Artificial Intelligence

SOFE 3720U

Assignment 2

Allan Santosh - 100557518
Vicki Derbyshire - 100513868

Introduction:

The purpose of this assignment was to use a genetic algorithm to work on finding a solution to the Travelling Salesman Problem (TSP).

A genetic algorithm implements the following steps to solve a given problem:

1. Initialization — Setting up your problem population
2. Evaluation — Rating each individual of the population by a fitness function
3. Selection — Selecting the “Fittest” members of the population
4. Crossover — Using traits of the most fit members as a model for the next gen
5. Mutation — Adding a level of randomness to our chosen traits for variation
6. Repeat from Step 2

Code:

```
package genetic_algorithm;


public class all_routes {

    create_route[] routes;

    public create_route get_best_route() {

      create_route best_route = routes[0];

        for (int i = 1; i < routes.length; i++) {
            if (best_route.get_fitness_value() <=
routes[i].get_fitness_value()) {
                best_route = routes[i];
            }
        }
        return best_route;
    }


    public all_routes(int num_of_routes, int value) {
        routes = new create_route[num_of_routes];
```

```
        if (value == 1) {

            for (int i = 0; i < routes.length; i++) {
                create_route new_route = new create_route();
                new_route.make_a_route();
                routes[i]= new_route;
            }
        }
    }
}

package genetic_algorithm;


public class all_routes {

    create_route[] routes;

    public create_route get_best_route() {

      create_route best_route = routes[0];

        for (int i = 1; i < routes.length; i++) {
            if (best_route.get_fitness_value() <=
routes[i].get_fitness_value()) {
                best_route = routes[i];
            }
        }
        return best_route;
    }


    public all_routes(int num_of_routes, int value) {
        routes = new create_route[num_of_routes];

        if (value == 1) {

            for (int i = 0; i < routes.length; i++) {
                create_route new_route = new create_route();
                new_route.make_a_route();
                routes[i]= new_route;
            }
        }
```

```
    }
}

package genetic_algorithm;

import java.util.*;

public class create_route{

     public static ArrayList<cities> visiting_cities = new ArrayList
<cities>();
    public ArrayList<cities> routes = new ArrayList<cities>();
    public double fitness_value;
    public int distance_value;


    public create_route(ArrayList<cities> city_route){
        routes = city_route;
    }

    public create_route(){
        for (int i = 0; i < visiting_cities.size(); i++) {
            routes.add(null);
        }
    }

public double distance(){

     if (distance_value == 0) {
           int route_length = 0;

            for (int i=0; i < routes.size(); i++) {

                cities start_city = routes.get(i);
                cities end_city;

                if( i+1 < routes.size()){
                     end_city = routes.get(i+1);
                }
                else{
                     end_city = routes.get(0);
                }
```

```
                route_length =  (int) (route_length +
start_city.calculate_distance_to(end_city));
            }

            distance_value = route_length;
        }
        return distance_value;
    }

    public void make_a_route() {

      for (int i = 0; i < visiting_cities.size(); i++) {
           routes.set(i, visiting_cities.get(i));
           fitness_value = 0;
           distance_value = 0;
        }
    }


    public double get_fitness_value() {
        if (fitness_value == 0) {
            fitness_value =
            1 / distance();
        }
        return fitness_value;
    }



    @Override
    public String toString() {
        String spacer = "\n";
        for (int i = 0; i < routes.size(); i++) {
            spacer += routes.get(i)+"\n";
        }
        return spacer;
    }
}

package genetic_algorithm;

import javax.swing.*;
import java.awt.*;
```

```
import java.util.*;

public class main extends JPanel {

     // Declaring some variables needed and also the city
coordinates

     public static ArrayList<Integer>numbers = new
ArrayList<Integer>();
     public static ArrayList<Integer>numbers2 = new
ArrayList<Integer>();
     public static double distance_check_number = 1000;
     public static int [] coordinates =
     {          60, 200,
                180, 200,
                80, 180,
                140, 180,
                20, 160,
                100, 160,
                200, 160,
                140, 140,
                40, 120,
                100, 120,
                180, 100,
                60, 80,
                120, 80,
                180, 60,
                20, 40,
                100, 40,
                200, 40,
                20, 20,
                60, 20,
                160, 20
     };

     public static cities [] all_cities = new cities[20];
     public static int num_routes = 50;
     public static int status = 1;
     public static int multiplier = 100;

     // Method to evolve the population.

    public static all_routes evaluation(all_routes route) {
```

```
      all_routes all_routes = new all_routes(route.routes.length, 0);
      int offset_value = 1;
      all_routes.routes[0]= route.get_best_route();

      // Doing Crossover over all routes.
      // Using a loop to create new routes from all new routes.

         for (int i = offset_value; i < all_routes.routes.length; i++)
{
             create_route route_a = selection(route);
             create_route route_b = selection(route);
             create_route route_ab = crossover(route_a, route_b);
             all_routes.routes[i]= route_ab;
         }

         // Doing mutation on all routes.

         for (int i = offset_value; i < all_routes.routes.length; i++)
{
             mutation(all_routes.routes[i]);
         }

         return all_routes;
     }

     // Selection for routes for Crossover

     public static create_route selection(all_routes route) {

         all_routes all_routes = new all_routes(5, 0);

         // Get a random route to a city and add it to the existing
route

         for (int i = 0; i < 5; i++) {
             int length = (int) (Math.random() * route.routes.length);
             all_routes.routes[i] = route.routes[length];
         }

         // Get the most efficient route

         create_route best_route = all_routes.get_best_route();
```

```
        return best_route;
    }


    // This method does crossover over 2 routes and produces another


    public static create_route crossover(create_route route_a,
create_route route_b) {

        int initial = (int) (Math.random() * route_a.routes.size());
        int end = (int) (Math.random() * route_b.routes.size());
     create_route route_ab = new create_route();

        for (int i = 0; i < route_ab.routes.size(); i++) {

           // Add the sub route to route_A is initial > end

            if (initial < end && i > initial && i < end) {
                route_ab.routes.set(i, route_a.routes.get(i));
                route_ab.fitness_value = 0;
                route_ab.distance_value = 0;

            }         // else if end bigger than initial then do this

            else if (initial > end) {
                if (!(i < initial && i > end)) {
                     route_ab.routes.set(i, route_a.routes.get(i));
                     route_ab.fitness_value = 0;
                     route_ab.distance_value = 0;
                }
            }
        }


        // Make a loop through route_B


        for (int i = 0; i < route_b.routes.size(); i++) {
            if (!route_ab.routes.contains(route_b.routes.get(i))) {
                for (int j = 0; j < route_ab.routes.size(); j++) {
```

```
                    // Finding if there is another route from route
AB
                    // If it is there then add it.

                   if (route_ab.routes.get(j) == null) {
                    route_ab.routes.set(j, route_b.routes.get(i));
                    route_ab.fitness_value = 0;
                    route_ab.distance_value = 0;
                       break;
                   }
               }
           }
       }
       return route_ab;
   }

   // This method mutates the routes

   public static void mutation(create_route route) {

       for(int i=0; i < route.routes.size(); i++){

          // Mutation Rate

           if(Math.random() < 0.015){

               // Take the cities and swap them

               int j = (int) (route.routes.size() * Math.random());
               cities first = route.routes.get(i);
               cities second = route.routes.get(j);
               route.routes.set(j, first);
               route.fitness_value = 0;
               route.distance_value = 0;
               route.routes.set(i, second);
               route.fitness_value = 0;
               route.distance_value = 0;
           }
       }
   }


    // The main function
```

