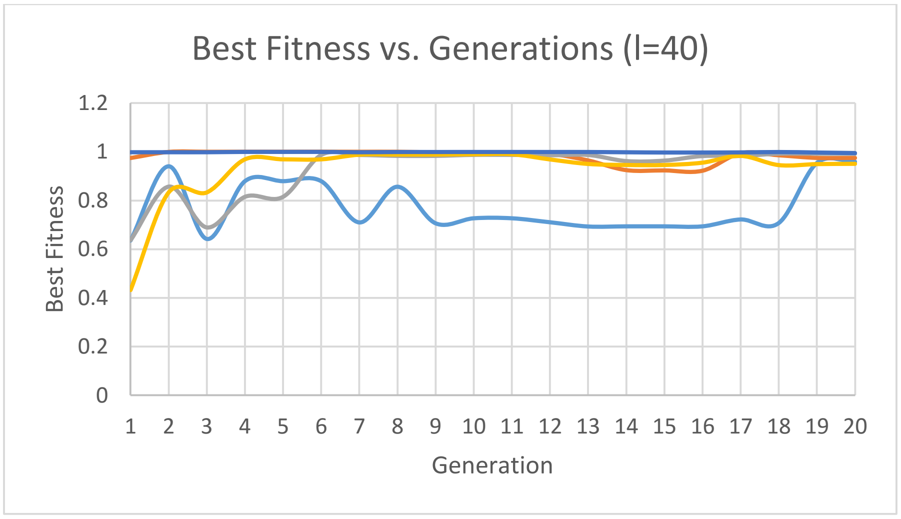


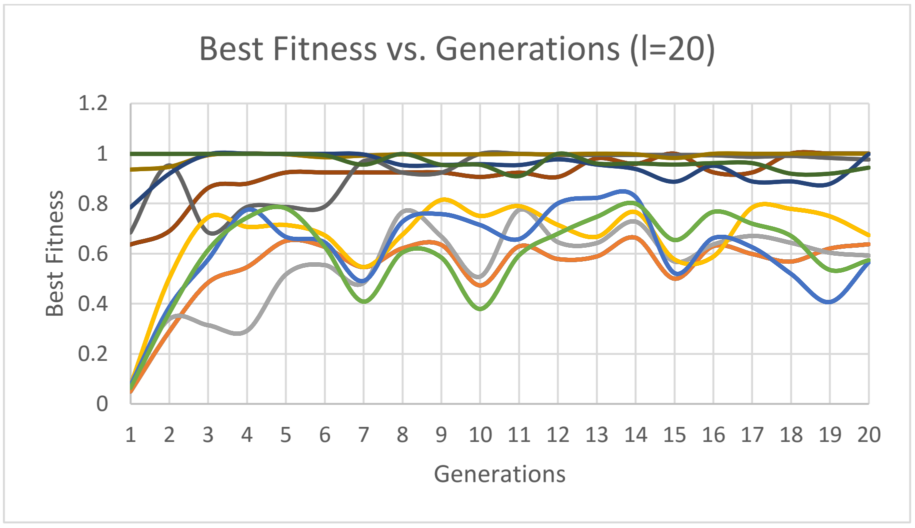
Figures 10: average fitness vs generations with different genetic string length



