**Task 1**

Initially, I wanted to store the dot grid as a single array, but then I chose to store the “dot” grid as a matrix as it was easier to visualize and to solve. Then, I implemented the depth-first search algorithm as we are trying to get the possible combinations. The conditions that check if the path can be further searched is split into different function for easier readability and makes it easier to solve. Once the algorithm is created, all I had to do was create a function named “listPatterns” that checks the first character of the string as well as the third character of the string, as well as using regular expression to find second character in the string.

**Task 2**

Objective 1

At first, I wanted to ensure that the implementation of the genetic algorithm is correct before changing any of the main parameters. If I had started with main parameters first, I could have wasted my time not realizing there is a bug in the sample code. So, I added a number of print functions to ensure that each function perform the operations correctly.

Once I have determined there is no issue with the code itself, I looked into the main parameters and analyzed the issue. It was found that the mutation rate is very high, which means that the original individuals that has the highest fitness evolves after very few generations. These mutated individuals then possibly give lower fitness that the original one, potentially leading to complete elimination of the mutated individuals after few generations. In addition, the crossover rate and elite size is too low, and so the beneficial features aren’t being passed between individuals that could potentially lead to very high fitness. For the number of generations, selection size, population size, they have very low values, which could lead to lack of diversity and hence doesn’t give good results.

Here are the new and old values for each variables:

|  |  |  |
| --- | --- | --- |
| Variable | New Value | Old Value |
| POP\_SIZE | 1000 | 10 |
| SELECTION\_SIZE | 20 | 4 |
| ELITE\_SIZE | 10 | 5 |
| NUM\_GENS | 100 | 20 |
| MUTATION\_RATE | 0.1 | 0.9 |
| CROSSOVER\_RATE | 0.9 | 0.1 |

The best total cost would be RM 117.04.

Objective 2

In order to satisfy the soft constraint of “Satisfy as many demands as possible”, the way the fitness value is calculated needed to be changed to include total demand.

In order to satisfy the soft constraint of “There is only 1 vehicle A and 1 vehicle B”, I needed to change the “cars” variable so that there are only two ids of different car types. I also changed the “num\_cars” variable to 2.

I expected that the code would run without issue, but a “ZeroDivisionError: division by zero” had occurred during the “calc\_fitness” function call that was near the end of the file. I looked into the “best\_individual” variable and it produced the data of the correct format. The “best\_score” variable, however, was zero and prompted me to look into the “individual\_cars” variable. The “individual\_cars” variable did generate some results, but sometimes the values for demand, distance and cost are all zeros (in one of them, never both). But somehow, the total cost, distance and demand is zero for the “sample\_data.json” file.

I reached the conclusion that there was likely no solution found for given vehicle combination for the “sample\_data.json” file.