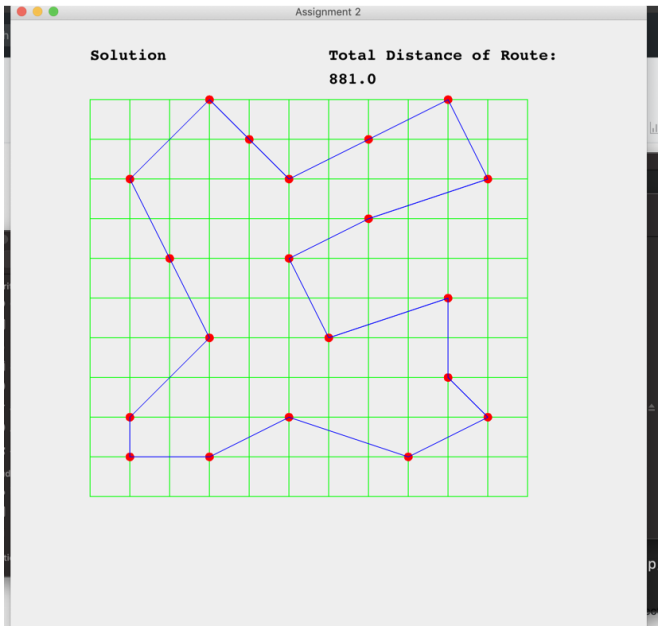
```
    public static void main(String[] args) {

      main assignment2 = new main();

    // Add all cities to cities class.

      for (int i = 0, x = 0; i < 20; i++) {
           all_cities[i] = new
cities(coordinates[x],coordinates[x+1]);
           x = x + 2;
           create_route.visiting_cities.add(all_cities[i]);
      }

   // Starting the genetic algorithm.

        all_routes all_routes = new all_routes(num_routes, status);

        all_routes = evaluation(all_routes);
        for (int i = 0; i < multiplier; i++) {
           all_routes = evaluation(all_routes);
        }

   // Try to find the optimal solution.

           numbers2.clear();
           distance_check_number =
all_routes.get_best_route().distance();
           System.out.print(all_routes.get_best_route());

   //Plotting solution on Java.

           JFrame window = new JFrame("Assignment 2");

        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setVisible(true);
        window.setSize(800,800);
        window.add(assignment2);

    }

    // This function uses the coordinates to plot.
```