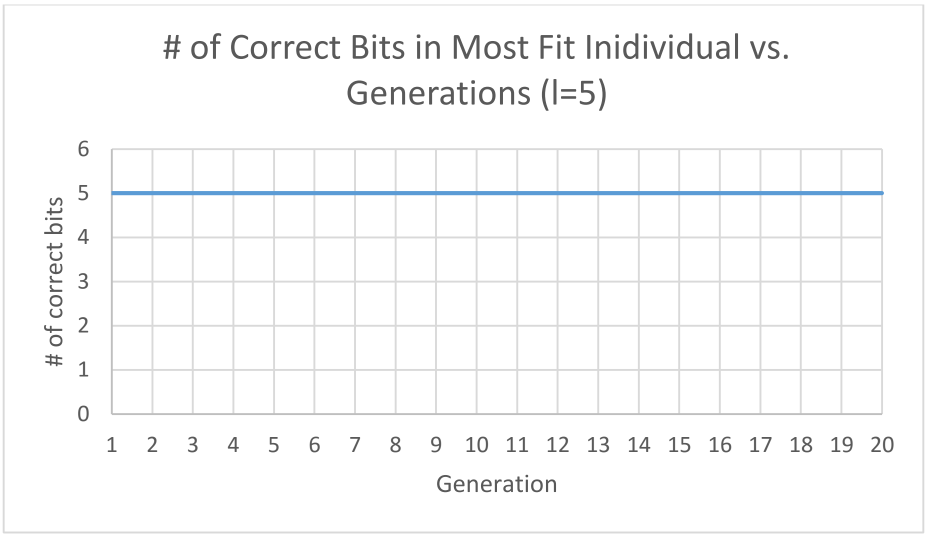




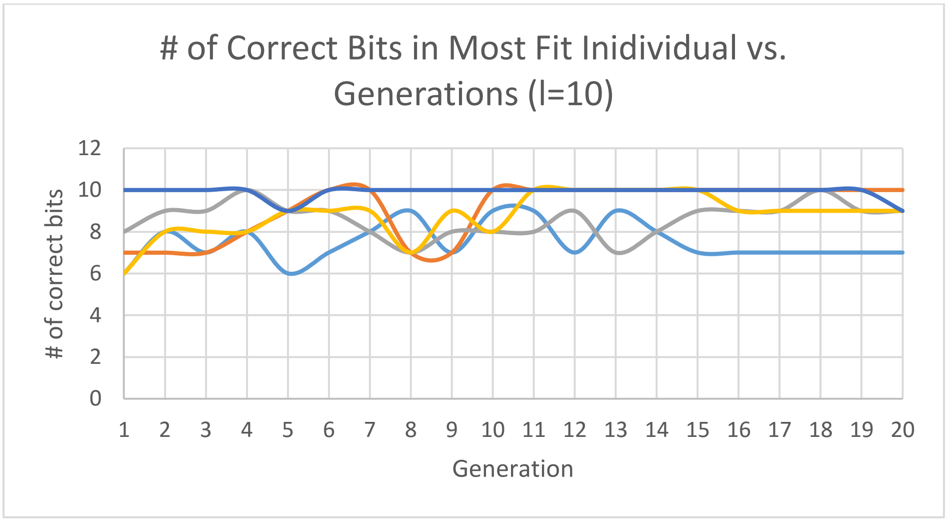




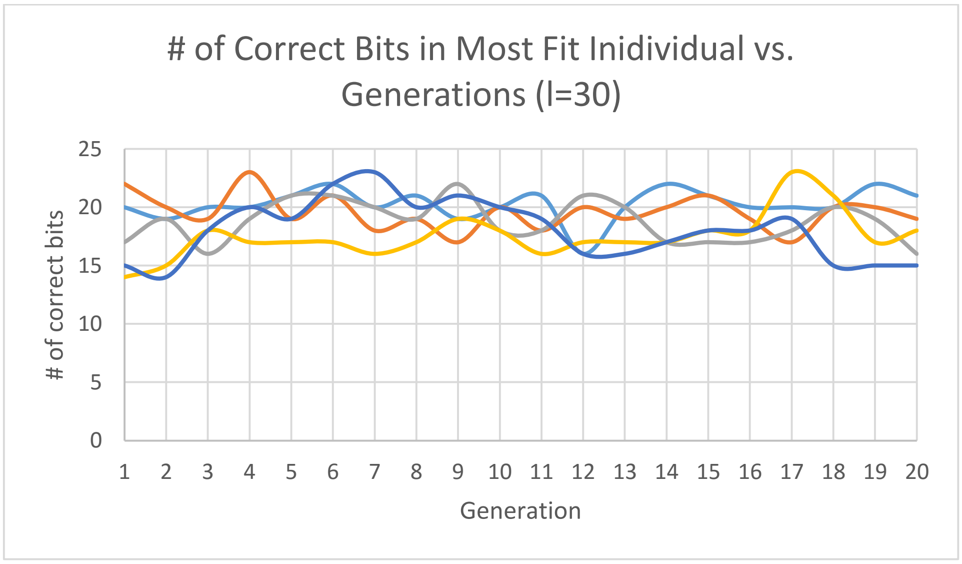
Figures 11: best fitness vs. generations with different genetic string length



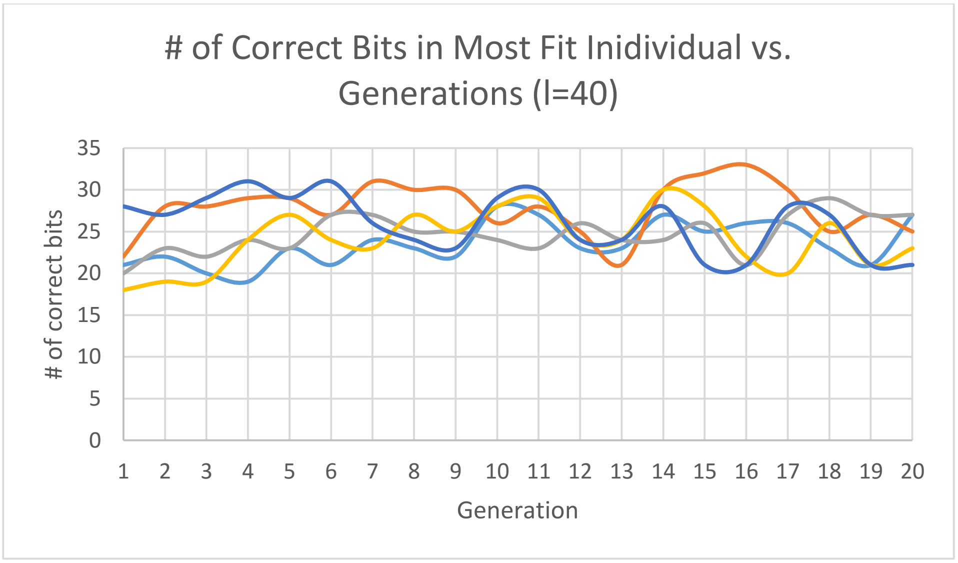
# of Correct Bits in Most Fit Inidividual vs. Generations (l=5)
# of correct bits
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Generation

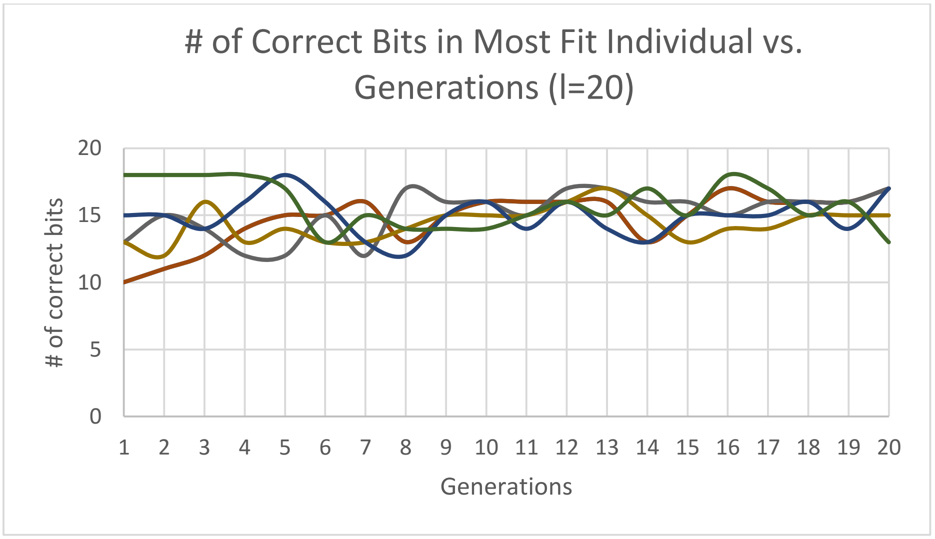


# of Correct Bits in Most Fit Inidividual vs. Generations (l=10)
# of correct bits
0
2
4
6
8
10
12
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Generation



