Assignment 6 –

package Practical\_Exam;

import java.util.\*;

public class TSPGeneticAlgorithm2 {

private static final int NUM\_CITIES = 4;

private static final int POPULATION\_SIZE = 50;

private static final int NUM\_GENERATIONS = 1000;

private static final double MUTATION\_RATE = 0.05;

// Distances between cities represented as an adjacency matrix

private static final int[][] distances = {

{0,10,15,20},

{5,0,9,10},

{6,13,0,12},

{8,9,9,0},

};

private void solveTSP() {

List<int[]> population = initPopulation(POPULATION\_SIZE);

for (int i = 0; i < NUM\_GENERATIONS; i++) {

population = evolve(population);

}

int[] bestRoute = getBestRoute(population);

System.out.println("Best Route: " + Arrays.toString(bestRoute));

System.out.println("Distance of Best Route: " + calculateDistance(bestRoute));

}

private List<int[]> initPopulation(int populationSize) {

List<int[]> population = new ArrayList<>();

for (int i = 0; i < populationSize; i++) {

int[] route = getRandomRoute();

population.add(route);

}

return population;

}

private int[] getRandomRoute() {

List<Integer> citiesList = new ArrayList<>();

for (int i = 0; i < NUM\_CITIES; i++) {

citiesList.add(i);

}

Collections.shuffle(citiesList);

return citiesList.stream().mapToInt(i -> i).toArray();

}

private List<int[]> evolve(List<int[]> population) {

List<int[]> newPopulation = new ArrayList<>();

while (newPopulation.size() < POPULATION\_SIZE) {

int[] parent1 = selectParent(population);

int[] parent2 = selectParent(population);

int[] child = crossover(parent1, parent2);

if (Math.random() < MUTATION\_RATE) {

mutate(child);

}

newPopulation.add(child);

}

return newPopulation;

}

private int[] selectParent(List<int[]> population) {

Random rand = new Random();

int tournamentSize = 5;

int[] bestRoute = null;

double bestFitness = Double.MAX\_VALUE;

for (int i = 0; i < tournamentSize; i++) {

int[] candidate = population.get(rand.nextInt(population.size()));

double fitness = calculateDistance(candidate);

if (fitness < bestFitness) {

bestFitness = fitness;

bestRoute = candidate;

}

}

return bestRoute;

}

private int[] crossover(int[] parent1, int[] parent2) {

Random r = new Random();

int startPos = r.nextInt(parent1.length);

// System.out.println(startPos);

int endPos = r.nextInt(parent1.length);

int[] child = new int[parent1.length];

Arrays.fill(child, -1);

if(endPos < startPos) {

int temp = endPos;

endPos = startPos;

startPos = temp;

}

for (int i = startPos; i <= endPos; i++) {

child[i] = parent1[i];

}

for (int i = 0; i < parent2.length; i++) {

if (!containsCity(child, parent2[i])) {

for (int j = 0; j < child.length; j++) {

if (child[j] == -1) {

child[j] = parent2[i];

break;

}

}

}

}

return child;

}

private void mutate(int[] route) {

int idx1 = (int) (route.length \* Math.random());

int idx2 = (int) (route.length \* Math.random());

int temp = route[idx1];

route[idx1] = route[idx2];

route[idx2] = temp;

}

private double calculateDistance(int[] route) {

double distance = 0;

for (int i = 0; i < route.length - 1; i++) {

int city1 = route[i];

int city2 = route[i + 1];

distance += distances[city1][city2];

}

distance += distances[route[route.length - 1]][route[0]];

return distance;

}

private boolean containsCity(int[] route, int city) {

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

if (route[i] == city) {

return true;

}

}

return false;

}

private int[] getBestRoute(List<int[]> population) {

int[] bestRoute = population.get(0);

double bestFitness = calculateDistance(bestRoute);

for (int[] route : population) {

double fitness = calculateDistance(route);

if (fitness < bestFitness) {

bestFitness = fitness;

bestRoute = route;

}

}

return bestRoute;

}

public static void main(String[] args) {

TSPGeneticAlgorithm2 tspGA = new TSPGeneticAlgorithm2();

tspGA.solveTSP();

}

}