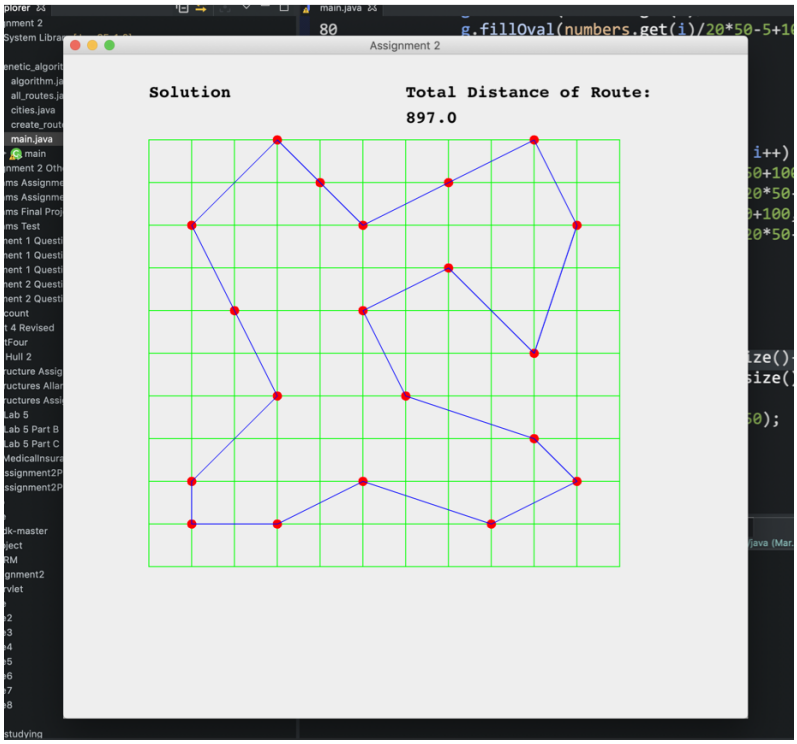
```
    public void paint (Graphics g) {

            g.setFont(new Font("Courier", Font.BOLD,20));
            g.setColor(Color.black);
            g.drawString("Solution",100,50);
            g.drawString("Total Distance of Route: ", 400, 50);

g.drawString(Double.toString(distance_check_number),400,80);
            g.setColor(Color.green);

            for(int i=0; i<11; i++) {
            g.drawLine(100,100+i*50,100+550,100+i*50);
            }
            for(int i=0; i<12; i++) {
            g.drawLine(100+i*50,100,100+i*50,100+500);
            }

            g.setColor(Color.red);
            for (int i=0; i<numbers.size(); i++) {
                 g.drawOval(numbers.get(i)/20*50-5+100,
650-(numbers.get(i+1)/20*50)-5-50, 10, 10);
                 g.fillOval(numbers.get(i)/20*50-5+100,
650-(numbers.get(i+1)/20*50)-5-50, 10, 10);
                 i++;
            }

            g.setColor(Color.blue);

            for (int i=0; i<numbers2.size()-2; i++) {
                 g.drawLine(numbers2.get(i)/20*50+100,
                           650-numbers2.get(i+1)/20*50-50,
                           numbers2.get(i+2)/20*50+100,
                           650-numbers2.get(i+3)/20*50-50);
                 i++;
            }



            g.drawLine(numbers2.get(numbers2.size()-2)/20*50+100,
                      650-numbers2.get(numbers2.size()-1)/20*50-50,
                      numbers2.get(0)/20*50+100,
                      650-numbers2.get(1)/20*50-50);
```