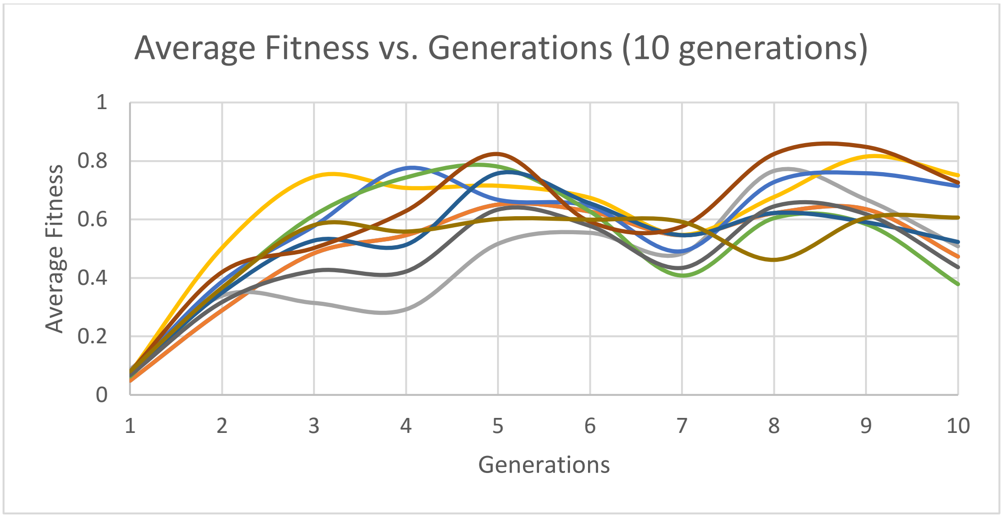
# of Correct Bits in Most Fit Inidividual vs. Generations (l=30)
# of correct bits
0
5
10
15
20
25
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Generation

