Genetic Algorithms

Thomas Rotunno | COSC 420

4/8/2020

Introduction

The core idea of evolution is that a population (whether it be animals, bacteria, etc.) evolves over time to better fit its environment. If we take this idea and abstract it, we can begin to use it in regards to computer programming – the application of this that we will be exploring is the idea of a "genetic algorithm", an algorithm that simulates evolution.

John Holland in the 1960's first came up with the idea of a genetic algorithm – he theorized that a computer could be used to program data into DNA-like structures that express a kind of "fitness" that could be toyed with. The DNA-like structures, across programmed generations, would be able to mate, cross over and mutate with each other in order to produce simulated offspring that would evolve towards a higher "fitness" value. Practically, this "fitness" that the population would evolve according to can be modeled as a mathematical function, taking the DNA-like structure and its properties as parameters.

Methodology

For the purposes of this project, the population can be modelled as a collection of individuals, each containing a binary string of "DNA" that represents their genes. A few key parameters must be defined in order to apply boundary conditions to this evolutionary process:

- l, the length of the binary string of "genes"
- N, the number of individuals in the population
- G, the number of generations
- p_m, the probability of a mutation occurring in one of the genes upon reproduction
- p_c, the probability of two parents crossing their genes upon reproduction

Now that we have these simulation parameters defined, we must come up with some way of measuring the "fitness" of an individual in the population. The fitness function used for this experiment is as such:

$$F(s) = \left(\frac{x}{2^l}\right)^{10} \tag{1}$$

Where x is the unsigned integer representation of the l-digit binary string s that is passed in. If one takes a close look at (1), it can be seen that the range of this function is $0 \leq F(s) \leq 1$ where an individual with a gene string s consisting of all 1's would be considered to be of the best possible fitness. It is also beneficial to define a function for normalizing this fitness:

$$NF(s) = \frac{F(s)}{Total\ Population\ Fitness} \tag{2}$$

where the sum of all normalized fitnesses in a population will be equal to 1.

To begin the simulation, a size-N array of individuals is created to represent a population P. Each of these individuals has an l-digit binary array containing randomly-chosen 1's and 0's. The fitness and normalized fitness of each individual in P is then calculated and stored alongside a running total RT of these normalized fitness values:

$$RT(s_i) = NF(s_i) + \sum_{j=0}^{i} NF(s_j) \tag{3}$$

Once these have been calculated, two parent individuals are selected to mate based on certain criteria. Two random floating-point numbers between 0 and 1 are chosen, we'll call them r_1 and r_2. Since the RT value of each individual in the population is built upon itself and previous individuals, it is naturally sorted in ascending order. Going through P, the first individual with

RT value higher than r_1 is chosen as the first parent, with the second parent being chosen the exact same way with the only caveat being that the chosen parents cannot be the same individual (in order to avoid self-mating). If this is the case, a new r_2 value is generated until the chosen parents are different individuals.

Next, the mating process takes place. The two parent's genes are copied directly to two new individuals, the children. Another random floating-point number between 0 and 1 is generated (we'll call it r_1 to be consistent). If $r_1 < p_c$, a crossover occurs – this is simulated by choosing a random index in the gene array between 1 and $l - 1$, and swapping the two children's genes after this index. After this process, each gene of each child will be tested against a random floating-point number r_2, where if $r_2 < p_m$ the gene's binary value will be flipped (a new value of r_2 will be generated for each gene).

Finally, both children will then be saved into a new population P' which then takes the place of the original population P. This fitness calcuation and mating process then repeats itself G times, and valuable statistical data is recorded for each generation.

Results

For each generation g, four pieces of data were recorded regarding the individuals in P:

- Average Fitness across all individuals
- Average number of "1" genes across all individuals
- Fitness value of the individual with the highest (best) Fitness value
- Number of "1" genes in the individual with the best Fitness

Using the suggested starting parameters ($G = 10, N = 30, l = 20, p_c = 0.6, p_m = 0.033$) the following results were obtained:

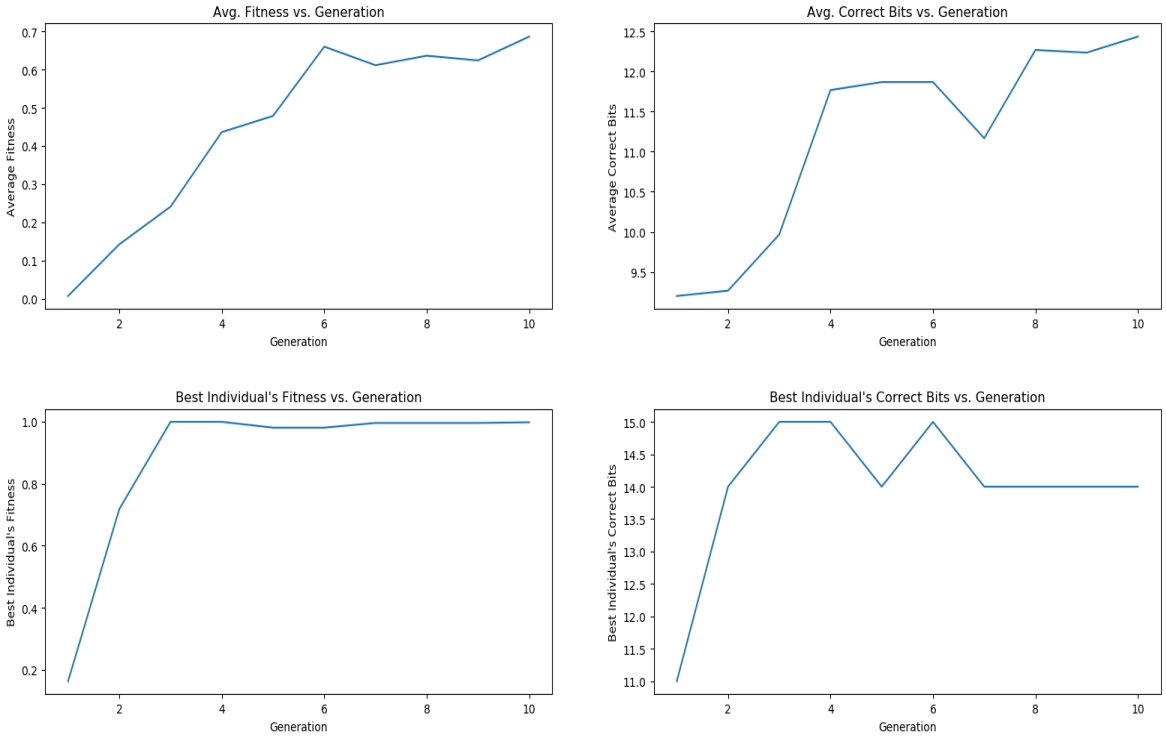

Figure 1: Generational Plots with: ($G = 10, N = 30, l = 20, p_c = 0.6, p_m = 0.033$)

As can be seen from figure 1, the Average Fitness plot seems to trend linearly upwards. After only 3 generations, the population consistently had at least one individual with a fitness of ~1.0. However, the evolution doesn't seem to trend linearly towards all individuals having l correct bits, as there is a dip in the Average Correct Bits vs. Generation plot at around $g = 8$. To see if the dip was just a product of too few generations, the same parameters were supplied again but with $G = 100$:

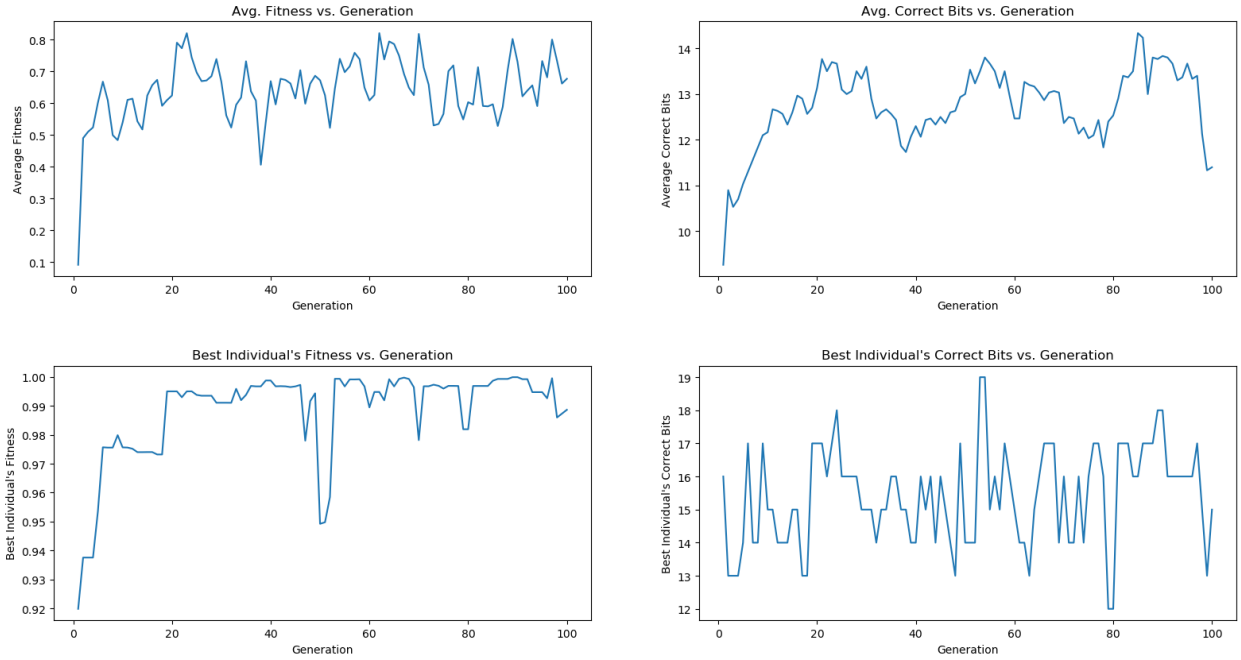

Figure 2: Generational Plots with: ($G = 100, N = 30, l = 20, p_c = 0.6, p_m = 0.033$)

Once again, the average stability seemed to settle at around 0.7 (±0.15) and the average number of correct bits seemed to settle at around 13 (±1) rather than trend upwards. After some brainstorming, another simulation was performed, but this time with $p_m = 0$:

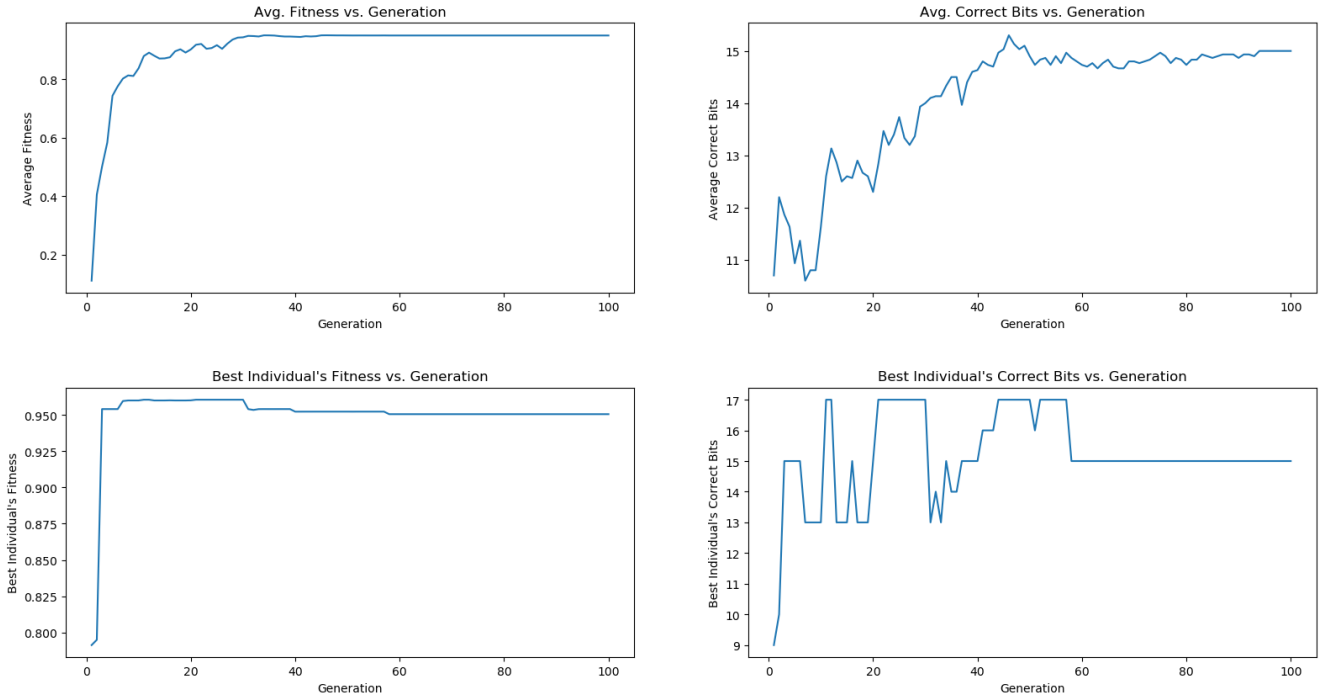

Figure 3: Generational Plots with: ($G = 100, N = 30, l = 20, p_c = 0.6, p_m = 0.0$)

With the possibility of a mutation nullified, the average fitness and average correct bit plots came out much more stable. However, after running with these parameters multiple times, the best individual's number of correct bits does not get as close to l as it does with the possibility of mutations.

