Genetic Algorithm Report – Part 1.1-1.3

Thomas Meehan – 20425946

1.1 One max problem

Description of aspects of algorithm

This problem focused on using a genetic algorithm to evolve a string to the maximum fitness level possible.

Representation: Each population consisted of individuals which were represented as binary strings of 0s and 1s, these were initially made randomly with a count of all the 1s in the string representing how fit an individual was.

Fitness Function: The fitness function calculated the fitness of a string by counting the 1s within it, this function was used to assess how the algorithm improved over time by using these fitness scores to see how the populations average fitness over each generation.

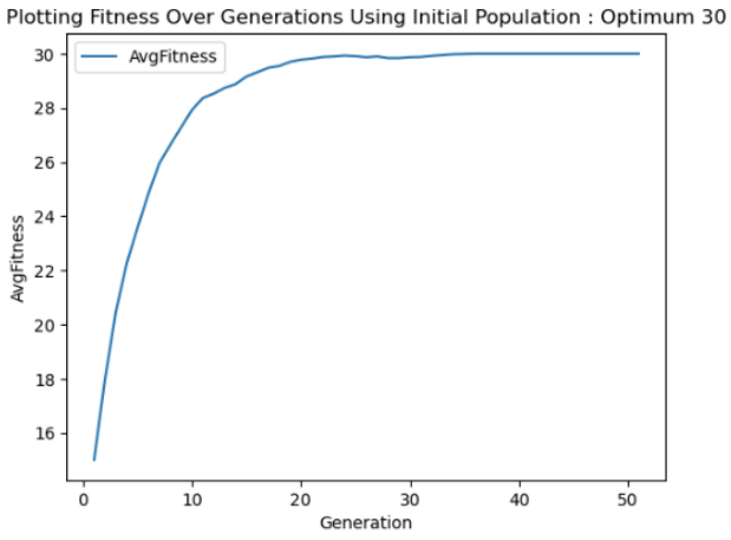
Selection: my selection involved randomly choosing a pair of parents from the current population, as the overall fitness of the population increased so did the parents and therefor the next generation.

Crossover: my crossover function determined a random crossover point and created two new child strings based on the two parent strings, taking the substring before the crossover point of one parent and combining it with the substring after the crossover point of the second parent.

Mutate: my mutate function mutated bits within the string provided a randomly generated number was below the mutation rate, this mutated string was then returned, in this case as we are searching for the optimum fitness of all 1s in the string the mutate function checked each bit to see if it was one and provided the mutation rate was greater than the random number flipped bits with a value of 0 to 1.

Elitism: I did not implement elitism in this problem due to the new population being constructed from the previous one meaning the genetic material from the previous generation is passed on.

Plots of Algorithms Performance



Discussion Of Results

We can see from plotting this algorithm's average fitness over generations that it initially rises very steeply from 18 reaching an average fitness of 28 within 10 generations, an increase of almost 1 in each generation. From here it slows down as it attempts to find the optimum, taking another 15 to 20 generations to consistently have an average fitness of 30, this could be due to the low mutation rate and the fact that the less "0"s within the string the slower they will mutate to "1"s and increase, Overall this algorithm found an average optimum fitness of 30 within the population after 30 generations.

1.2 Evolving to find a target string.

Description of aspects in the algorithm

This problem focused on evolving individuals within a population to find a predefined target string consisting of a combination of "1" s and "0" s.

Representation: Again, each population was made up of individual randomly generated binary strings of 0s and 1s, the target string was defined as a string with a combination of 1s and 0s

Fitness Function: The fitness function calculated the average fitness of each population and was again used to assess how the algorithm behaved when evolving to a string of non-optimum fitness.

Selection: My selection for this algorithm was based upon choosing 20 strings that were deemed most like the target string and creating the new population using these.

