Genetic Algorithmic Simulation

Johnny Stuto
UIdaho CS 515
Assignment 1
February 24th, 2023
stut2030@vandals.uidaho.edu

Abstract—A population of 1000 individuals possessing a genome of 1000 nucleotides are simulated over 1000 generations. Three popular selection methods are used with three different crossover techniques under a common mutation algorithm. Two metrics are used to track the fitness and diversity of the mean genome of the population. The fitness score is a general CG island counting algorithm and the diversity score mirrors the overall average nucleotide distribution of the population.

I. INTRODUCTION

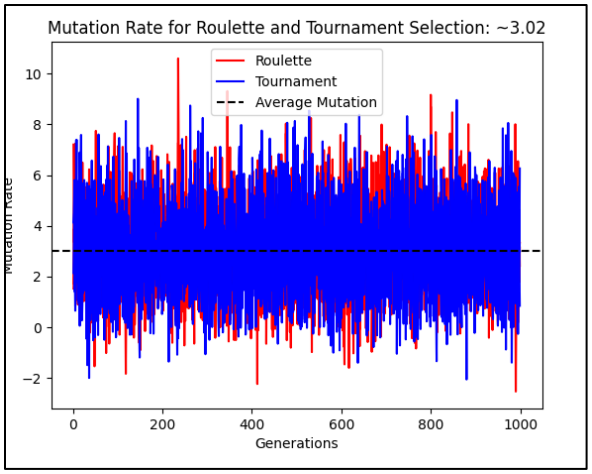
Genetic algorithms, inspired by the process of natural selection and a subclass to the wider field of evolutionary algorithms, is the mechanism by which a population of candidate solutions is optimized iteratively toward desired solutions. In this experiment, a genetic algorithm that simulates the evolution of one thousand DNA nucleotide genomes, within a population of one thousand hypothetical species for one thousand iterations is run. Three common methods of genomic crossover between parents are paired with three common methods of individual selection from the population. Using a common stochastic mutation function bound by a seed, these nine varying experiments are monitored for time, fitness & diversity. Additionally rating score will be computed by scaling the average fitness by the average diversity. Fitness will be monitored by counting the total number of CG islands in the genome. The diversity is proportional to distribution of the four nucleotides. Algorithmic details for all mechanics listed above will be discussed below.

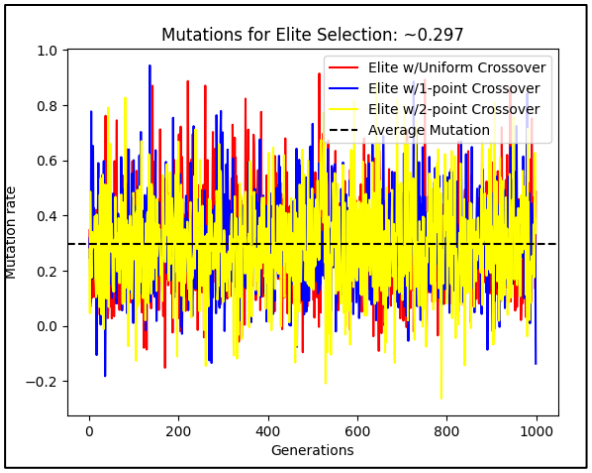
II. MUTATION, SELECTION & CROSSOVER ALGORITHMS

A. Mutation algorithm

A simple common mutation algorithm is used by all crossover/selection combinations. The theoretical mean of the algorithm calculated over time is approximately 0.316 with variance 0.00416. This works well over the one thousand iterations.

The mutation function uses the random module to generate a random mutation rate. The random seed *is not used in the mutation function due to the need of stochasticity* within the genetic algorithmic. The imposed random seed on the mutation function caused the fitness to flat line across all experimental pairs.





Notice the difference between mean mutation.

The mutation function has three main components:

1. A mutation base variable is assigned a random value between 2 and 4, inclusive, using the *random.uniform* function. This variable represents the base mutation rate.
2. Another variable is then assigned a random value from a normal distribution with mean 0.1 and standard deviation 0.05, using the *random.gauss* function. This variable represents the variance of the mutation rate.
3. The final mutation rate variable is calculated as the product of mutation base and variance from above. This represents the final mutation rate.

