COSC 420: Biologically Inspired Computation

Project 4: Genetic Algorithms

Kelsey Kelley

9 April 2019

Introduction

Crossover, mutations, and selection, these biological processes can be considered the backbone of survival in any society, they are also the most important aspects of a genetic algorithm. The selection process depends on how it is desired to be implemented, ideally speaking to make the population more ‘fit’ the individuals that should be selected should be the individuals with the highest fitness. This will create a bias within the population and is a poor way to do a simulation. So, the individuals to be selected are randomly chosen. Biologically speaking, crossover occurs during meiosis, when haploid gametes are formed, this is the process where the genes from the two parents are randomly mixed. In effect this is the ‘mating’ aspect of the population. Using crossover creates variation in the gametes. In the genetic algorithm, it keeps the simulation varied so the results are not all the same. The mutation process has several different variants in a biologic sense, these are; duplication, deletion, translocation, and inversion. For the purpose of this project, inversion was used. A genetic mutation is an altering of the DNA in a gene. Inversion is the process where bits are flipped. Normally, in a genetic algorithm, each individual has a low probability of mutating. Using these three processes a successful genetic algorithm could be simulated for a population of size N.

Theory/Methods

To accomplish the simulation of a genetic algorithm, first values had to be set. The values needed were; number of genes in the genetic string, population size, mutation probability, crossover probability, seed for random number generator, and number of generations. To begin a random population of size N with string length l had to be created. Then the generational steps had to occur. The fitness was then calculated for each individual in the population using the equation

$$F(s) = \left(\frac{x}{2^{l}}\right)^{10}$$

Figure 1. Fitness function

Where x is a running sum of the integer represented by that individuals bit string. While the fitness was calculated a running total was kept for later calculations. Then the fitness was normalized using the equation below

$$F_{norm} = \frac{F_s}{F_{total}}$$

Figure 2. normalized fitness function

While normalized fitness was calculated a running total for each value was also calculated. The next step was to perform the biological processes associated with the Genetic Algorithm, i.e. selection, crossover, and mutation. These next steps had to be performed N/2 times, or half the population amount of times. First selection had to be done to get the parents. To select the parents, two random numbers between zero and one had to be generated. Then using the normalized running totals calculated earlier, the range where each of the numbers fell within would result in the parent being assigned the upper bound. Then after it was made sure that the parents were two distinct numbers, the next process is crossover. To perform crossover a random number had to be generated to determine if crossover could happen. If crossover happened another random number was generated between zero and length to determine the crossover point, then the parents split up which genes went within each of the two children. Next it had to be determined if mutation would occur. Each child's bit string was passed through and a random number generated each time to determine if mutation would take place on that bit, if mutation was deemed to happen the bit was reversed. Once all the children were created the next generation is complete and ready to take over the population. Once it had been successfully swapped, parameters could be calculated for each generation. These values were average fitness, best fitness, and correct number of bits.

Results/Discussion

The results from the simulated experiments, with each of the five parameters being varied both lower and higher are shown below. The data received from these simulations is not what was expected. The main issue within the genetic algorithm simulation was in the selection stage for finding parents. The approach taken was to generate two random numbers and find where in the accumulated total they fell. The problem with this either comes from the random number generator not being varied enough to encompass many individuals, or within the crossover point of the selection process also not creating a good variation of random numbers. Even though the data was not exactly ideal, conclusions can still be made about these simulations. These are made after the graphs below.

