Genetic Algorithm

import java.util.\*;

public class GeneticTSP {

// Parameters

static final int POP\_SIZE = 50;

static final int GENS = 500;

static final double CROSS\_RATE = 0.8;

static final double MUT\_RATE = 0.2;

static final int NUM\_CITIES = 5;

static final double[][] cities = {

{0, 0}, {1, 5}, {5, 2}, {6, 6}, {8, 3}

};

static Random rand = new Random();

// Individual representation

static class Individual {

int[] route;

double fitness;

Individual(int[] route) {

this.route = route.clone();

this.fitness = evaluate(route);

}

}

// Euclidean distance

static double distance(double[] c1, double[] c2) {

return Math.sqrt(Math.pow(c1[0]-c2[0], 2) + Math.pow(c1[1]-c2[1], 2));

}

// Total distance of a route

static double totalDistance(int[] route) {

double dist = 0;

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

dist += distance(cities[route[i]], cities[route[(i+1)%NUM\_CITIES]]);

}

return dist;

}

// Fitness function

static double evaluate(int[] route) {

return 1.0 / totalDistance(route);

}

// Initialize population

static List<Individual> initPopulation() {

List<Individual> population = new ArrayList<>();

int[] base = new int[NUM\_CITIES];

for(int i=0;i<NUM\_CITIES;i++) base[i]=i;

for(int i=0;i<POP\_SIZE;i++) {

int[] route = base.clone();

shuffle(route);

population.add(new Individual(route));

}

return population;

}

// Shuffle array

static void shuffle(int[] array) {

for(int i=array.length-1;i>0;i--) {

int j = rand.nextInt(i+1);

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

// Tournament selection

static Individual select(List<Individual> population) {

int tournamentSize = 5;

Individual best = null;

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

Individual ind = population.get(rand.nextInt(POP\_SIZE));

if(best==null || ind.fitness > best.fitness) best = ind;

}

return best;

}

// Ordered Crossover (OX)

static Individual[] crossover(Individual p1, Individual p2) {

if(rand.nextDouble() > CROSS\_RATE) {

return new Individual[]{ new Individual(p1.route), new Individual(p2.route) };

}

int start = rand.nextInt(NUM\_CITIES);

int end = rand.nextInt(NUM\_CITIES);

if(start > end) { int tmp=start; start=end; end=tmp; }

int[] c1 = new int[NUM\_CITIES];

Arrays.fill(c1, -1);

int[] c2 = new int[NUM\_CITIES];

Arrays.fill(c2, -1);

// Copy slice

for(int i=start;i<end;i++) { c1[i]=p1.route[i]; c2[i]=p2.route[i]; }

// Fill remaining

fillRemaining(c1, p2.route, end);

fillRemaining(c2, p1.route, end);

return new Individual[]{ new Individual(c1), new Individual(c2) };

}

static void fillRemaining(int[] child, int[] parent, int start) {

int idx = start;

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

int city = parent[i];

if(!contains(child, city)) {

while(child[idx%NUM\_CITIES]!=-1) idx++;

child[idx%NUM\_CITIES]=city;

}

}

}

static boolean contains(int[] arr, int val) {

for(int v: arr) if(v==val) return true;

return false;

}

// Swap Mutation

static void mutate(Individual ind) {

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

if(rand.nextDouble()<MUT\_RATE) {

int j = rand.nextInt(NUM\_CITIES);

int temp = ind.route[i];

ind.route[i] = ind.route[j];

ind.route[j] = temp;

}

}

ind.fitness = evaluate(ind.route);

}

public static void main(String[] args) {

List<Individual> population = initPopulation();

Individual best = null;

for(int gen=0;gen<GENS;gen++) {

List<Individual> newPop = new ArrayList<>();

// Track best

for(Individual ind: population) {

if(best==null || ind.fitness>best.fitness) best=ind;

}

// Generate new population

while(newPop.size()<POP\_SIZE) {

Individual p1 = select(population);

Individual p2 = select(population);

Individual[] offspring = crossover(p1, p2);

mutate(offspring[0]);

mutate(offspring[1]);

newPop.add(offspring[0]);

if(newPop.size()<POP\_SIZE) newPop.add(offspring[1]);

}

population = newPop;

}

// Output best

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

System.out.println("Shortest Distance: " + totalDistance(best.route));

}

}

Output:

