Genetic Algorithms

Conner Brinkley

05.01.2020

Biologically-Inspired Computing

The University of Tennessee

INTRODUCTION

The purpose here is to implement a basic genetic algorithm to explore its behavior with varying parameters, with the goal of discovering which parameter settings affect the algorithm's ability to find the best solution. For each combination of parameters, the program outputs data over 20 separate trials in order to more clearly see a pattern of typical behavior. The raw data is outputted to a CSV file for each new parameter combination to make it easier to graph in Excel.

PROCESS (& THEORY)

The simulator starts by initializing the population with random binary strings. Once the entire population (*N*) is initialized, the algorithm starts out by going through and calculating the fitness of each individual in the population and using this to find the fittest individual. The fitness of each individual is calculated by the following fitness function, $F(s)$, where x is the integer that results from reading s as an unsigned binary number, and l is the length of the genetic string (which is specified by the user).

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

Figure 1. Real-valued fitness function.

Once each individual's fitness is calculated, then it records the average fitness, best fitness, and number of correct bits (bits that are 1) for this generation.

Now that the data for this generation has been recorded, the algorithm determines what the next generation is going to look like. To do this, it starts by picking two different parents at random and determining if there is a crossover. The probability of a crossover is one of the parameters that the user provides the simulator (p_c), and if the simulator decides not to do a crossover, then the exact genetic structure of parent 1 and parent 2 is copied directly to child 1 and child 2, respectively. If there is a crossover, then the same thing is done. However, before adding the children to the new generation, a random point is picked in the genes, and then parent 2's genetic structure is copied to child 1's up to that point and vice versa. This way each child contains a portion of each of its parent's genetic structure.

Next, the algorithm determines if there needs to be a mutation in the children's genes. It does this by going through each of their genes and determining either yes or no based on the probability of mutation parameter that the user provides the simulator (p_m). If the gene is mutated, this means that the bit is simply flipped.

Finally, now that it has finished creating the new generation, the new generation becomes the current generation, and the process is repeated for as many generations (G) that the user specified. Average fitness, best fitness, and the number of correct bits (1) is recorded for each generation to see how they change over time with each new combination of parameters.

SIMULATOR

The program was written in C++ with the help of Dr. Van Hornweder's write up, and the source code can be found in the geneticAlgorithm.cpp file in the source directory. The simulator generates a CSV file for each new combination of parameters. To help automate with data collection, another program was written that had an array of strings which each contained a different parameter combination. Then, the program just ran through the array and executed the genetic algorithm program with each parameter string. The source code for this program can be found in the generateData.cpp file also in the source directory.

The simulator was written so that it would be easy to modify in the future to accept more parameters from the user, but for the time being, the program is simply run by the following command after compiling with the given makefile:

USAGE: ./geneticAlgorithm file_ID $\boldsymbol{l}$ $\boldsymbol{N}$ $\boldsymbol{G}$ $\boldsymbol{p_m}$ $\boldsymbol{p_c}$

The parameters are as follows: file_ID is the name of the CSV file that the data will be written to, l is the length of each individual's bit string, N is the population size, G is the number of generations, p_m is the probability of mutation, and p_c is the probability of a crossover.

Once the program finished calculating everything, it outputted all of the data as a CSV file, which was then saved as an Excel workbook to graph the results using Excel's plotting tool.

RESULTS

As mentioned earlier, the program generates data over 20 separate trials for each new parameter combination in order to reduce the risk of getting outlier data. For each new set of parameters, each trial was analyzed to find "typical" behavior. This means that for each new simulation, one trial was selected that adequately represents the trend of all 20.

In total, 14 different simulations were conducted to test each of the five following user inputted parameters: number of genes (l), population size (N), number of generations (G), probability of a mutation (p_m), and the probability of a crossover (p_c). One set of parameters was set as the control set, while the others changed in order to more clearly see how each parameter affects average fitness, best fitness, and the number of correct bits for each generation. Below are the results from the control set, where l is set to 20, N is 30, G is 10, p_m is 0.033, and p_c is 0.6.

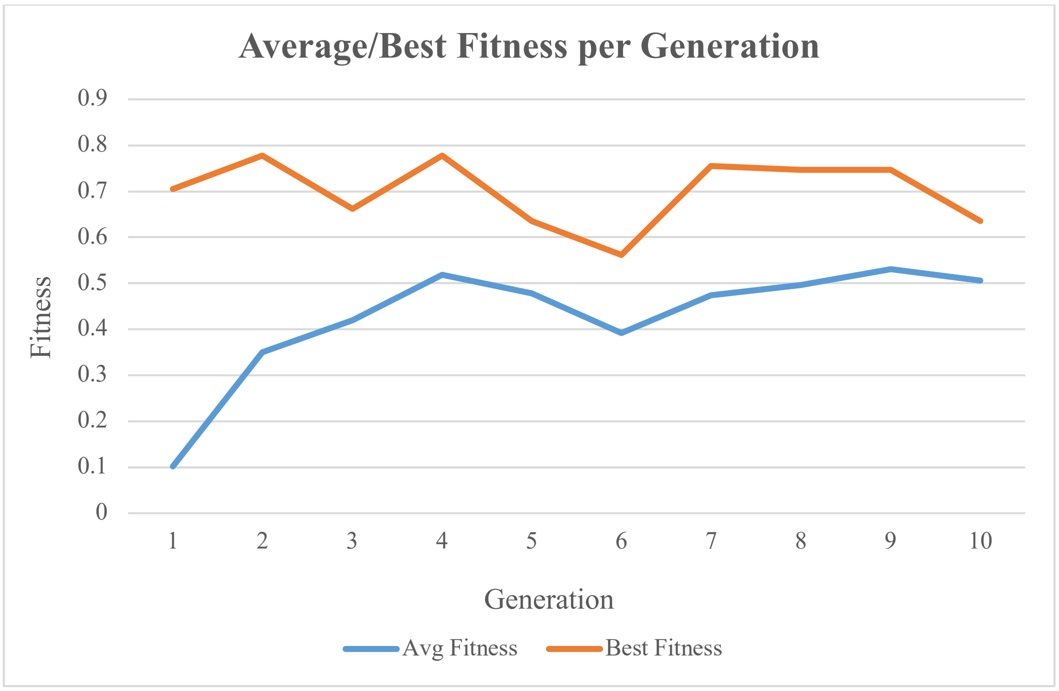

Figure 2. Comparison between average fitness and best fitness across all generations.

The first thing to note is how the fitness of the best individual oscillates within ± 0.3 from generation to generation with a somewhat downward trend, which will stay true for almost every trial in each simulation until the population size is increased. Another thing to note is how the average and best fitness values begin far apart before eventually starting to converge near the end of the generation cycle. This pattern also stays true throughout all of the simulations, and it is more obvious to see as the parameters change, especially when population size is increased.

Below is another graph from the control set that shows how the number of correct (1) bits in the best individual changes over time.

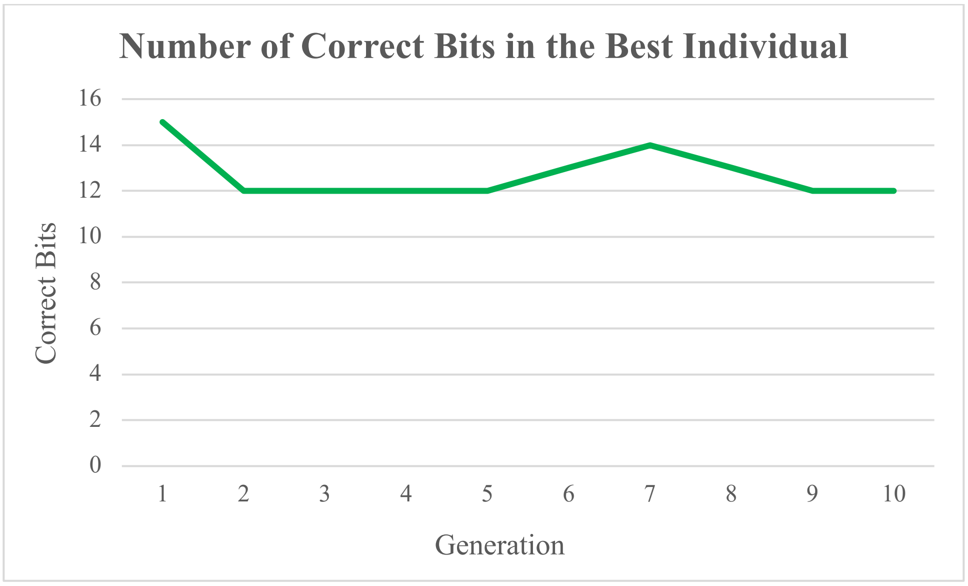

Figure 3. The number of correct bits in the best individual when the gene size is 20.

