Genetic Algorithms

Ronald Randolph
CS420: Biologically-Inspired Computation
April 9, 2019

Introduction

Genetic Algorithms (GAs) are adaptive, metaheuristic search algorithms that take inspiration from the biological process of natural selection and evolution. They belong to a larger group of algorithms, Evolutionary Algorithms. Genetic Algorithms are most popularly used in solutions to optimization and search problems.

The genetic algorithm was popularized by John Holland in 1960. At its conception, it was based on the concept of Darwin's theory of evolution. The process of genetic algorithms involves a population of individuals – or phenotypes. These solutions are then evolved toward better solutions through the process of selecting parents and iterating through generations.

Theory and Methodology

A genetic algorithm is composed of a population of *N* candidates that is usually comprised of randomly generated individuals with a random set of genes. These individuals are most commonly represented with separate arrays of bits. Each bit corresponds to the presence of a specific gene.

After the initialization of the starting population, it then undergoes a series of "evolutions". The population at each iteration – or evolution – is called a generation. For each generation, the fitness of each individual is calculated and recorded. The function for determining individual fitness is and objective function. However, for this experiment, the following fitness formula (**Figure 1**) will be used.

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

Figure 1: Real-Valued Fitness Function

The fitness formula in the figure above is a straightforward one. From a quick analysis, the genotype that would result in the optimum level of fitness would be one where the it is all 1's. While this seemingly makes for an easy problem, it creates a very narrow optimum. This can be clearly seen in the graph below (**Figure 2**).

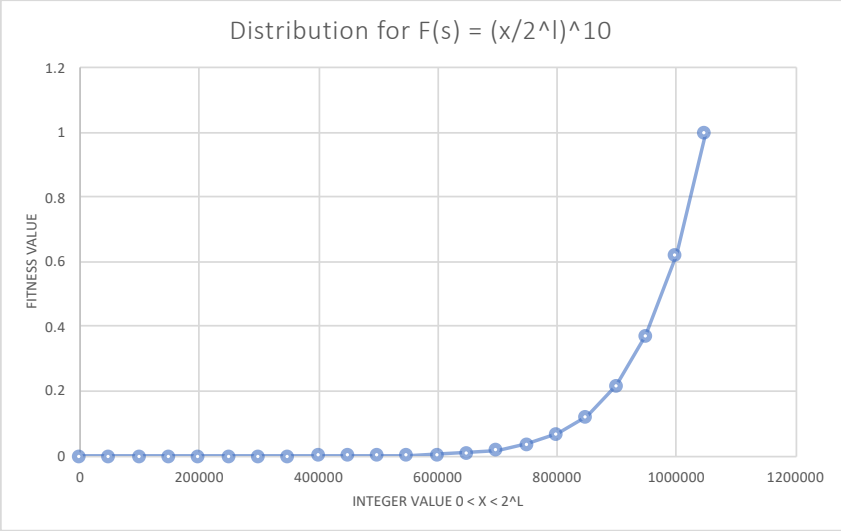


Figure 2: Distribution of the Fitness Function

After the fitness value has been calculated for each individual a new generation must be created. This is accomplished by selecting a pair of "parent" individuals from the current generation to breed. Using the determined values for the probabilities of mutation and crossover, a new child is formed from its parents' genotypes. This is repeated until an entirely new population of size *N* is generated. This process results in a new generation with completely different genotypes. The statistics of each new generation is recorded, and this process is repeated for a designated number of generations, *G*.

The included C++ program, *p4.cpp* takes the following arguments (**Figure 3**) for use in the process described above.

ℓ	Number of genes (bits) in the Genetic String
N	Size of the Population
G	Total number of Generations
P_m	Probability of Gene Mutation
P_c	Probability of Gene Crossover

Figure 3: Input Arguments

The program, *p4.cpp*, executes each of the described steps above. This process is iteratively repeated *G* times. For each value of *G*, generational statistics including the average fitness, most fit individual, and the average number or correct bits in the population are recorded. The main functions of the program operate as follows:

- **Initialize_Population** – creates *N* arrays of ℓ randomly initialized bits (genes).
- **Calculate_Fitness** – iterates through *N* individuals and calculates fitness values.
- **Find_Parents** – selects two distinct individuals to produce two offspring
- **Process_Offspring** – creates two child genotypes from given parents
- **Process_Stats** – Calculates generational statistics for each generation
- **Print_GenStats** – prints out generational statistics to an external *.csv* file.