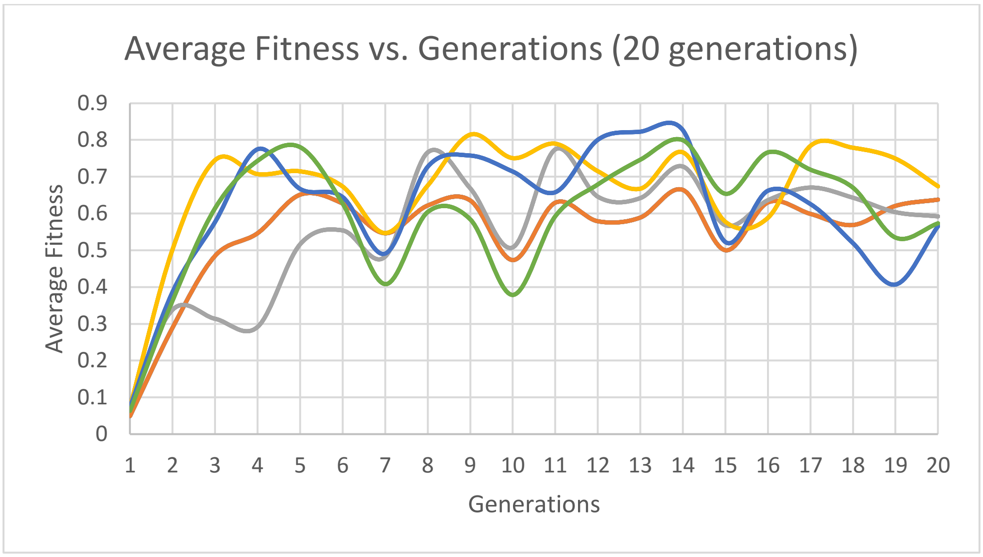


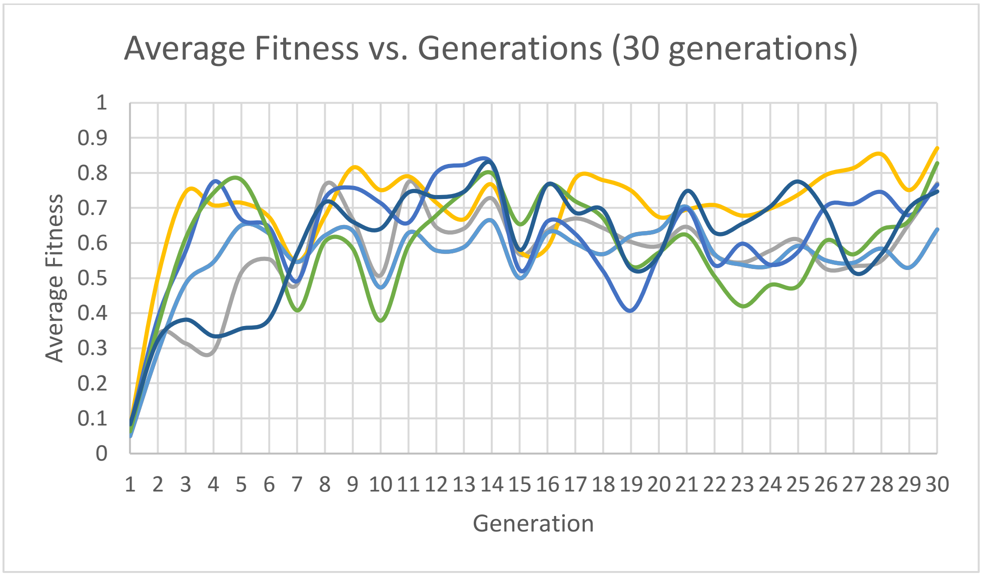


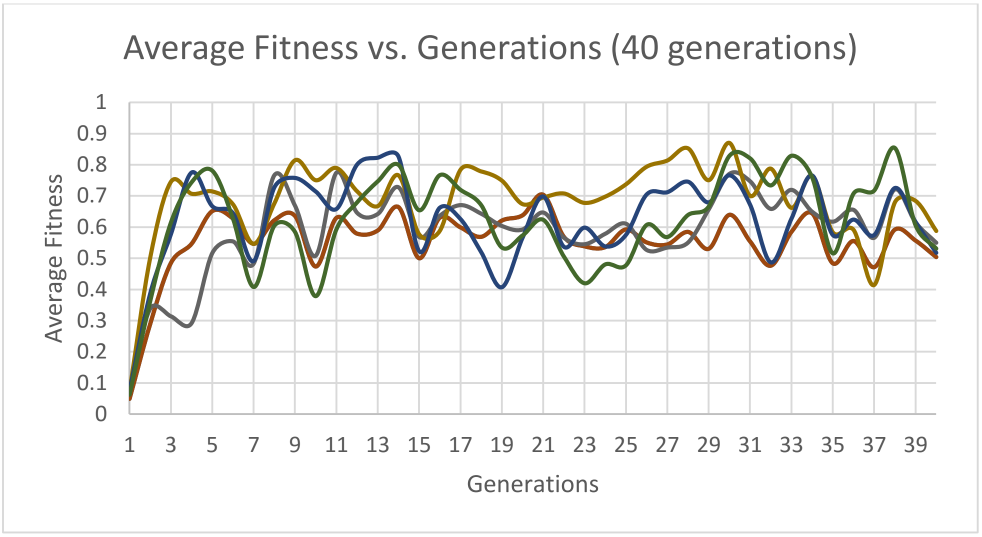
Figures 12: number of correct bits in most fit vs. generations with different genetic string length

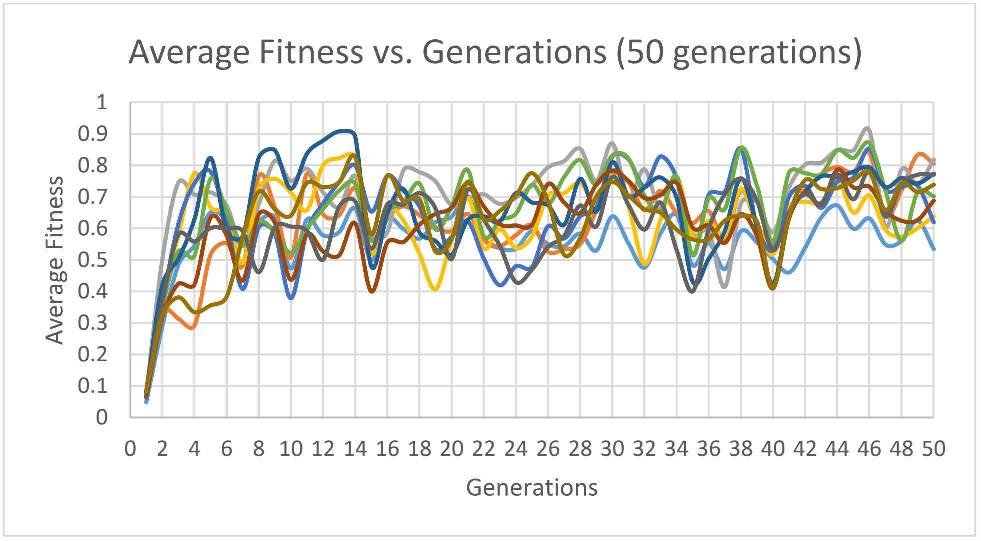
Figures 13, **14**, and **15** are results of the varying the number of generations (G=10, 20, 30, 40, and 50) and represent average fitness, best fitness, and the number of correct bits in the fittest individual in the population with respect to time, respectively.


Average Fitness vs. Generations (10 generations)
Average Fitness
Generations
0
0.2
0.4
0.6
0.8
1
1
2
3
4
5
6
7
8
9
10

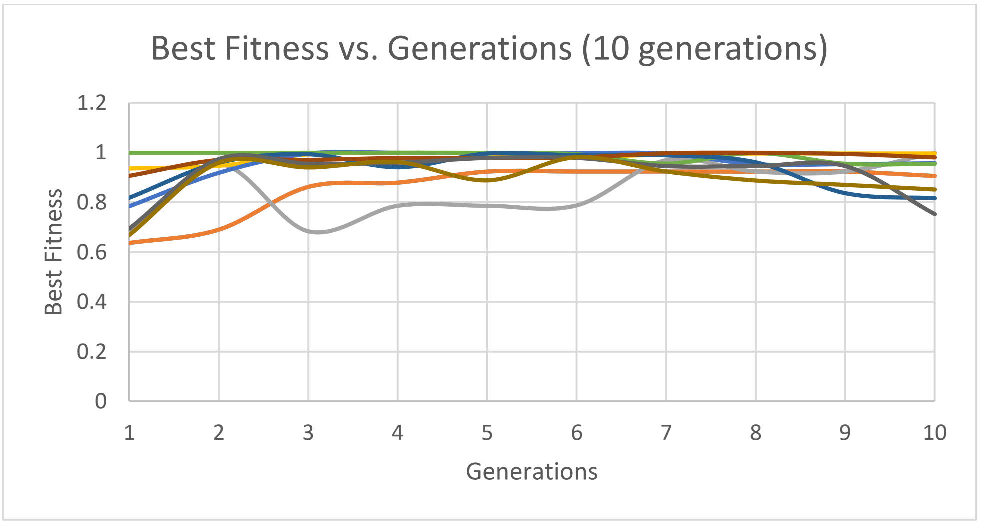

Average Fitness vs. Generations (20 generations)
Average Fitness
Generations
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

