Source code

```
import java.util.Random;
/**
 * Created by yauheni on 05.10.16.
 */
public class Point {
	int lable;
	double x;
	double y;
	// Constructs a randomly placed city
	public Point(){
		Random random = new Random();
		this.x = (random.nextInt(100) / 10.);
		this.y = (random.nextInt(100) / 10.);
	}
	public int getLable() {
		return lable;
	}
	public void setLable(int lable) {
		this.lable = lable;
	}
	public Point(double x, double y){
		this.x = x;
		this.y = y;
	}
	public double getX(){
		return this.x;
	}
	public double getY(){
		return this.y;
	}
	public double distance(Point point){
		double xDistance = Math.abs(getX() - point.getX());
		double yDistance = Math.abs(getY() - point.getY());
		double distance = Math.sqrt( (xDistance*xDistance) + (yDistance*yDistance) );
		return distance;
	}
	@Override
	public String toString(){
		return lable + ": (" +getX()+", "+getY()+")";
	}
}


import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
/**
 * Created by yauheni on 05.10.16.
 */
public class Route implements Comparable<Route>{
	List<Point> route = new ArrayList<>();
	public Route(List<Point> points) {
		ArrayList<Point> newPoints = new ArrayList<>();
		for (int i = 1; i < points.size(); i++) {
			newPoints.add(points.get(i));
		}
```

```
            Collections.shuffle(newPoints);
            route = new ArrayList<>(newPoints);
            Collections.reverse(route);
            route.add(points.get(0));
            Collections.reverse(route);
            route.add(points.get(0));
      }
      public List<Point> getRoute() {
            return route;
      }
      public void setRoute(List<Point> route) {
            this.route = route;
      }
      public double getRouteDistance() {
            double distance = 0.0;
            for (int i = 0; i < route.size() - 1; i++) {
                  distance += route.get(i).distance(route.get(i + 1));
            }
            return distance;
      }
      public String printRoute() {
            String str= "";
            for (int i = 0; i < route.size(); i++) {
                  str += route.get(i).getLable() + " -> ";
            }
            return str;
      }
      @Override
      public int compareTo(Route o) {
            int i = 0;
            return (o.getRouteDistance() > getRouteDistance()) ? 1 : (o.getRouteDistance() ==
getRouteDistance()) ? 0 : -1;
      }
}
```

```
import java.util.ArrayList;
import java.util.Formatter;
import java.util.List;
import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;
/**
 * Created by yauheni on 21.09.16.
 */
public class Test {
      public static void main(String[] args) {
            List<Point> points = new ArrayList<Point>();
            for (int i = 0; i < 20; i++) {
                  Point tempP = new Point();
                  tempP.setLable(i);
                  points.add(tempP);
            }
            for (int i = 0; i < 20; i++) {
                  for (int j = 0; j < 20; j++) {
                        System.out.printf("%.3f ",points.get(i).distance(points.get(j)));
                  }
                  System.out.println();
```

```
		}
		Population pop = new Population(points);
		int count = 0;
		while(true) {
			count++;
			if(count == 1000)
				break;
			System.out.println(pop.getFittestIndividual().getRouteDistance());
			System.out.println(pop.averageFitness());
			System.out.println(pop.getFittestIndividual().printRoute());
			System.out.println("======================");
			List<Route> newIndividuals = new ArrayList<>();
			pop.getFittestIndividual();
			List<Route> routes = pop.getIndividuals();
			for (int i = 0; i < 50; i++) {
				int first = ThreadLocalRandom.current().nextInt(0, 50);
				int second = ThreadLocalRandom.current().nextInt(0, 50);
				int razrez = ThreadLocalRandom.current().nextInt(2, 19);
				List<Point> firstInd = routes.get(first).getRoute();
				List<Point> secondInd = routes.get(second).getRoute();
				List<Point> newFirst = new ArrayList<>(firstInd);
				List<Point> newSecond = new ArrayList<>(secondInd);
				List<Point> pointsMockForFirst = copyPoints(points);
				pointsMockForFirst.remove(0);