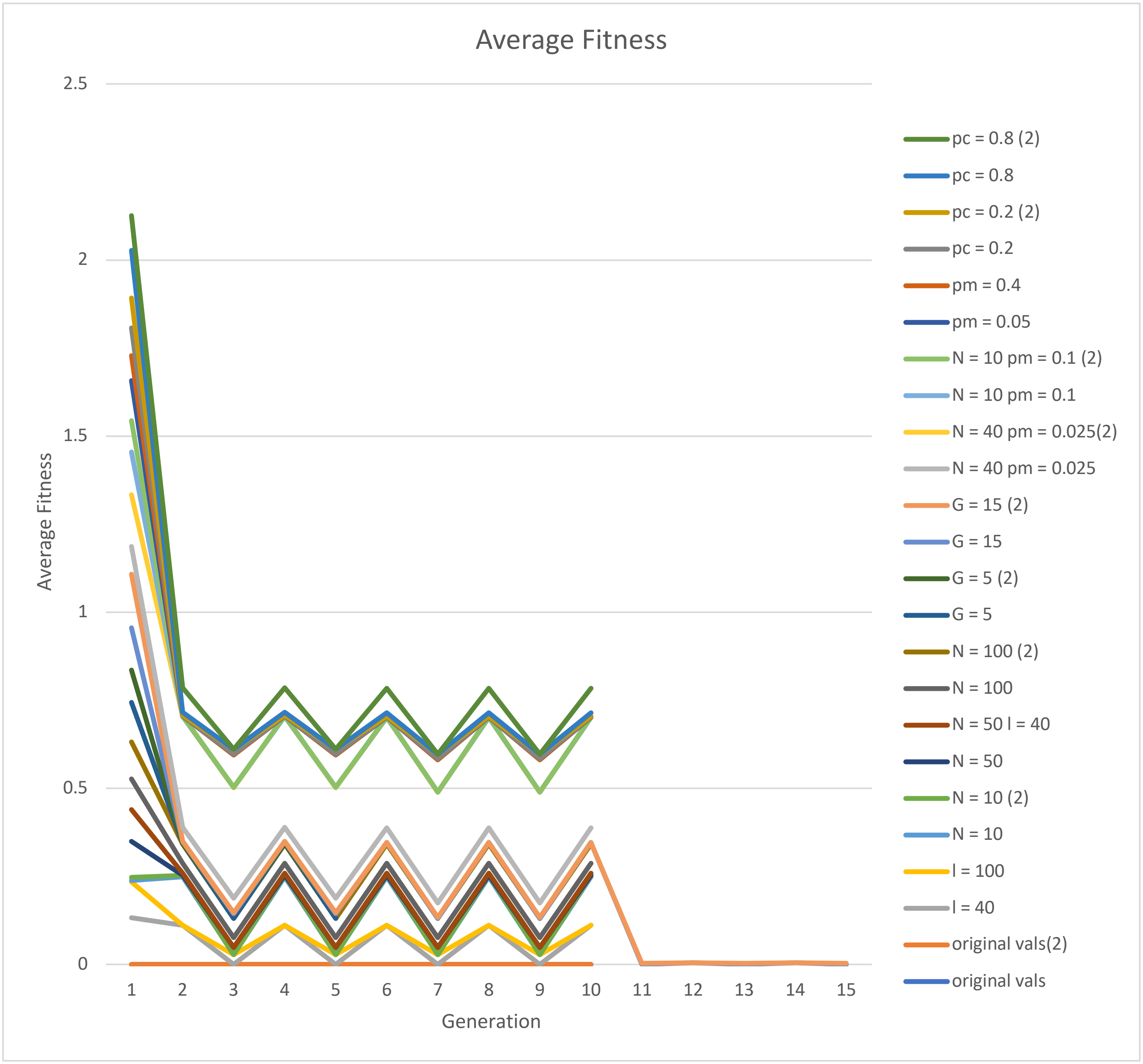

Figure 3. Graph of Average Fitness

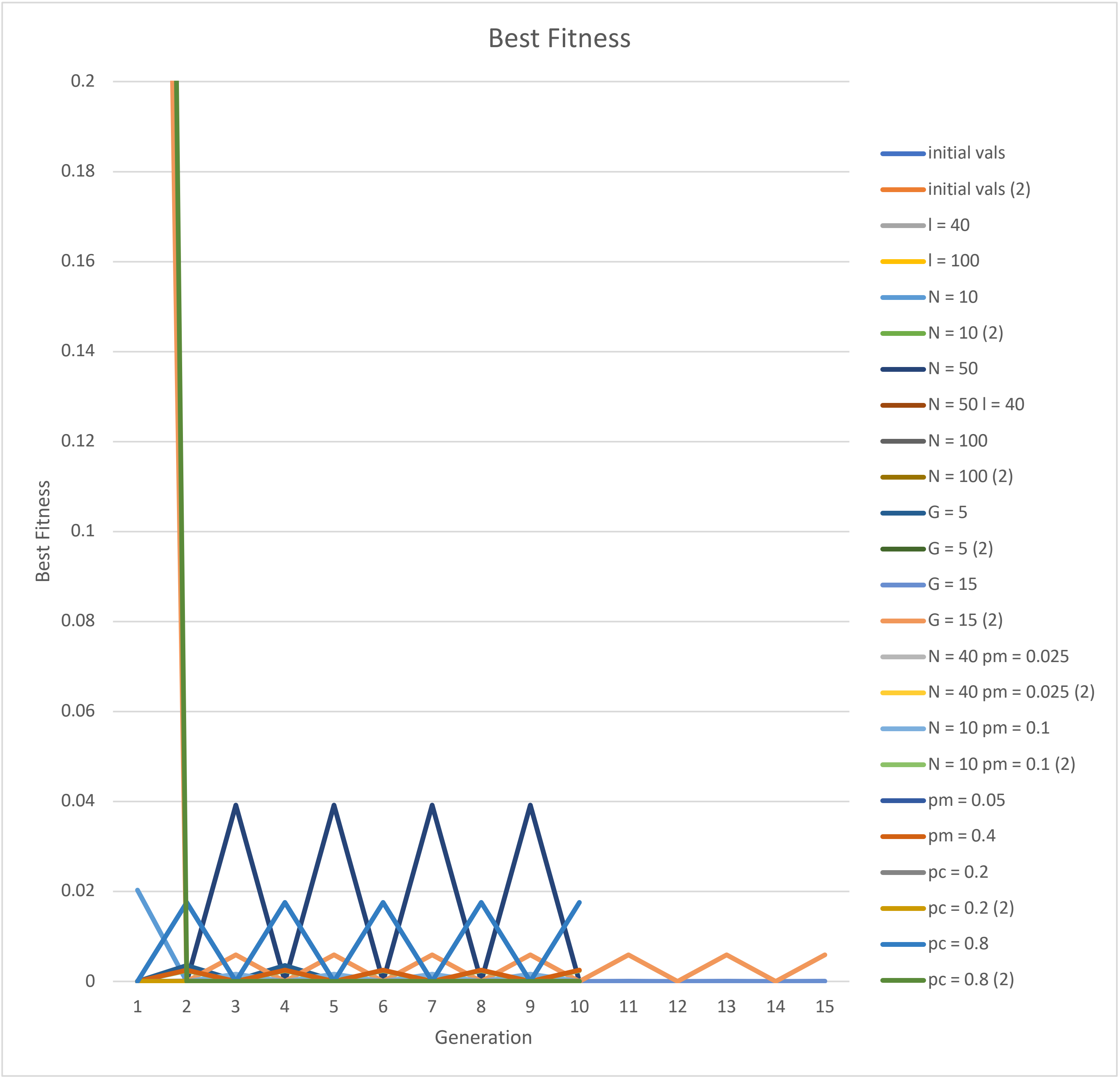

Figure 4. Graph of Best Fitness

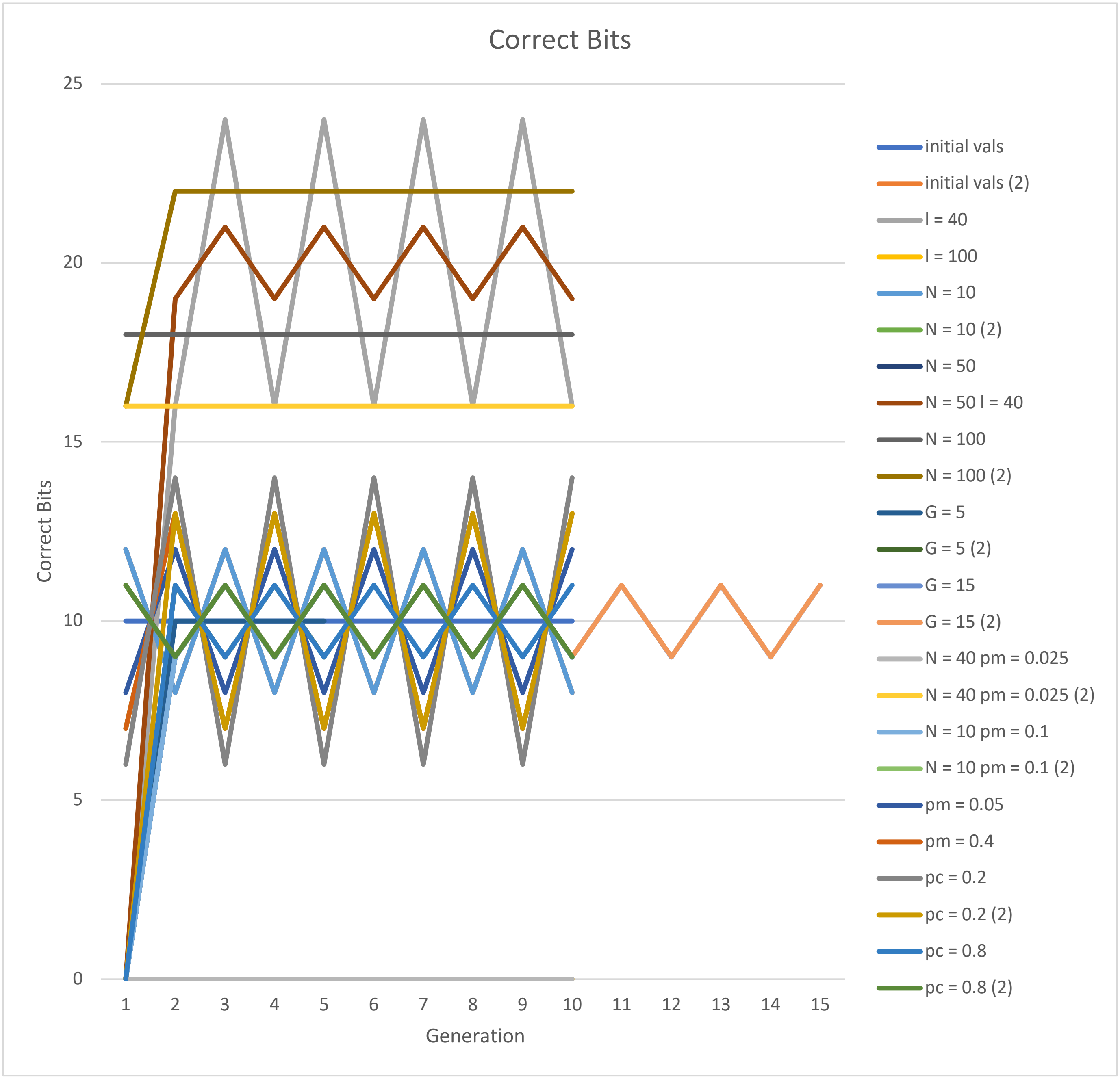

Figure 5. Graph of Number of Correct Bits

Looking at figure 3 whenever either the mutation probability is greater than 0.1 or crossover probability is greater than 0.7 the average fitness is higher than if these values were lower. This is probably due to the mutation and crossover happening more frequently and creating a more diverse population. When crossover population was made higher it also allowed the best fitness values to be greater than any other simulation as can be seen in figure 4. In figure 5 it can be seen that whenever the population size was made to be 40 or greater or if the length was made greater than 40, the correct bits increased. This makes sense because if length of genetic string is longer obviously there are more bits available that could be correct. When crossover probability was made larger the number of correct bits were either consistently the same or only varied by plus or minus two. When crossover probability was decreased it made the difference between the correct bits larger.

Conclusion

Creating the genetic algorithm was an interesting way to see how the biological processes of selection, crossover and mutation can affect a population. Overall the algorithm was not perfect, but it gave varying results that led to interesting points. The algorithms overall ability to find an optimum solution would need more debugging and work to be able to accomplish these tasks. It was interesting to be able to apply this algorithm with a binary genetic string and accomplish these simulations albeit in a not so ideal way.