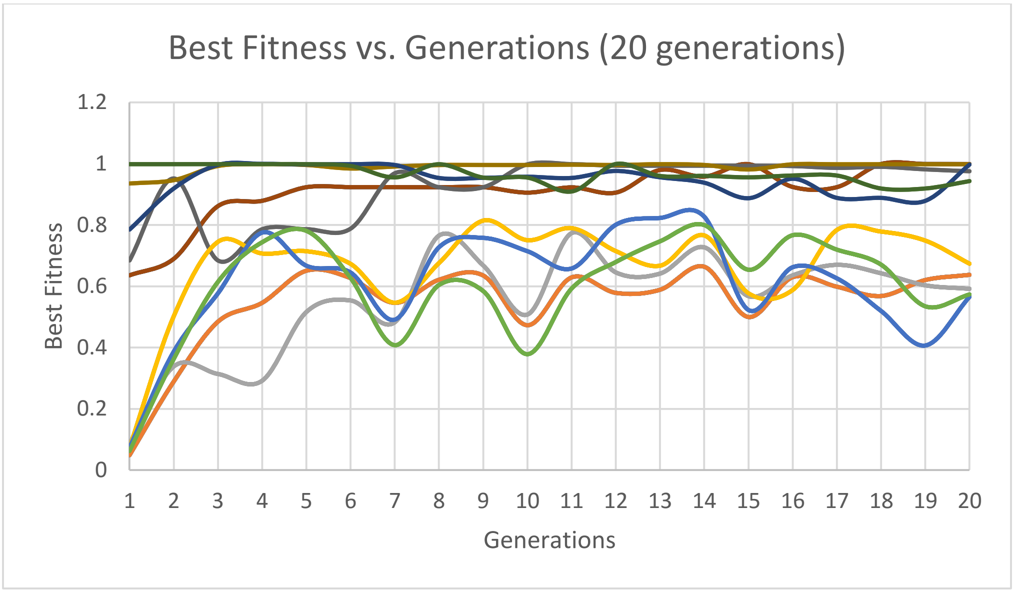

Average Fitness vs. Generations (30 generations)
Average Fitness
Generation
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

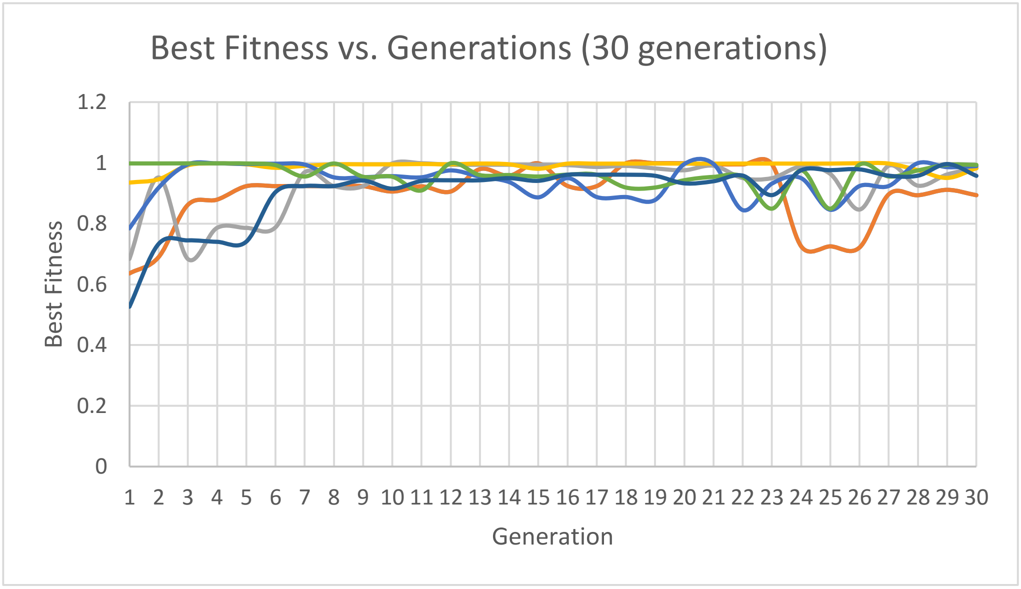


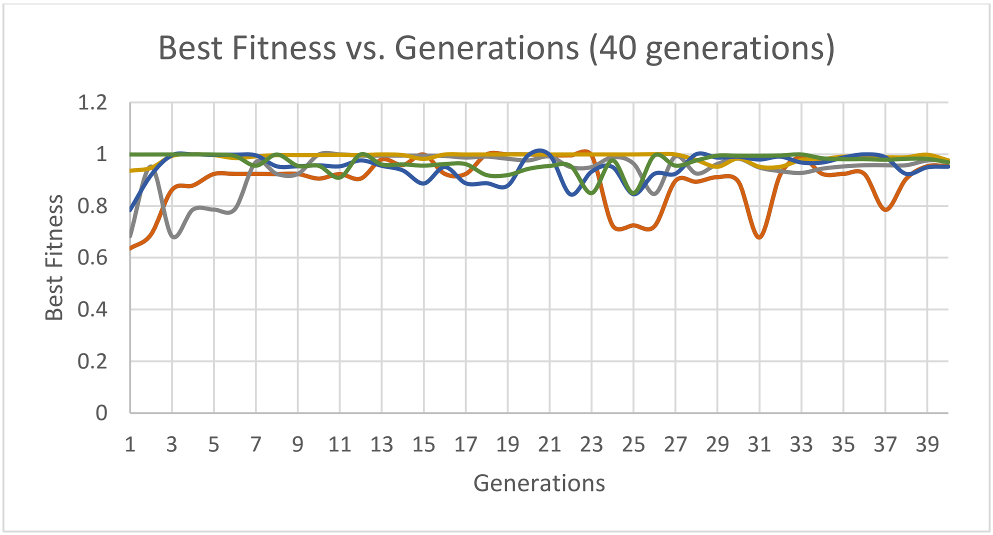


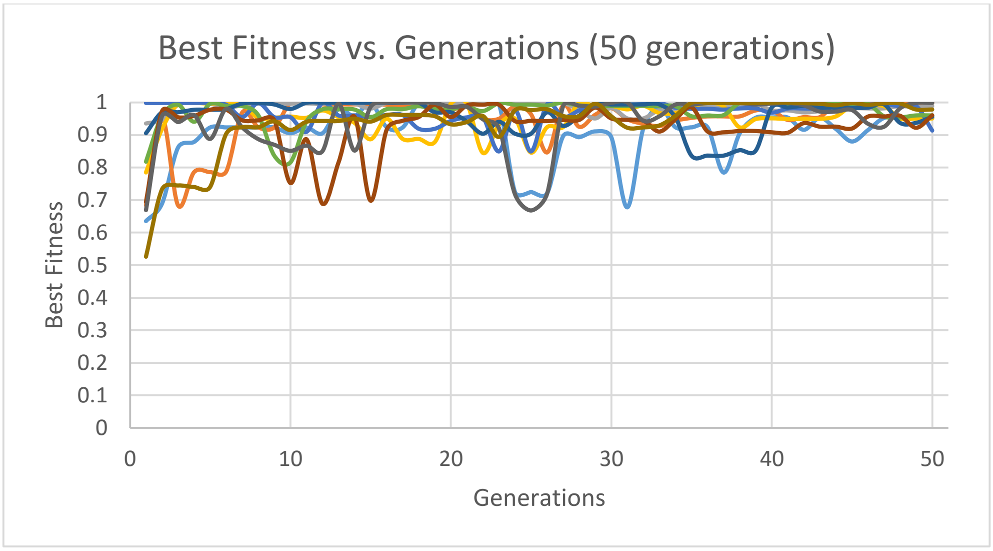
Figures 13: average fitness vs. generations with different generation numbers



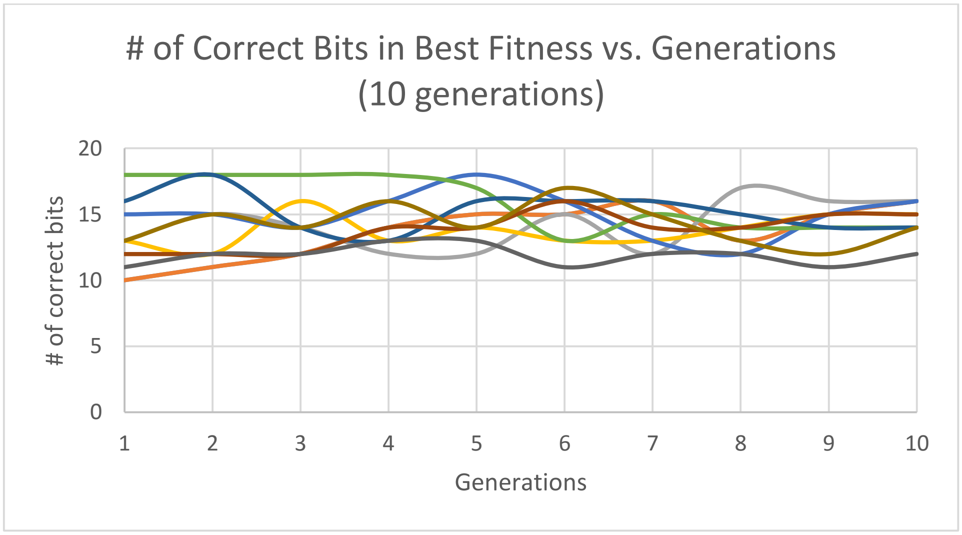

Best Fitness vs. Generations (20 generations)
1.2
1
0.8
0.6
0.4
0.2
0
Best Fitness
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Generations

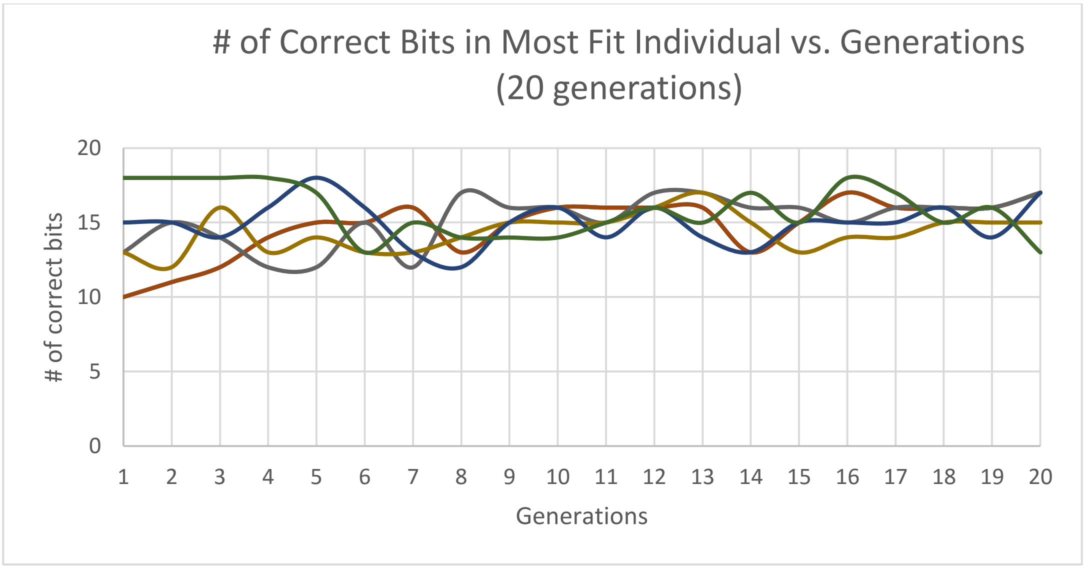

Best Fitness vs. Generations (30 generations)
1.2
1
0.8
0.6
0.4
0.2
0
Best Fitness
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Generation