				List<Point> pointsMockForSecond = copyPoints(points);
				pointsMockForSecond.remove(0);
				for (int j = 1; j < 20; j++) {
					if(j <= (razrez + 1)) {
						newFirst.set(j, getNewPoint(pointsMockForFirst,
firstInd.get(j).getLable()));
						newSecond.set(j, getNewPoint(pointsMockForSecond,
secondInd.get(j).getLable()));
					} else {
						newFirst.set(j, getNewPoint(pointsMockForFirst,
secondInd.get(j).getLable()));
						newSecond.set(j, getNewPoint(pointsMockForSecond,
firstInd.get(j).getLable()));
					}
				}
				Route r1 = new Route(points);
				r1.setRoute(newFirst);
				Route r2 = new Route(points);
				r2.setRoute(newSecond);
				newIndividuals.add(r1);
				newIndividuals.add(r2);
			}
			Population newPopulation = new Population(points);
			newPopulation.setIndividuals(newIndividuals);
			pop = newPopulation;
		}
	}
	public static List<Point> copyPoints(List<Point> points) {
		List<Point> newPoints = new ArrayList<>();
		for (int i = 0; i < points.size(); i++) {
			newPoints.add(points.get(i));
		}
		return newPoints;
	}
	public static Point getNewPoint(List<Point> points, int oldPointLabel) {
```

```
            if((points.size()- 1) == oldPointLabel) {
                  Point p = points.get((points.size()- 1));
                  points.remove(0);
                  return p;
            }
            if(points.size() > 0) {
                  Point p = points.get(oldPointLabel % points.size());
                  points.remove(oldPointLabel % points.size());
                  return p;
            } else {
                  Point p = points.get(0);
                  points.remove(0);
                  return p;
            }
      }
}
```

Matrix of distances between cities

0.000	4.617	2.302	2.025	6.935	6.294	5.787	6.454	3.289	7.350	10.253	5.924	9.153	8.249	10.678	6.428	8.580	3.900	7.892	9.552
4.617	0.000	4.528	6.075	9.742	5.900	3.384	9.640	2.818	7.506	9.139	3.764	7.245	6.217	9.458	9.265	6.964	1.487	5.270	9.613
2.302	4.528	0.000	2.000	5.385	4.046	4.201	5.162	1.970	5.048	7.998	4.211	7.066	6.239	8.429	4.885	6.450	3.220	6.100	7.256
2.025	6.075	2.000	0.000	5.000	5.725	6.201	4.455	3.883	6.378	9.569	6.207	8.896	8.132	10.012	4.501	8.246	4.977	8.075	8.429
6.935	9.742	5.385	5.000	0.000	5.636	8.028	0.922	6.934	4.866	7.910	7.780	8.493	8.254	8.322	0.510	7.810	8.293	8.900	5.749
6.294	5.900	4.046	5.725	5.636	0.000	2.915	6.073	3.640	1.628	3.960	2.550	3.256	2.717	4.386	5.349	2.571	4.528	3.265	3.734
5.787	3.384	4.201	6.201	8.028	2.915	0.000	8.238	2.500	4.522	5.757	0.400	3.912	2.884	6.075	7.640	3.585	2.408	2.163	6.428
6.454	9.640	5.162	4.455	0.922	6.073	8.238	0.000	6.877	5.504	8.658	8.026	9.080	8.756	9.080	0.806	8.386	8.233	9.305	6.580
3.289	2.818	1.970	3.883	6.934	3.640	2.500	6.877	0.000	5.099	7.433	2.640	6.021	5.061	7.824	6.464	5.517	1.360	4.610	7.325
7.350	7.506	5.048	6.378	4.866	1.628	4.522	5.504	5.099	0.000	3.324	4.144	3.640	3.551	3.770	4.710	2.973	6.103	4.455	2.247
10.253	9.139	7.998	9.569	7.910	3.960	5.757	8.658	7.433	3.324	0.000	5.376	2.236	3.102	0.447	7.852	2.220	8.011	4.301	2.470
5.924	3.764	4.211	6.207	7.780	2.550	0.400	8.026	2.640	4.144	5.376	0.000	3.581	2.561	5.701	7.406	3.220	2.720	1.970	6.030
9.153	7.245	7.066	8.896	8.493	3.256	3.912	9.080	6.021	3.640	2.236	3.581	0.000	1.030	2.400	8.302	0.700	6.301	2.121	4.111
8.249	6.217	6.239	8.132	8.254	2.717	2.884	8.756	5.061	3.551	3.102	2.561	1.030	0.000	3.338	8.008	0.922	5.280	1.217	4.565
10.678	9.458	8.429	10.012	8.322	4.386	6.075	9.080	7.824	3.770	0.447	5.701	2.400	3.338	0.000	8.273	2.500	8.363	4.510	2.789
6.428	9.265	4.885	4.501	0.510	5.349	7.640	0.806	6.464	4.710	7.852	7.406	8.302	8.008	8.273	0.000	7.612	7.824	8.601	5.787
8.580	6.964	6.450	8.246	7.810	2.571	3.585	8.386	5.517	2.973	2.220	3.220	0.700	0.922	2.500	7.612	0.000	5.917	2.138	3.689
3.900	1.487	3.220	4.977	8.293	4.528	2.408	8.233	1.360	6.103	8.011	2.720	6.301	5.280	8.363	7.824	5.917	0.000	4.534	8.261
7.892	5.270	6.100	8.075	8.900	3.265	2.163	9.305	4.610	4.455	4.301	1.970	2.121	1.217	4.510	8.601	2.138	4.534	0.000	5.727
9.552	9.613	7.256	8.429	5.749	3.734	6.428	6.580	7.325	2.247	2.470	6.030	4.111	4.565	2.789	5.787	3.689	8.261	5.727	0.000