**Algorithm Used**

I have chosen to use genetic algorithm because I had some experience working with it in a past assignment in previous semester, hence I had some understanding regarding how it works.

**Approach**

The customer (or also known as stop), vehicle, and fitness are separated into classes due to having multiple attributes and methods that are related to them. Using classes is easier to understand each of the component and their functions.

**Classes**

Customer

A computer code on a dark background

Description automatically generated

The customer class is also referred to as the stop that the vehicle needs to pass through. It is given an id, latitude, longitude and demand attributes. The distance function is to calculate its distance relative to another customer or stop. \_\_repr\_\_ function is to represent the instance in a format of (latitude, longitude).

Vehicle

A screen shot of a computer program

Description automatically generated

A vehicle object has the attributes of type (either A or B), list of customers that it stops at, price rate, and capacity. The addCustomer function is to add a customer to the list of customers to serve. The setRateCapacity is to set the price rate and capacity based on its type and is typically called after a vehicle instance is created. The setStartPoint function is to add the depot stop to the routes before any other stops are added.

Fitness

A computer screen shot of a program

Description automatically generated

The Fitness class stores the vehicle that is used to make the delivery, fitness which is the total cost of the route that the vehicle went through, and total distance of the distance travelled to make the delivery. The getDistance function is to get total distance of the route, getCost is to get total cost of the route, and getDemand is to get the total capacity that the vehicle used.

**Generate Population**

**A computer screen shot of a number

Description automatically generated**

**A computer screen shot of a program code

Description automatically generated**

All the customer data are first stored in a csv file, which is then read and each row is inserted as a customer instance into a list called customers.

A screen shot of a computer program

Description automatically generated

A total of 1000 solutions or chromosomes will be created. For each solution, there will be a list of vehicles that contains the customers that they serve. A random vehicle type will be created, and based on that vehicle’s capacity, it will be filled with random customers. When the vehicle’s capacity has reached, another vehicle will be created. Process repeats until all 10 customers have been added to vehicles. All the created vehicles are then added to the solution or chromosome.

A total of 1000 solutions or chromosomes will be created.

**Selection**

**A screen shot of a computer program

Description automatically generated**

The fitness function is to calculate the total cost of the route. The distance function is to calculate the total distance of the route. The getParents function is to get the top two solutions that produced the least costs, which is also the function used for selection.

**Crossover**

A computer screen shot of a program code

Description automatically generated

The findMissing method is used to find missing customer in a list of customers, which will be used later. The convertToTwoDList is to convert the lists of vehicle objects into a nested list of customers, and a list of vehicles. The convertToOneDList function is to convert the nested list of customers to a single list of customers.

A whiteboard with blue writing

Description automatically generated

A computer screen shot of a program code

Description automatically generatedA screenshot of a computer program

Description automatically generated

A random choice will be generated. If the choice of 1 and 2 is generated, crossover occurs, where the first parent will be the main parent in case 1 and second parent will be the main parent in case 2 and a child will be produced. If choice of 3 and 4 is generated, no crossover occurs. The first parent will be taken as the child in case 3, and the second parent will be taken as the child in case 4.

I have developed my own crossover algorithm since this problem is not as straight forward as other genetic algorithm problems due to the constraints and having to take into consideration that the genes cannot be repeated. To deal with this, methods like cycle crossover and partially mapped crossover can be implemented. However, those methods are unnecessarily complex, and I want to avoid using other’s code. Hence, I have came up with a simpler crossover solution.

A whiteboard with numbers and arrows

Description automatically generated

One parent will be the main parent. The parent is a nested list of customers to visit. One random index in the main parent is chosen. One random list inside the nested list of the secondary parent is chosen. The selected list will replace the list in the index position in the main parent. The result list of customers is then converted into one dimensional list.

Then, it will check which customer is not in the list using the findMissing method. A list of missing ids is stored. It will check for duplicated ids in the result list, and replace the second occurrence of the id with one of the missing ids. If all duplicated ids are already replaced and there are still missing ids, the ids will be appended to the result list. The result list is the offspring or child.

**Mutation**

A computer screen shot of a program code

Description automatically generated

A computer screen shot of a code

Description automatically generated

There are two types of mutation: one to mutate the vehicle list, another to mutate the route. For the first mutation, bit flip mutation is applied where vehicle type A can become vehicle type B or the other way round. For the second mutation, swap mutation is applied where two random indexes are selected to swap the elements inside it.

Vehicle mutation has a mutation rate of 50% and route mutation has a mutation rate of 70%.

**Convert back to Class**

Once the child has been mutated, it can be converted back into list of vehicle objects that contains customer objects. This is because we need to use the methods inside those classes to get the distance and costs.

A computer screen shot of a program code

Description automatically generated

Given the two parents, the getOffspring method will call the getMutatedOffspring method, which calls the getCrossoverOffspring method. The code then converts the offspring back into list of objects representations.

When converting back into lists of vehicle objects, vehicle will be created based on the vehicle list. The vehicle will then be filled in order of the customer that is in the one-dimension list until vehicle is full. New vehicle is created and filled until eventually all customer stops are covered.

**Main function**

A screenshot of a computer program

Description automatically generated

Firstly, two best parents are selected from the population generated earlier on. Then for each iteration, two offsprings will be produced from two parents and looped until the new population consists of 1000 offsprings. From the new generation of population, two best solutions are then selected again. These will be the parents for the next iteration.

Stopping condition is the number of iterations and the best solution is stored.

Results are then printed out:

A computer screen shot of text

Description automatically generated

A computer screen shot of a number of text

Description automatically generated