CT421 Project 1: Evolutionary Search **Part A: Initial search landscapes**

Cristina Wall (20438522)

Description of code structure:

The program is made up of many functions which are explained below.

“random_string(length)” creates a new random string of binary numbers of a defined length. This allows us to start with a new random string each time the program is run.

“count_fitness(input_string)” calculates the fitness of a string passed in to the function. This function was different for each different task. For the one-max problem, the 1s in the string were counted and this was the fitness. For the deceptive landscape task this was changed so that after the 1s were counted, if the fitness was equal to 0, it would be changed to be twice the length of the string passed in. For the task of finding a target string it was changed more so that you could pass two parameters into the function, the second was the target string. The function would then compare them and count the amount of positions where they were equal to each other.

“single_point_crossover(parent_string1, parent_string2, point)” handles the single point crossover between two strings that are passed in as well as the point (the point is calculated in the “best_crossover_point” function). This function takes the two strings and splits them at the given point and crosses over the two ends of the strings to create two new strings, which are then returned.

“best_crossover_point(string1, string2)” calculates the point where a crossover there would lead to the highest fitness. This function loops through the two strings together and calls the “single_point_crossover” function on each point, and then calculates the fitness of these. The point at which the highest fitness is found is recorded and this is returned.

“mutate_string(string)” allows the strings to be randomly mutated. This function chooses a random element of the string and flips it from 0 to 1 or from 1 to 0.

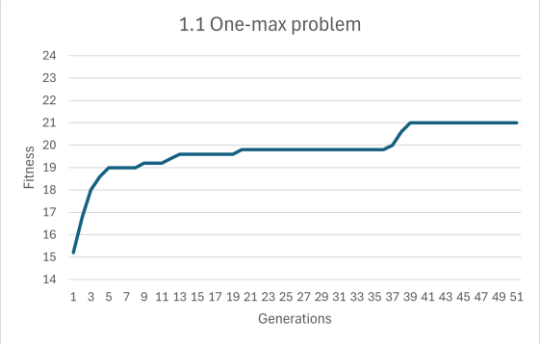
After all the functions a short program sets up the strings and, for each generation, carries out the process of choosing the best crossover point, carrying out the single point crossover, counting the fitness and then mutating for the next generation.

Finally the fitness of the best string in each generation is added to an array and this was used to create the final plots.

Results:

One-max problem:

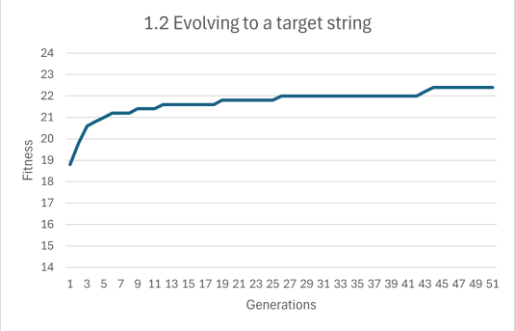
The results show the average fitness at each generation for ten runs of the program. It shows that the fitness started around 15 which is expected, as this is half of the highest possible fitness. However within 50 generations, the



program was not able to reach the target of having all 1s in the string. Even with a test run of 1,000,000 generations (not shown on graph), the target was not achieved. This shows that the genetic algorithm with these parameters is not an efficient way to find the target string.

Evolving to a target string:

This graph also shows the average fitness at each generation for ten runs of the program. The goal of this program was the same as the one-max problem and it had very similar results. It also did not



Deceptive landscape:

This graph also shows the average fitness at each generation for ten runs of the program. The target is not reached again, and the bigger target of all 0s is never found.