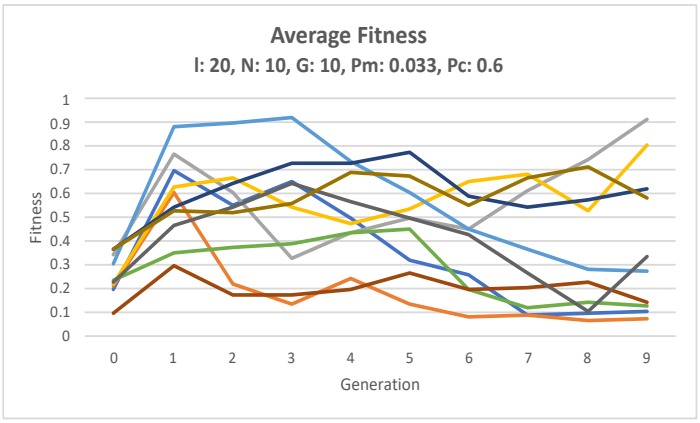
The program first creates *N* randomly initialized arrays of ℓ bits. After initialization, the program then calculates the corresponding integer value for each individual's genotype. The calculated integer value is then plugged into the aforementioned fitness function (**Figure 1**). The program calculates the fitness value and normalized fitness value whilst maintaining a running total for both.

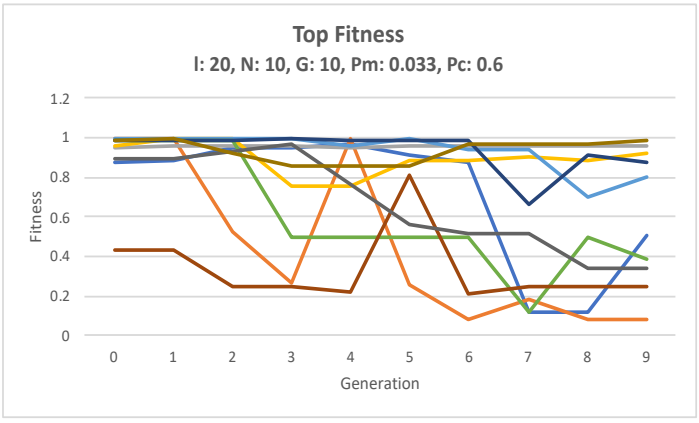
Using a random number between 0 and 1, the program selects two distinct individuals to be "parents". The program then performs any crossover or mutation as deemed statistically necessary. This results in two "child" individuals that are created from both of parent individuals. This process is repeated for until the new population reaches size *N* (2/*N* iterations).

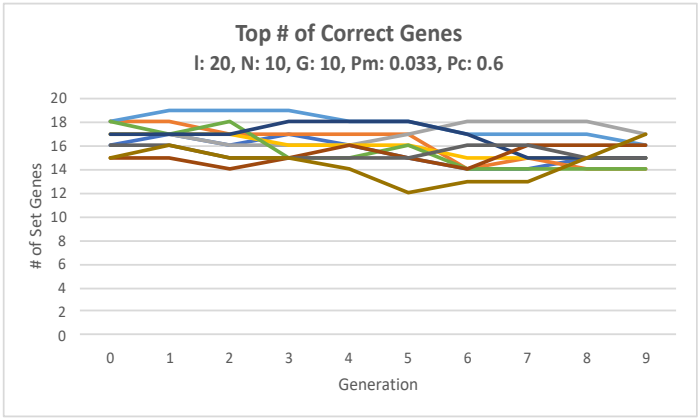
After the new population has been created, it overrides the previous parent population. At this point the program then records all requested generational statistics. This entire process is repeated for a total of G times. The data from each generation is culminated into arrays and printed to an external *.csv* file before exit.

Results and Findings

The data produced by the program, *p4.cpp*, can be found in the included file *data.cvs*. After a quick observation of the output data from the first run, there appears to be a distinct rise in population fitness around the fourth and fifth generations. The graphs below (**Figure** 4) show the recorded generational statistics for several runs with the initially suggested parameters (**ℓ =** 20, N = 10, G = 10, Pm = 0.033, Pc = 0.6).





