Austin Starks

Assignment 3: Genetic Algorithm Report

Introduction

My goal for this assignment was to implement a genetic algorithm to generate a sentence. Genetic algorithms get their power from mimicking the process of natural selection. We start with a target sentence. A population of candidate sentences is randomly generated, and each sentence is evaluated by a fitness function. The best individuals in the population are selected to create a new generation, and this cycle repeats until we are at the target sentence. My goal for this assignment was to implement a genetic algorithm that mimics real-life mutations.

Part 1: Implementation

Calculate Fitness:

The fitness of a sentence is defined as the summed absolute difference for each character in the sentence. If the target phrase is “aa” and the current phrase is “ac”, the summed absolute difference is ‘a’ – ‘a’ + ‘a’ – ‘c’, which is 0 + 2.

To ensure the fitness value is 1 when the difference is minimal and 0 when the difference is maximal, the final fitness value is obtained by doing 1 / (1 + 0.05*score). This equation was obtained by experimentation and has the property of being close to 0 when the difference is large and exactly 1 when the difference is small.

Crossover

When organisms split via meiosis, they undergo a process called *recombination*, which is when two molecules of DNA exchange information. This process creates genetic diversity and

ensures each haploid is not genetically identical. It is the reason why siblings do not share the same DNA sequence.

We cannot mimic recombination exactly using a genetic algorithm. Nevertheless, we can try to ensure two children from a set of parents are genetically different. To do this, I randomly select genes from each parent to make a child. I iterate over each parent, and randomly decide if the child is getting parent 1's gene or parent 2's gene.

Mutate

This is the most interesting function that I've implemented. In biology, there are many different types of mutations. The most common mutations are missense mutations and frameshift mutations.

Missense mutations are mutations that happen when one allele gets converted to another allele, and that mutation changes the protein sequence that the gene encodes for. These mutations don't typically have huge effects on the organism's fitness (although some missense mutations do).

Frameshift mutations are mutations that result in insertions or deletions of alleles that shift the DNA sequence. These mutations typically have large effects on the organism's fitness because they result in drastic changes in the proteins the genes encode for.

To mimic real mutations, I implemented missense mutations and frameshift mutations. A missense mutation was simple; I randomly selected a gene and changed it. This change might have been minor (changing an 'e' -> 'f') or major (changing an 'e' to a completely random character).

To mimic frameshift mutations, I had to be a little more creative. A major consequence of frameshift mutations (in protein-encoding genes) is the change of large portions of the encoded protein. To mimic this, I randomly selected a number and direction (left or right), and then randomly changed all of genes in that direction

Lastly, it can be argued that the above implementation might break the Genetic Algorithm's stochastic behavior. Changing a letter from 'e' to 'f' might not fit within the standard rules of a genetic algorithm because the mutation is not entirely random. Thus, I have also implemented a more standard implementation of the mutate function in the source code. At the end of the report, the results for the standard implementation and the modified implementation are both shown.

Natural Selection

I put every individual in the population in the mating pool. However, individuals in the top 25% percentile in fitness are added to the mating pool at much higher frequencies.

Generate New Population

In creating a new population, we select two individuals in the mating pool and have them produce an offspring. The father is almost always the most fit individual in the population. Though, to ensure genetic diversity, the father can instead be randomly chosen from the mating pool. The mother is always randomly chosen from the mating pool. If the mother and father happen to have the same genes, the mother is re-chosen until they have different genes.

Part 2: Results

Implementation with modified mutate function

Settings	Population Size	Mutation Rate	Average of 20 generations	Execution Time (in seconds)
1	200	0.01	757.0	249.8
2	200	0.02	370.0	99.2
3	100	0.01	1410.1	177.5
4	10000	0.01	26.0	766.7
5	1000	0.01	151.9	200.0
6	1000	0.02	83.9	114.3

Implementation with "standard" mutate function

Settings	Population Size	Mutation Rate	Average of 20 generations	Execution Time (in seconds)
1	200	0.01	711.3	995.7
2	200	0.02	1645.6	480.6
3	100	0.01	7303.8	941.5
4	10000	0.01	57.85	2170.9
5	1000	0.01	641.2	939.3
6	1000	0.02	355.1	454.6

Important Note: These results were obtained on an older machine, so they may not be typical.

Part 3: Answer Questions

1. What happens if you *increase* the mutation rate (Setting 2)? Does the algorithm converge faster or slower than Setting 1? Explain why.

   A higher mutation rate increases the rate of convergence because we're more likely to change an incorrect gene to a correct one. We also have increased levels of genetic diversity, so when we pick two individuals from our mating pools with high fitness levels and different genes, we're more likely to generate an offspring with the 'traits' that we're looking for.

2. What happens when you *decrease* the size of the population (Setting 3)?

   When we decrease the size of the population, we increase the number of generations it takes to get target sentence.

3. What happens when you *increase* the size of the population (Setting 4)?

   When we increase the size of the population, we decrease the number of generations it takes to get the target sentence.

4. How does the size of the population impact the computational cost to process (generate and evaluate) the population?

   As the population size increases, it takes more computational power to look for the best offspring and pick offspring. However, we're also more likely to have created the best offspring because we are producing more offspring per generation. Thus, the average number of generations it takes to find the target sentence is lower in large populations.

5. A genetic algorithm is a kind of stochastic algorithm based on the theory of probability. On top of stochastic behavior, GA has parameters to control its execution. How do the

mutation rate, crossover probability, and population size impact the genetic algorithm? In other words, explain how these three parameters work together to make the algorithm converge to a solution.

These parameters help control the genetic diversity of subsequent generations, and ultimately increase the fitness of individuals. By crossing over two high fitness individuals, we're hoping to pass down desirable characteristics and create an even more fit child. Moreover, by introducing mutations in the population, we're hoping to introduce novel genes that might also increase the fitness of the individual.