One thing to note about this data is how the number of correct bits tends to stay the same within ± 3 of the starting point, but generally it ends with less than it starts with. This will stay true throughout all of the simulations. Some of them do not appear to follow this rule and instead stay constant or even increase, but this is most likely because the parameters did not allow the algorithm to have enough time to develop a trend. The idea is that if only generations 4-7 were analyzed in the above graph, then it would look like an upward trend, but with the bigger picture, it is actually the opposite. One explanation for the downward trend in correct bits in the best individuals could be that, as stated before, normally the best fitness decreases over time in order to converge with the average fitness.

Only four parameters were actually changed to see how they affect the data, since the probability of mutation (p_m) stays inversely proportional to the population size. The first parameter that was tested was the number of genes (l) in the genetic string. Unfortunately, there is not much to say about this one because the data looks very similar to the control set. It is obvious that the average fitness and the best fitness lines are converging, but it is not clear how the number of genes really affects the data. Perhaps when combined with another set of parameters, it would be more obvious to see. Below is the data when the number of genes in the genetic string is changed to 5.

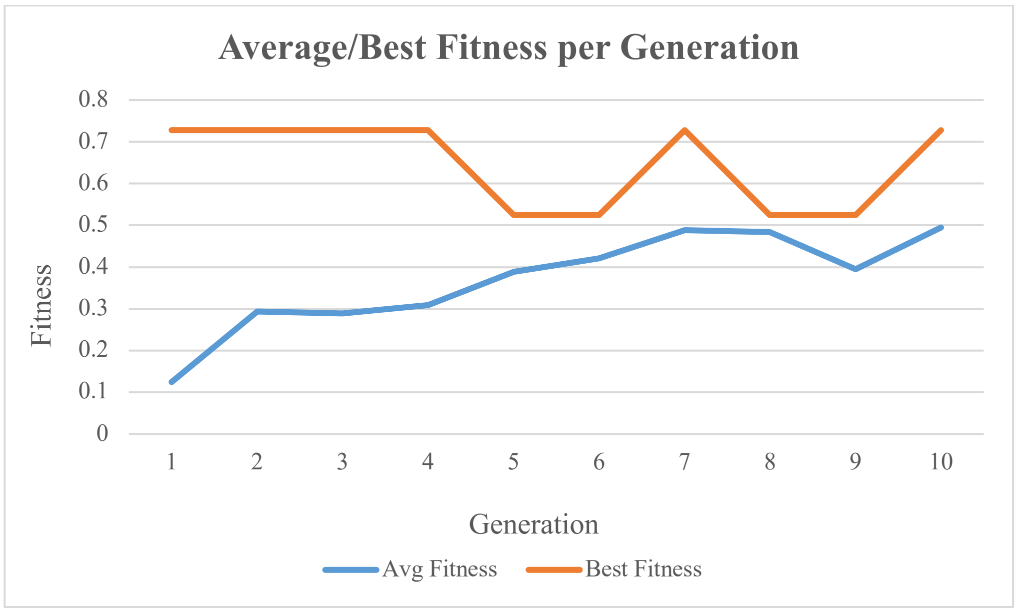

Figure 4. Average and best fitness over time for simulation 2.

The next parameter that was tested was the population size, which ended up being the most promising. It was very clear to see that the more the population size was increased, the easier it was to identify a trend among all of the data and to see where the average and best fitness converge. Below are the results when the population size was increased to 1000.