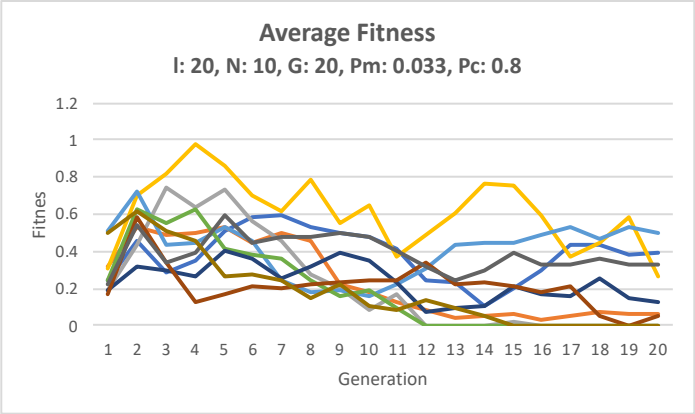


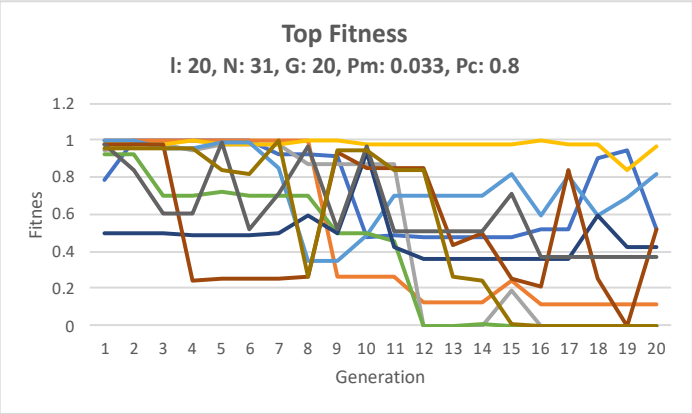
<u>Figure 4</u>: ℓ = 20, N = 10, G = 10, Pm = 0.033, Pc = 0.6

In this run the parameters remained at their suggested level and the population went through ten generations. From the graphs above there appears to be a distinct increase in fitness immediately after a the first few generations. However, that increase steadily declines with the following iterations. From the graph of the number of correct genes, there was an overall steady amount of set genes throughout the generations.

Figure 5: ℓ = 10, N = 30, G = 10, Pm = 0.033, Pc = 0.6

In **Figure 5** above, the parameters were tweaked to explore if there were any direct effects of larger populations with smaller genotypes. From initialization, the average fitness of the population was lower but quickly jumped after the first iteration of new offspring. Interestingly, the top fitness levels started at the max level and slowly started to decrease after four or five generations passed.





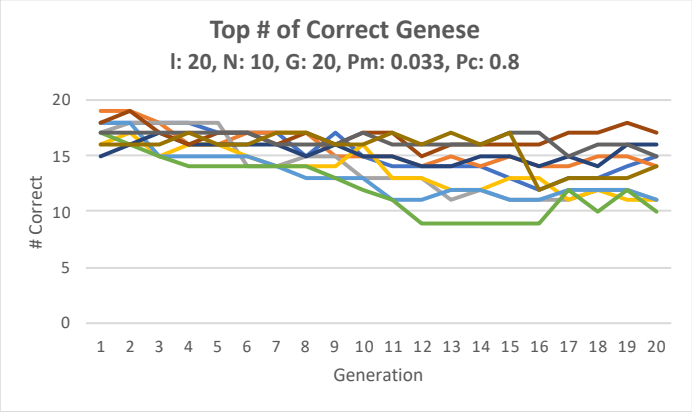


Figure 6: ℓ **= 20, N = 10, G = 20, Pm = 0.033, Pc = 0.8**

For the run detailed above in **Figure 6**, the parameters were adjusted to explore the effects of a larger amount of generations with an increase in the rate of genetic crossover. This increase in crossover resulted in a larger decrease in fitness over the span of 15 generations. The graph of the top fitness levels for each generation shows the effects of an extreme amount of genetic crossover. There almost appears to add more random genetic variation into the population. Even so, the top number of set genes remains stable for most runs with only slight dips in totals throughout the generations.

Conclusion

In conclusion, biological algorithms provide a unique view on solving and optimizing problems using inspiration from real world patterns and biological systems. Clearly, the genetic algorithms are not perfect, nor do they operate entirely as expected. Nonetheless, since their

conception. these algorithms have proved themselves useful in multiple applications throughout history from image processing, to the optimization of molecular structure in chemistry.