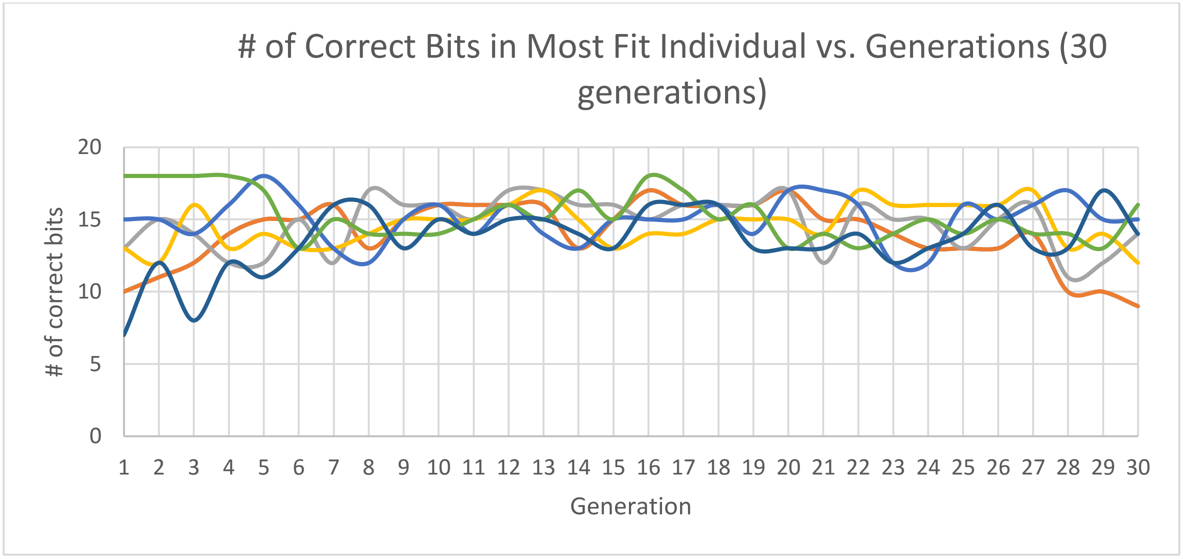

Best Fitness vs. Generations (40 generations)
1.2
1
0.8
0.6
0.4
0.2
0
Best Fitness
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
Generations

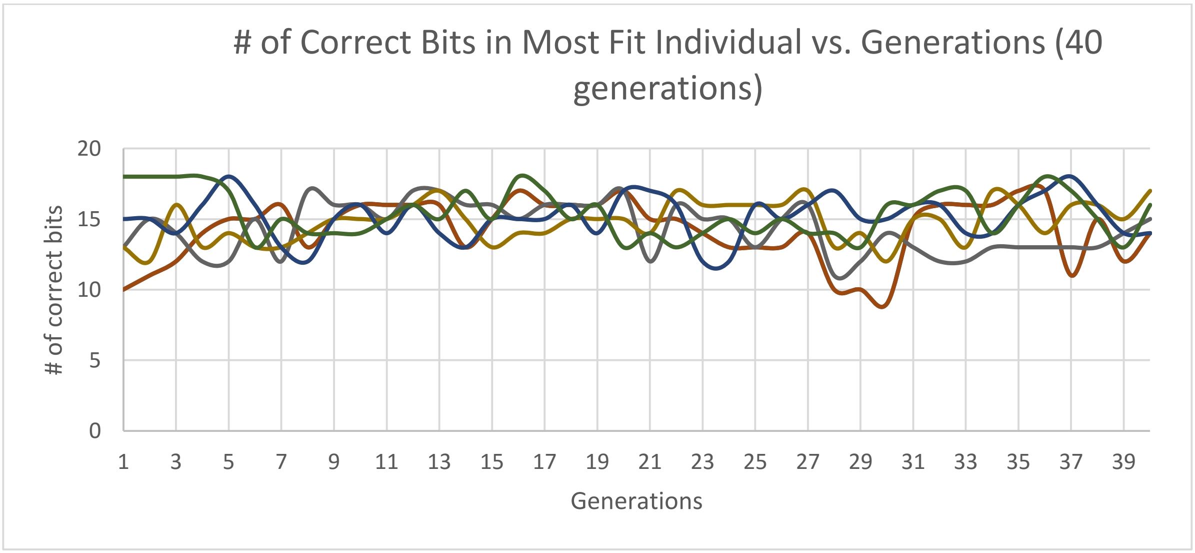


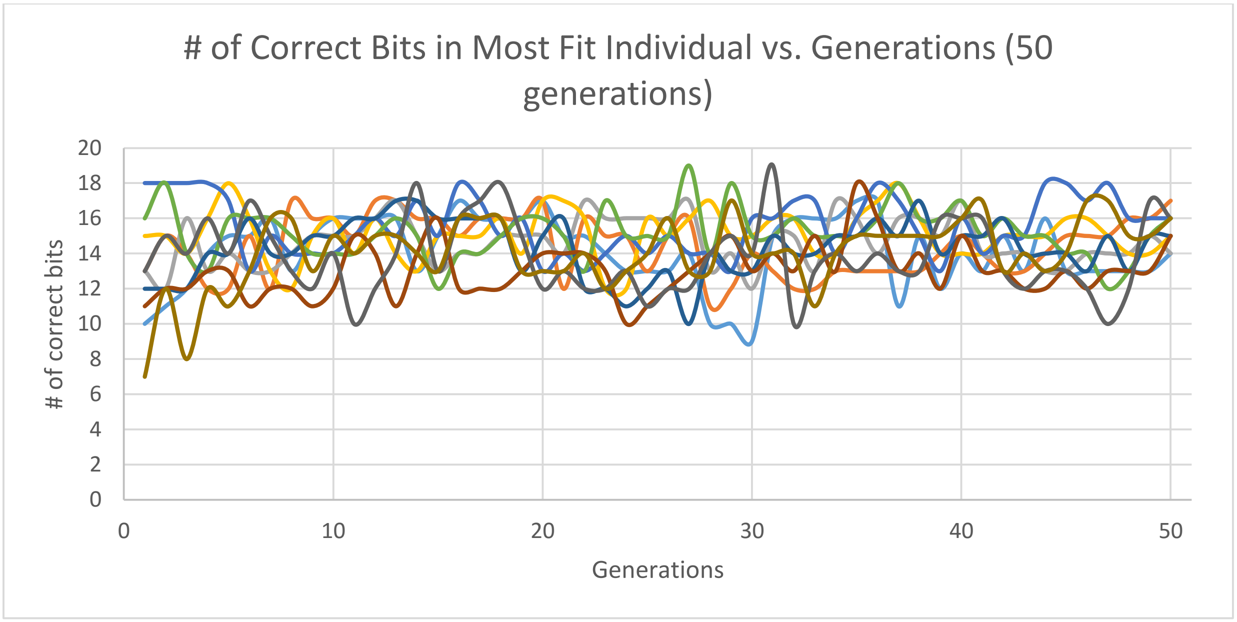
Figures 14: best fitness vs. generations with different generation numbers




# of Correct Bits in Most Fit Individual vs. Generations (20 generations)
# of correct bits
Generations
0
5
10
15
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


# of Correct Bits in Most Fit Individual vs. Generations (30 generations)
# of correct bits
Generation
0
5
10
15
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30