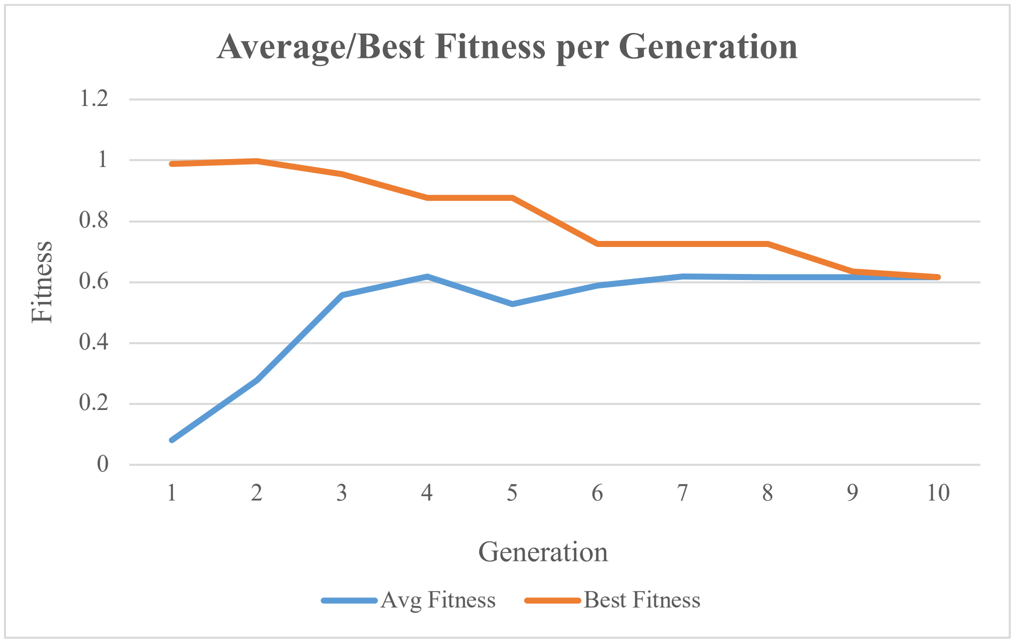

Figure 5. Average and best fitness over time for simulation 8.

The downward trend in the number of correct bits in the best individual is also much easier to see when the population size is increased as shown below.

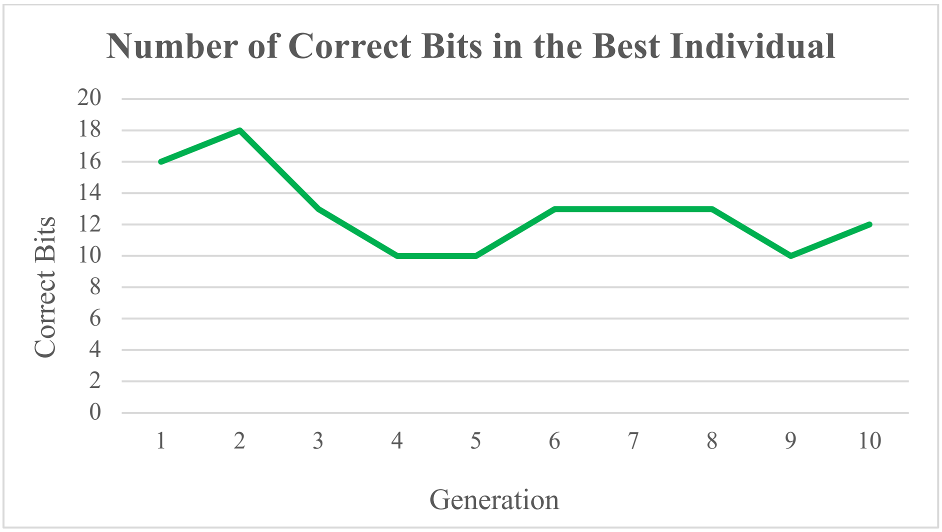

Figure 6. Number of correct bits in the best individual for simulation 8.

When the number of generations (G) was increased, it made finding trends almost impossible based on the raw data because of the amount of variability between each generation. Below is a graph when the number of generations was increased to 100.

