Nick Williams

Population Simulator Project Report

Project Statement:

I set out to create a genetic algorithm that would be able to make individuals well suited to any given environment. This task involves making a random environment (world) generator, matching traits of an individual to traits of their environment, and picking the best ones to make offspring based on their own personal traits. The world had 5 traits (vegetation density, number of predators, land mass coverage, food scarcity, and temperature) that matched up to the 5 traits each individual has (size, mobility, dehydration adaptations, socialization, and cold tolerance. All traits for world and individual were on a scale of 1 - 10. For the world, a higher number meant a more difficult environment to survive in. For the individual, a higher number meant that they had more of that adaptation which benefited them for the matching environment trait.

Algorithm:

I used a genetic algorithm to make the population simulator. Before the algorithm truly begins, it generates an initial population of random individuals. The traits of these individuals in the initial population do not sum to more than 40. This point cap was created for the starting individuals to prove that the algorithm improves their stats over time.

Once the initial population has been created, the algorithm then checks to see how fit each individual is within the population. It does so by doing some basic calculations on the difference and ratio between the individuals characteristics and their matching world traits.

After fitness has been determined, a group is selected to reproduce. This group of individuals is randomly selected with probabilities weighted more heavily to the more fit individuals.

Each of those who have been selected for reproduction pair with another chosen individual to make two unique children. The first child will have the first two genes of parent

A and the last three genes of parent B, while the second child will have the first two genes of parent B and the last three genes of parent A.

The generated offspring will then have a small (currently 30%) chance to mutate one of their genes into a random value that could be better or worse.

~~~~~~~~PSEUDOCODE~~~~~~

```
For x in initialPopCount:
    Create an initial population that has random genes
For f in initialPop:
    Calculate fitness values for each member of the population
randomChoices: choose the most fit members through weighted probabilities based on
    fitness
Make offspring using roughly half the genes of one fit member and the rest of the genes
    from another fit member
For s in offspring:
    Small chance for a random gene of an offspring to mutate to some random value
```

Analysis:

This solution is a very effective way to solve this very specific and basic problem.

In terms of strengths:

- It always produces at least some offspring that are capable of surviving the world
- It would be pretty easy to add more to this code to make it even more powerful or fun to use
- I was unable to successfully implement a genetic algorithm to solve the 8 queens problem, so this has been a personally sweet victory

In terms of weaknesses:

- This is a problem I made up, so I also got to make the rules for what solving it looked like
~~~~~~~~

- Did not have enough time to make the fitness function cool with multiple traits of the world matching up with multiple other traits of the individual.

This directly relates to the content we covered on local search in class. A genetic algorithm is a type of local search because it does not matter how we got to the final solution, just that we got there. I would also like to mention that I coded the entire thing from scratch. The only things I referenced were the chapter 4 slides and my previously failed implementation of a genetic algorithm for the 8 queens problem.

Summary:

The final product does work very well. The majority of randomly generated worlds yield above a 50% survival rate, with it often exceeding an 80% survival rate. Each major component highlighted in the pseudo code was tested individually with a lot of debugging, and was found to be functioning correctly. It is also easy to use as incorrect inputs do not break the program, nor will they end or skip something. I am quite proud of this work.