B. Crossover Algorithms

Crossover is the method in which the genetic information of two parents is stochastically recombined to generate children. Here we compared three different crossover methods commonly used in genetic algorithms.

1. Uniform Crossover chooses a letter from either parent genome with equal probability. In our method, a random uniform integer is chosen between 0-9 and then compared to another random integer with range 0-100. If the first random variable is larger than the second, the iteration number is used to place the swapping location on the parents' genome.

2. One-point Crossover is where one point on both selected genomes is stochastically chosen. This chosen crossover point is where the letters to the right of that point are swapped between the two selected genomes. The

resulting two children carry some part of each parent genome. The genome length is used as the random number generator maximum.

3. Two-point Crossover is where two points on the selected genomes is stochastically chosen. These two chosen crossover points are the swapping delimiting left/right markers between the two selected genomes. The resulting two children carry some part of each parent genome. The genome length is used as the random number generator maximum.

C. Selection Algorithms

The selection of a genetic algorithm is where individuals in the population are stochastically chosen to be parents. The parents are then subject to the crossover function and then onto the mutation function. In this experiment, only two parents are chosen by the selection function and then sent into the crossover function for recombination. The three selection algorithms are detailed below:

1. Tournament selection choses 2 individuals and a 'tournament' is conducted between the two selected. Stochastically, the fittest individual among the two has a greater chance of selection to be passed on to recombination.

2. Roulette wheel selection derives from the spinning roulettes in casinos. This algorithm produces outcomes proportional to fitness. Higher probability of selection would be dependent on higher fitness scores.

3. Out custom Elite algorithm only keeps the top two candidates from the population. This algorithm sorts the population by fitness and then directly choses the top two individuals which then are sent to recombination.

III. FITNESS AND DIVERSITY

A. Fitness metric

The fitness function calculates a metric based on the total count of CG islands in the genome. No other restraint in this algorithm is applied (i.e., in padding characters around the CG island) In biology, these islands are regions of DNA where a C nucleotide is followed by a G. In our metric, this condition is calculated in the same spirit. The following algorithm details this simple metric.

1. First the algorithm initializes a fitness score of zero.
2. Conditionally and iteratively checks the length of genome is prevent runover.
3. If letter (i)= C AND letter (i+1) = G, then the fitness score is incremented by 1.
4. Returns fitness score.

B. Diversity metric

The diversity function calculates a metric based on the frequency of each character. Initial experimental inclination was that the distribution of letters in the genome would tend toward a CG-dominated distribution due to fitness pressure. Keep in mind that a diversity score of 1 would mean that the genome contains an equal number of all possible characters or considered perfectly diverse. The following algorithm details this simple metric.

1. First it creates an empty dictionary to store the counts of each character in the genome.
2. Iterates through each character in the genome adding one to the count of the character in the dictionary. If the character is not already in the dictionary, set its count to 1.
3. Calculates the frequency of each character by dividing its count by the total length of the genome (n).
4. Calculates the final metric by iterating through each frequency, calculating the product of the frequency and its complement (1 - frequency) and then summing up the results.
5. Return the diversity score.

C. Rating metric

The rating metric is just an experimental score that is the fitness score multiplied by the diversity score. Theoretically, the lower diversity score scales the fitness score downward.

IV. ALGORITHMIC COMPLEXITY

The potential computational complexity for an evolutionary algorithm with the following input parameters is difficult to estimate precisely, but we can make some rough estimates:

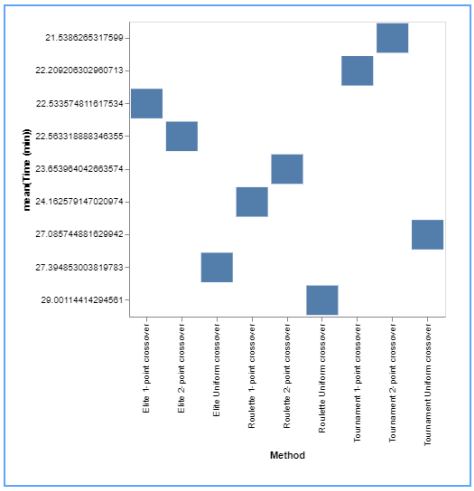


Figure 1: Computational Time in minutes

1. Genome size (*genomesize*): The size of each genome is 1000, so the algorithm needs to manipulate a set of 1000 characters for every individual in the population.
2. Number of generations *(gens)*: The algorithm will run for 1000 generations, so it will perform the evolutionary process for 1000 times.
3. Population size (*popsize*): The population size is set at 1000, so the algorithm will have the stochastic opportunity to manipulate 10000 individuals in each generation.

The main computational complexity of the evolutionary algorithm is derived from evaluation of fitness & diversity functions for every individual in the population which is likely have a linear complexity with respect to the genome size.

Potentially, complexity of this evolutionary algorithm can be very high since it grows linearly with the genome size, the number of generations, and the population size.

The experiment revealed that the Uniform crossover methods across all selection methods took the longest time to compute. This is due to the iteration across all characters in the genome for everyone in the population. See figure 1 above for all computation times.

V. SIMULATION RESULTS & CONCLUSIONS

The three mean metrics are tabulated below in table 3. Notice that the Elite selection methods dominate the average fitness and rating scores even with the burden of lower diversity scores scales downward. The Elite selection methods do have some competition from the Tournament methods particularity from the one-point crossover simulation. Another observation is the diversity scores of the Roulette method are higher than the rest of the experiments which is delivered with lower fitness and thus rating. The mutation rate is inversely proportional to the fitness of the genome. Diversity is bolstered by more mutation which makes sense.

Take notice of the computation times of each experiment. The uniform crossover method added about seven minutes additional time to compute for each. This is due to the iterative addition of letter-by-letter mutation rather than endpoint mutation of the other crossover methods. The Roulette method with Uniform crossover ran for 29 minutes, which is the longest, probably due to roulette random number generation, sorting, probabilities calculation and genome length random number generation.

Summary for all experimental evolutionary simulations

	Method	Time (min)	ave mutations	ave score	ave diversity	rating
0	Roulette Uniform crossover	29.00	3.14	119.51	0.73	87.50
1	Roulette 1-point crossover	24.16	3.06	140.29	0.73	102.78
2	Roulette 2-point crossover	23.65	2.96	137.62	0.74	101.18
3	Tournament Uniform crossover	27.09	2.96	329.29	0.63	208.48
4	Tournament 1-point crossover	22.21	2.93	363.83	0.62	225.37
5	Tournament 2-point crossover	21.54	2.98	351.72	0.63	220.87
6	Elite Uniform crossover	27.39	0.30	368.69	0.64	235.08
7	Elite 1-point crossover	22.53	0.30	380.19	0.63	237.68
8	Elite 2-point crossover	22.56	0.30	369.37	0.64	234.63

Table 1: Result Metrics from all experiments

Notice below in table 4, the maximum fitness score obtained for each experiment differs at which iteration it (*Max@iter)* was calculated. The lowest max fitness score, Roulette with Uniform crossover, occurred at iteration 625. The others occurred at later iterations, closer to the ending iteration of 1000. This would suggest that Roulette with Uniform Crossover experiment, even with the larger complexity and calculation times, did not improve over the last 374 iteration

Method	ave mutations	max mutations	ave score	ave diversity	max score	Time (sec)	Max @ Iter
Roulette Uniform crossover	3.135386	10.552784	119.507162	0.732138	136.710	1740.068649	626
Roulette 1-point crossover	3.059377	10.552784	140.290757	0.732599	160.531	1449.754749	893
Roulette 2-point crossover	2.957023	9.681600	137.616544	0.735196	155.327	1419.237843	921
Tournament Uniform crossover	2.961795	9.380843	329.287962	0.633117	390.927	1625.144693	956
Tournament 1-point crossover	2.931920	9.556901	363.832165	0.619426	420.537	1332.552378	851
Tournament 2-point crossover	2.975921	9.256429	351.717123	0.627980	409.830	1292.317592	938
Elite Uniform crossover	0.298273	0.888121	368.687200	0.637603	422.657	1643.691180	977
Elite 1-point crossover	0.298344	1.157976	380.192348	0.625168	430.000	1352.014489	984
Elite 2-point crossover	0.302159	0.913203	369.369477	0.635221	425.000	1353.799133	961