# of Correct Bits in Most Fit Individual vs. Generations (40 generations)
# of correct bits
Generations
0
5
10
15
20
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39



Figures 15: number of correct bits in most fit individual vs generations with different generation numbers

Analysis
Based on the data collected, the population of size 5 generated the lowest average fitness through the 20 generations with average fitness mostly fluctuating between 0 and 0.5. Peak average fitness is reached at the end of 20th generation at around 0.55. When population size is increased to 10, peak average fitness is reached to be ~0.8 and occurred approximately three times at generations 5, 10 and 14. Increasing population to 20, 40, and 80 increased peak average fitness to ~0.9 and the number of generations that had fitness values close to the peak average fitness (**Figures 1**). In other words, the average fitness increased as a whole when population size is increased and became more stabilized (less fluctuations and dips in average fitness) over generations. Similarly, in **Figures 2**, best fitness values fluctuated wildly between 0 and 1 when population is just 5, and the fluctuation decreased and became more clustered at 1 when population is increased. Especially at population=80, the best fitness stabilized to 1 at ~2-3 generations. The number of correct bits ranges from 10 to ~20 for populations with size 10 and on, but at size of 5, number of correct bits ranges from ~6 to ~17 (**Figures 3**). As suggested by the three types of graphs, fitness increases when population is increased and converges to near around 1. When population is increased, the fitness also seem to stabilize more quickly to the neighborhood of one.
Based on **Figures 4**, **5**, and **6** above, when mutation probability is set to 0, the population's average quickly converges to near 1 (ranging from 0.6 to 1) at around generation 4 (**Figures 4**). When mutation probability is increased, so did the number of dips and rises and the peak average fitness decreased. When pm=0.03, the average fitness roughly fluctuated around 0.6 to 0.9; at pm=0.1, the average fitness fluctuated around 0.25 to 0.6; at pm=0.5, the average fitness fluctuated around 0.05 to 0.15. And when pm is set to 1, the average fitness mostly fluctuates around 0. The number of correct bits and best fitness also saw more fluctuations when mutation probability is increased (and best fitness is small when pm=1, see **Figures 6**).

Figures 7, **8**, and **9** include data collected by varying crossover probability from 0 to 0.2 to 0.4 to 0.8 to 1.0. When pc=0, the average fitness first reaches its peak (~0.9) at generation 5 and then fluctuated from ~0.5 to 0.9. When crossover probability is increased, the number of generations that took for population to reach first fitness peak stayed around 5 generations and so did the peak average fitness (~0.9) and the fluctuation range (0.4 to 0.9). The best fitness graphs (**Figures 8**) show that the best fitness takes longer to stabilize to 1 with increasing crossover probability.

Figures 10, **11**, and **12** include data collected by varying the genetic string length from 5 to 10 to 20 to 30 to 40. When genetic string length is 5, average fitness peaked at around 0.7 and fluctuated around 0.4 and 0.7. Increasing it to 10 increased the peak average fitness to around 0.8. And increasing the length to 20, 30, and 40 have increased the peak average fitness values for the populations to around 0.8-0.9 (**Figures 10**). Looking at the best fitness graphs (**Figures 11**) reveals that when l=5, peak fitness stayed constant at around 0.7 while for lengths 10 and greater, best fitness of 1 was reached in at least one of the 6 runs. The number of correct bits in most fit individual was no longer perfect (as opposed to 5/5 correct and 10/10 bits correct for l=5, 10, respectively) as the length increased to 20, 30, and 40. This is to be expected with longer string length.

Figures 13, **14**, and **15** include data collected by varying the number of generations from 10 to 20 to 30, 40, and 50. As generation number increased, the peak fitness did not seem to increase significantly as the peak average fitness is reached at around 5 generations for this population. Similarly, number of correct bits in the most fit individual fluctuated around 10 to 18 with all generation numbers. And best fitness vs. generations fluctuated around 1 for all generation numbers. Based on these data, it seems that increasing the number of generations past ~5 doesn't improve the population fitness.

In the best individual in the final few generations, the bits that are wrong (or 0's) are usually in the second half of the genetic string. An example is that in generation 19 of varying generation numbers, the most fit individual is 11111111111111000101 with a fitness value of 0.999437. As in generation 20 of when varying mutation probability (pm = 0.1), the most fit individual is 11111111011011011011 with a fitness value of 0.977897. This makes sense arithmetically as 1's in more significant digits have greater base 10 values, which would generate fitness closer to 1 based on the fitness function in **Equation (1)**.

Conclusion

Based on the data collected, the basic genetic algorithm generated results with average fitness that peaked around ~0.8 to 0.9 after about 5 generations. Running it past around 5 generations doesn't seem to improve fitness. The genetic string length that's the smallest in the cases tested (l=5) generated the lowest fitness. Increasing it past 20, however, doesn't seem to further improve fitness.

The fitness fluctuated based on the noise, or mutation probability. Mutation rate determines the probability that a chromosome will have its genes modified. As shown in the simulations, large mutation probabilities generated poor fitness values in the five cases run. However, setting mutation probability to 0 doesn't encourage genetic diversity as the population average fitness quickly converges to a constant value after a few generations. If the goal is to promote genetic diversity, a small mutation probability can be given.

The crossover rate is the probability that the parents' genes will be mixed to create offspring (genetic recombination). It is essentially the rate that individuals in the population will be selected to breed since otherwise, their offspring remains the same as themselves. In literatures, it generally suggests that a crossover rate of ~0.66 (between 0.5 and 0.9) is generally a good default value. In the cases run in this project, varying the crossover rate did not significantly improve the fitness of the population using the specific parameters listed in section "Method."

Some literature suggests that the optimal population size is linked to the genetic string length (l=30 should use N=30) and some suggests N=20-30 is a good default value. As shown in the data collected, a small population size of 5 generated poor fitness, but increasing my population N past 10 didn't increase the fitness. However, increasing population size past 10 did minimize the fluctuations in fitness so that it’s more stable. Possible next step is exploring if there’s a relationship between population size and genetic string length. The results in varying the genetic string length showed that the fitness is poor when l=5, and improved as l increased. However, the improvement tapered off when l=20 and on.