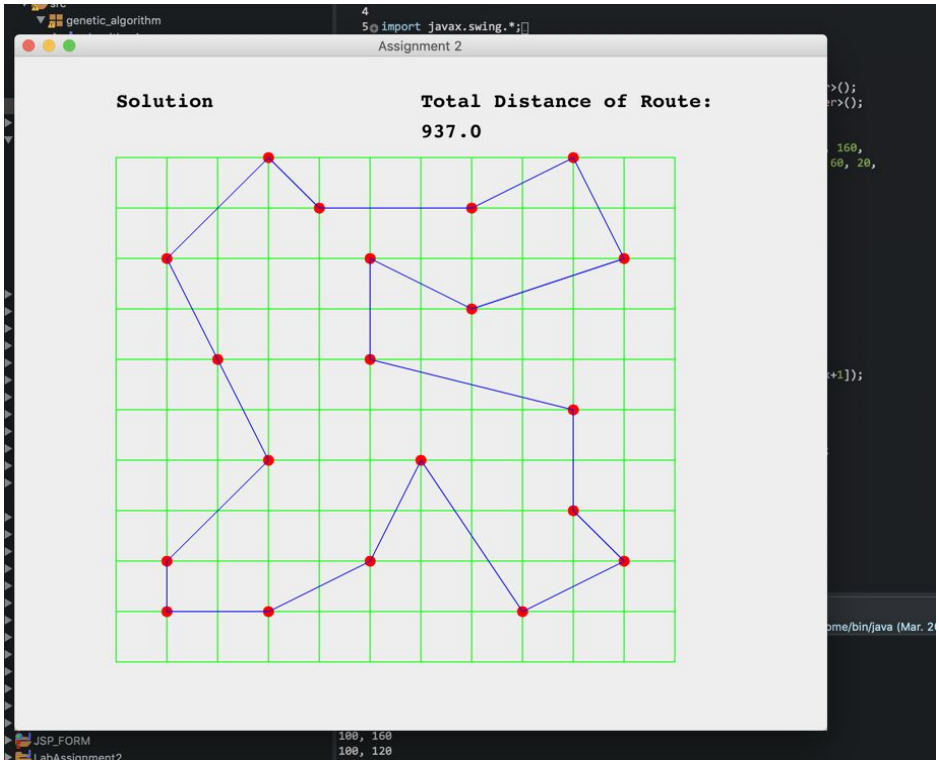
```
} }
```

Screen Shots:

Part 1




Assignment 2
Solution
Total Distance of Route:
897.0

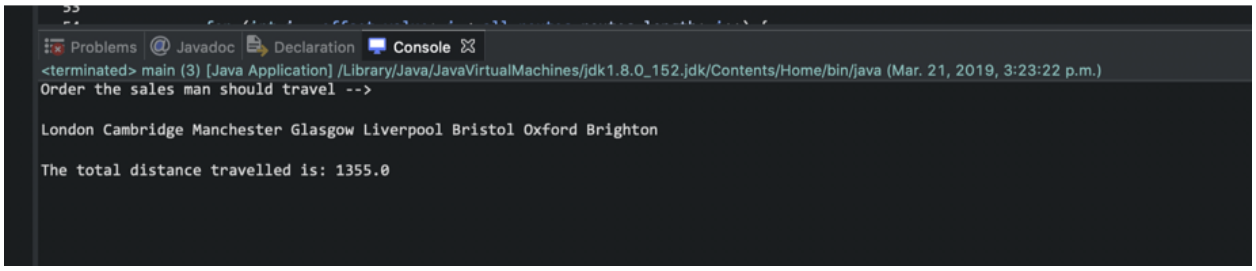

Assignment 2
Solution
Total Distance of Route:
937.0

Part 2

```
51                  all_routes.routes[i]= route_ab;
52              }
53
```

Problems Javadoc Declaration Console

```
<terminated> main (3) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java (Mar. 21, 2019, 3:22:03 p.m.)
Order the sales man should travel -->

Bristol Liverpool Glasgow Manchester Oxford Cambridge London Brighton

The total distance travelled is: 1404.0
```



Genetic Algorithm

Initialization :

A random sized population was created and each individual in the population is a random route from the initial city to the end of the route visiting all other cities.

Evaluation:

After the population is generated, each route is then evaluated using fitness. The fitness is the inverse of the route distance and the only way to get a minimum distance is to have a larger fitness value. All the routes are sorted based on the fitness value of each route.

Selection:

In the process of selection, the worse routes are disregarded and only the good routes remain. The purpose of this method is to make sure that only the fittest route is selected for the next generation.

Crossover:

Create new individuals from combining the selected individuals. The goal is that the combined individual will inherit the traits from the selected individual.

Mutation:

By making small changes in the randomness of the populations genetics, we can ensure that a route is there where every location is included at least once and only once.

Swap mutation was used where two locations in the route were selected at random and their positions swapped.

Terminating Condition:

The program terminates once it is done looping through the genetic algorithm a number of times dictated by the variable *multiplier*. Upon completion of this, the best route is chosen and plotted.

Trial 1 Distance	Trial 2 Distance	Trial 3 Distance
881	897	937

For each run the value of the distance changes based on which the algorithm thinks is the best and most fittest route.