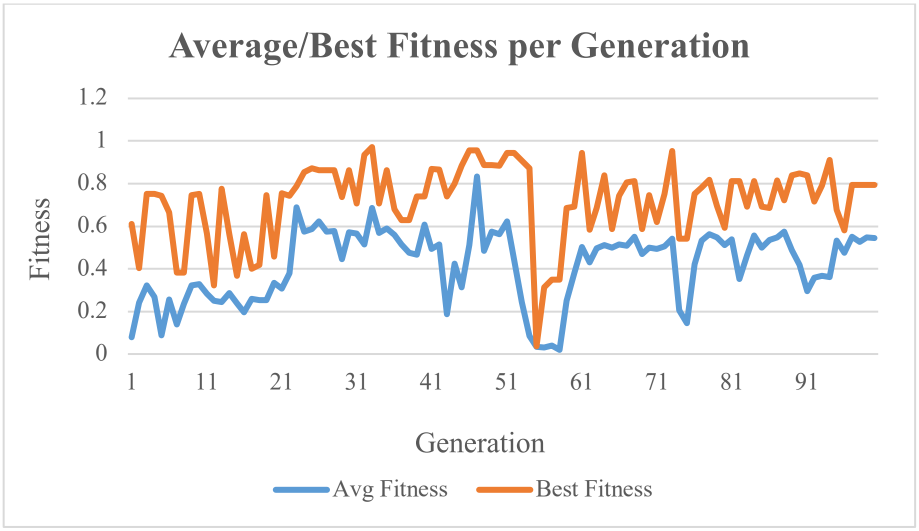

Figure 7. Average and best fitness over time for simulation 11.

Finally, when the probability of a crossover (p_c) was increased, the data seemed like it took a much longer time to converge, if it ever did. Below is a graph of the average and best fitness when p_c was increased to 0.99.

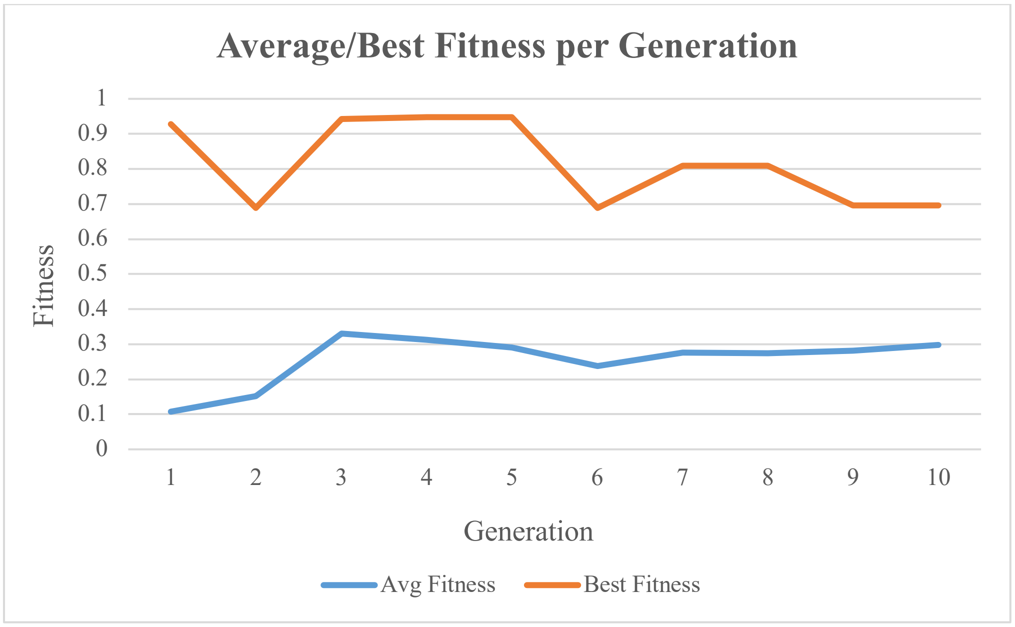

Figure 8. Average and best fitness over time for simulation 14.

CONCLUSION

Based on the resulting data, it seems like the best way to optimize the algorithm to find the best solution would be to increase the population size to around 1000, keep the number of generations around 10-20, and keep the probability of a crossover around 0.6. It is hard to say what the optimal number of genes are based on the data in this experiment, but in the future, it might be interesting to take the other parameters based on what was learned and combine them with a varying number of genes to see how that affects the trends, especially with the increased population size.