As a final test, the simulation was run with some modifications to the other parameters not changed yet (N, l and p_c). The idea in changing these was to add more diversity across generations.

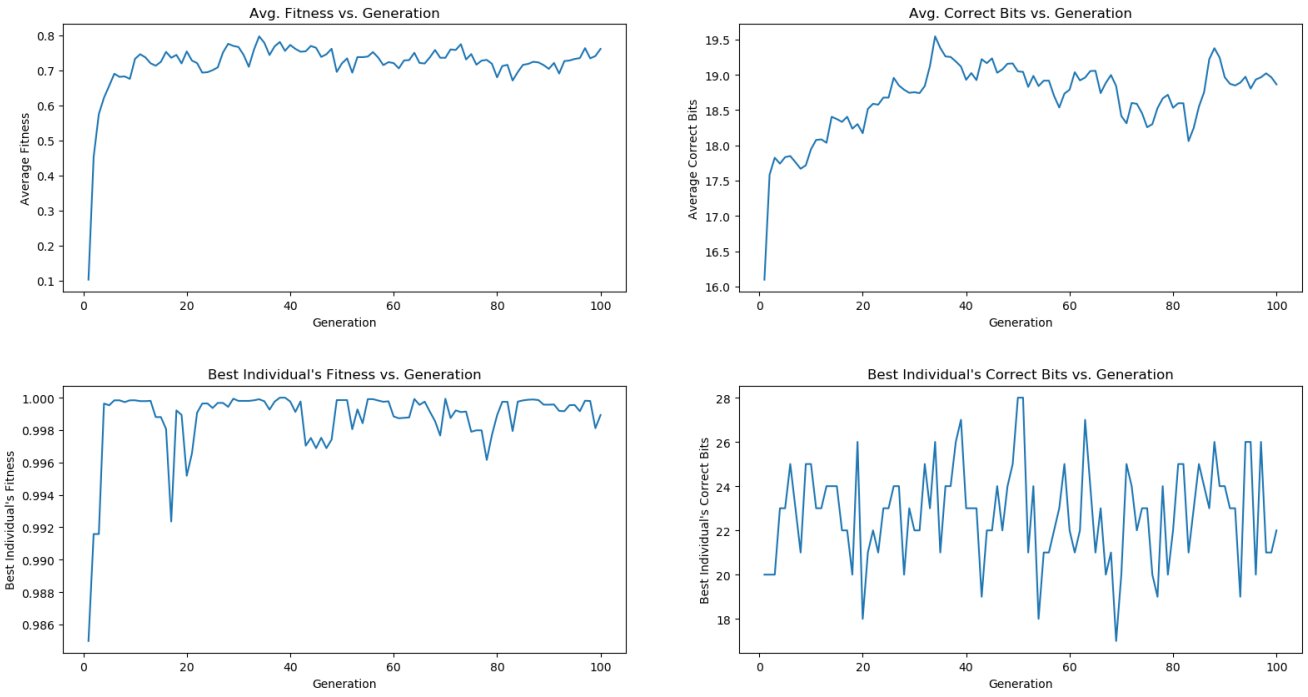

Figure 4: Generational Plots with: ($G = 100, N = 250, l = 32, p_c = 0.8, p_m = 0.033$)

By changing p_c and N to higher values, this allowed for more mixing and diversity between generations. The raising of l did not seem to change much in the resulting data. What resulted from the changes to p_c and N was a more stable average fitness across generations, as well as a much higher average number of correct bits.

Discussion / Conclusion

From all of the simulations, the most interesting data points by far were the Average Fitness vs. Generation and Average Correct Bits vs. Generation. The other metrics such as the Best Individual's Fitness and their respective number of correct bits were useful, but ultimately did not reveal much about the evolution of the system. The key factor in constraining the average

fitness / correct bits values to a high, low-variance number was to make N large. Increasing p_c to be closer to 1 alongside this promoted more genetic diversity amongst the children. A higher l is necessary as N is increased, as it allows for more possible unique combinations of genes. As for p_m, the possibility of a mutation helps to achieve higher fitness in the early generations. However, as p_m is increased, the stability of the average fitness / correct bit values tends to drop tends to drop due to the randomness introduced.

In order to increase the stability of the two averages, it seems that a low p_m value is necessary for the fitness function used to consistently produce values close to 1. If a mutation occurs and a highly-significant bit is flipped, the individual's fitness value decreases down significantly.