Table 2: Additional Stats from all Experiments

Other interesting graphics below in figure 2 reveal the fitness score flat line effect of the Roulette with Uniform Crossover. The other two curves suggest more iterations would convey more fitness. Roulette seems to flat line at a fitness score around 150. The algorithm might be flawed, or the random seed could have prevented variability that would have allow greater fitness scores.

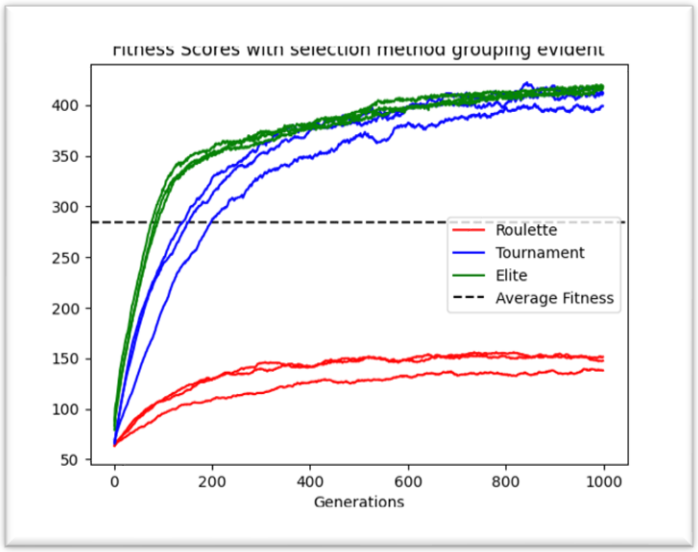


Figure 2: Roulette Flat line

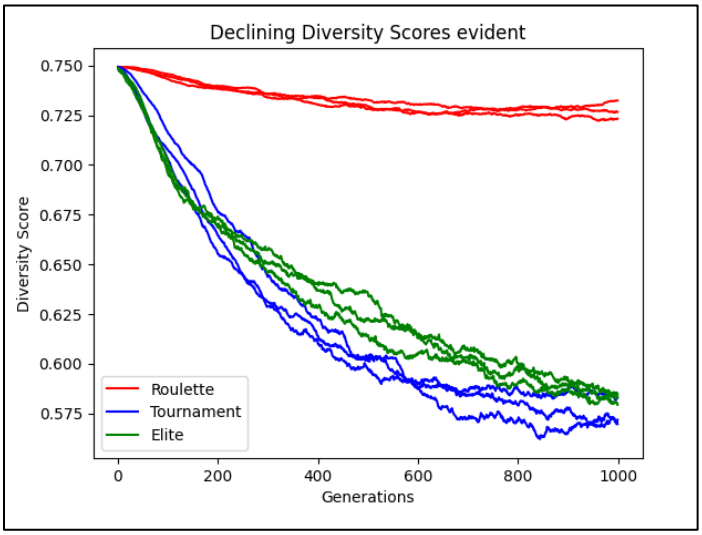


Figure 3: Diversity

Diversity is inversely proportional to rating due to the fitness selection imposing higher C-G islands to form. See above figure 3.

Elite selection seems to dominate over other two selection methods. I suspect that the relatively low mutation rate slows the learning rate of the genetic algorithm allowing it to overcome local optimization traps.

VI. PYTHON CODE

This code is available for inspection and reproducibility at the follow GitHub address:

https://github.com/ArcturusMajere/515

VII. REFERENCES

[1] Soule, T: code from https://canvas.uidaho.edu/courses/17109/modules

[2] https://en.wikipedia.org/wiki/CpG_site