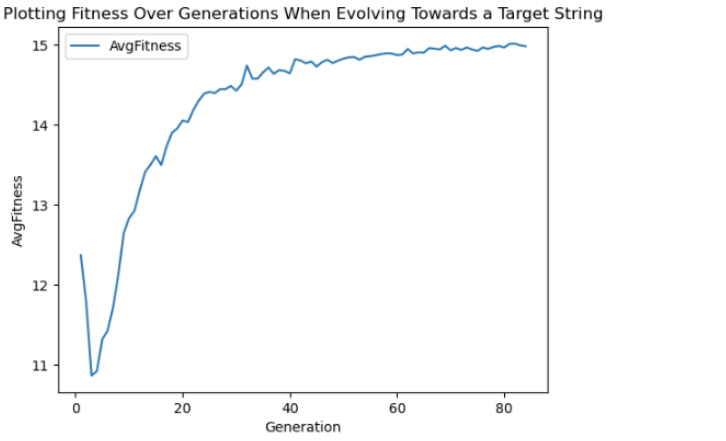
Crossover: The crossover function was identical to that in part 1.1 and created two new strings based on a crossover point and combining the relevant aspects of each parent string

Mutate: this mutate function checks if each bit is a 1 and if it is its preserved again only changing bits that have a value of 0.

Elitism: Elitism was used in this algorithm, identifying 20 strings that were most like the target string and only using these as a pool of parents, meaning that with each improving generation only the 20 most similar strings from that generation were used to create the new population

Plotting the Algorithms Performance

```
Target String Found in generation -  85
String 11010100101100101110101001O110 found matching targetString 11010100101100101110101001O110
```



Discussion Of Results

We can see from the graph of the algorithms performance that initially the average fitness of the population drops as elitism comes into it, removing strings with high fitness as it identifies them as being not similar enough to the target string, we can then see fitness gradually rise as all strings in the population become more similar to the target string again due to this elitism, this continues until the target string is found, the average fitness of the population never rising above the fitness of the target string. In this case the target string is found after 85 generations and both strings are output.

1.3 – Deceptive Problem

This problem consisted of a deceptive solution that should be challenging for the genetic algorithm to find the optimal solution to.

Representation: Again, this population consisted of individuals made up of randomly generated binary strings of 0s and 1s

Fitness Function: The fitness function again counts the number of 1s in each individual and returns this as a measure of the individual's fitness, however it also defines a string completely absent of 1s as the optimal fitness, giving a string of all 0s double that of all 1s

Selection: Selection involves randomly choosing a pair of parents from the population, meaning each generation was inheriting similar fitness scores to the one before it.

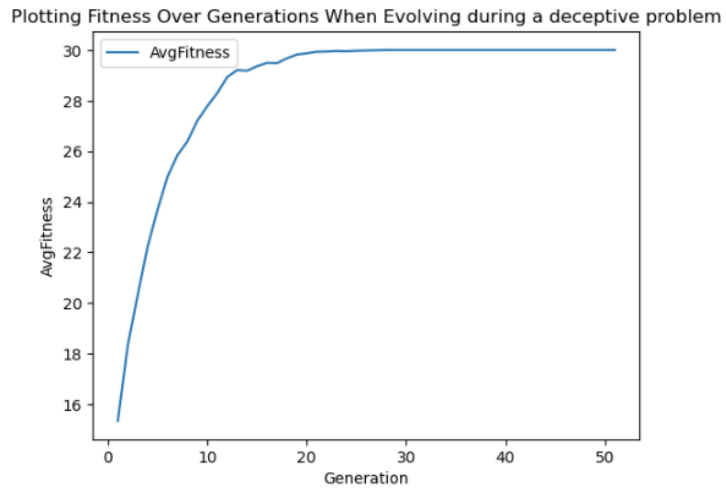
Crossover: Crossover again consisted of choosing a random crossover point and using two parent strings to create two child strings based on these.

Mutate: My mutate function used was the same as the one used in 1.1, checking bits within the string to see if they were 1 and if not flipping them to 1 provided the mutation rate was larger than the randomly generated number

Elitism: Elitism was not implemented in this since taking the fittest individuals in each population would remove any chance of the algorithm solving the deceptive problem based on chance.

Plotting the Algorithms Performance

```
[15.35, 18.33, 20.32, 22.24, 23.7, 25.0, 25.85, 26.39, 27.21, 27.78, 28.3, 28.93, 29.21, 29.18, 29.36, 29.49, 29.48, 29.67, 29.
82, 29.87, 29.93, 29.94, 29.96, 29.95, 29.97, 29.98, 29.99, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 3
0.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0, 30.0]
```



Discussion of Algorithms Results

In this case the algorithm did not find the optimum fitness of all 0s and it continued to gradually move towards populating each individual with all 1s, this could be due to the mutation function not flipping bits of 0 to 1, and is down to the algorithms preconception that the more 1s in a string the better. We can see from the graph that it behaved very similarly to the initial algorithm in 1.1.