

INFO 6205 Spring 2023 Project

(SECTION - 03)

THE TRAVELLING SALESMAN PROBLEM



Group Members:

Shobhit Srivastava - 002965382

Tiyasha Sen - 002727486

Abhay Deshpande - 002775776

Introduction

Aim:

The aim of the project is to develop a solution for the Traveling Salesman Problem (TSP) using the Christofides algorithm and various optimization techniques. The TSP is a difficult problem in which one must find the shortest tour of n cities or points in a two-dimensional space. This problem is classified as NP-hard, which means that deterministic solutions have complexity $O(k!)$. The proposed solution will utilize the Christofides algorithm as a baseline approach and then use tactical optimization techniques such as random swapping, 2-opt, and 3-opt improvement to refine the solution. Additionally, strategic optimization methods such as simulated annealing, ant colony optimization, genetic algorithms, etc. will be utilized to further optimize the solution. The goal of the project is to find the most optimal solution to the TSP problem.

Approach:

The approach of the project is to develop a solution for the Traveling Salesman Problem (TSP) using the Christofides algorithm as the initial approach. The Christofides algorithm is a heuristic algorithm that provides an approximate solution to the TSP. After obtaining a solution with the Christofides algorithm, tactical optimization techniques such as random swapping, 3-opt improvement will be used to refine the solution further. These techniques involve systematically swapping cities in the solution to try to find a shorter route. Additionally, strategic optimization methods such as simulated annealing, ant colony optimization will be utilized to further optimize the solution. These methods involve exploring the search space to find the best possible solution. The goal of the project is to find the most optimal solution to the TSP problem using the optimization algorithms.

Program:

Data Structures & Algorithms:

Data Structures: Array, ArrayList, Collections, HashSet

Classes: The algorithms has been segregated to different classes as mentioned below:

CrimeDataProcessor.java, CrimePoint.java,
ntOptimization.java, OptimizeThreeOpt.java, OptimizeRandomSwapping.java,

Main.java, SimmulatedAnnealing.java, AntOptimization.java, Driver.java

Test Classes: CrimeDataProcessorTest.java, CrimePointTest.java, AntOptimizationTest.java, OptimizeThreeOptTest.java, OptimizeRandomSwappingTest.java, SimmulatedAnnealingTest.java, AntOptimizationTest.java, Driver.java

Algorithms:

The algorithms we used to optimize the Travelling Salesman problem:

Christofides Algorithm:

Christofides Algorithm is an approximate algorithm that guarantees that the solution will be within a factor of $3/2$ of the optimal solution length.

On average : $\text{Christofides} / (1\text{-Tree lower bound}) = 1.1$

Worst case : $\text{Christofides} / \text{Optimal TSP} = 1.5$

Now, that we have a defined solution to get reasonable solution to TSP, can we have any method to make modifications and generate a better solution?

Yes, there are several transformations we can make to see if we can improve upon the solution. These are the improvements listed below:

- Three-opt improvement : The 3-opt algorithm is a heuristic approach for solving optimization problems, commonly used for the traveling salesman problem (TSP) and related routing problems. The TSP involves finding the shortest possible route that visits a set of cities and returns to the starting point.
- Simulated annealing : Simulated annealing is a probabilistic optimization algorithm that is used to find the global minimum of a complex function. It is inspired by the physical process of annealing in metallurgy, where a material is heated and then slowly cooled to increase its durability and reduce defects.
- Ant colony optimization : Ant colony optimization is a metaheuristic optimization algorithm that is inspired by the behavior of real ants. The algorithm is based on the observation that ants can find the shortest path between their nest and a food source by leaving a trail of pheromones on the ground. Other ants are then able to follow this trail, reinforcing it and making it stronger over time.

- Random swapping : Random swapping is a simple technique used in various optimization algorithms to improve the quality of a given solution. This technique involves randomly selecting two elements or components of a solution and swapping them. This can be done multiple times to create a set of new candidate solutions, with the hope that one of them will lead to an improved solution.

Invariants:

OptimizeThreeOpt.java

- The input `double[][] dist` is a two-dimensional array representing the distance matrix between each pair of vertices.
- The input `List<Integer> tour` is a permutation of 0 to $n-1$ where n is the number of vertices in the graph.
- The method `tourLength` computes the total length of the given tour based on the distance matrix `distanceMatrix`.
- The method `reverse` takes four indices i, j, k , and l such that $i < j < k < l < n$, where n is the length of `tour`.

OptimizeRandomSwapping.java

- The `randomSwapOptimization` method takes in a `List<Integer>` called `tour`, a `double[][]` called `distanceMatrix`, and an `int` called `iterations`.
- The method creates a new `ArrayList<Integer>` called `optimizedTour` and initializes it with the same values as `tour`.
- The method initializes a `double` called `currentTourLength` with the length of the initial tour.
- After every iteration of the loop, `optimizedTour` always represents a valid tour of cities, i.e., it contains all the cities from the original tour and has no duplicates.
- The `tourLength` method calculates the length of a tour represented by a `List<Integer>` called `tour` and a `double[][]` called `distanceMatrix`.

SimulatedAnnealing.java

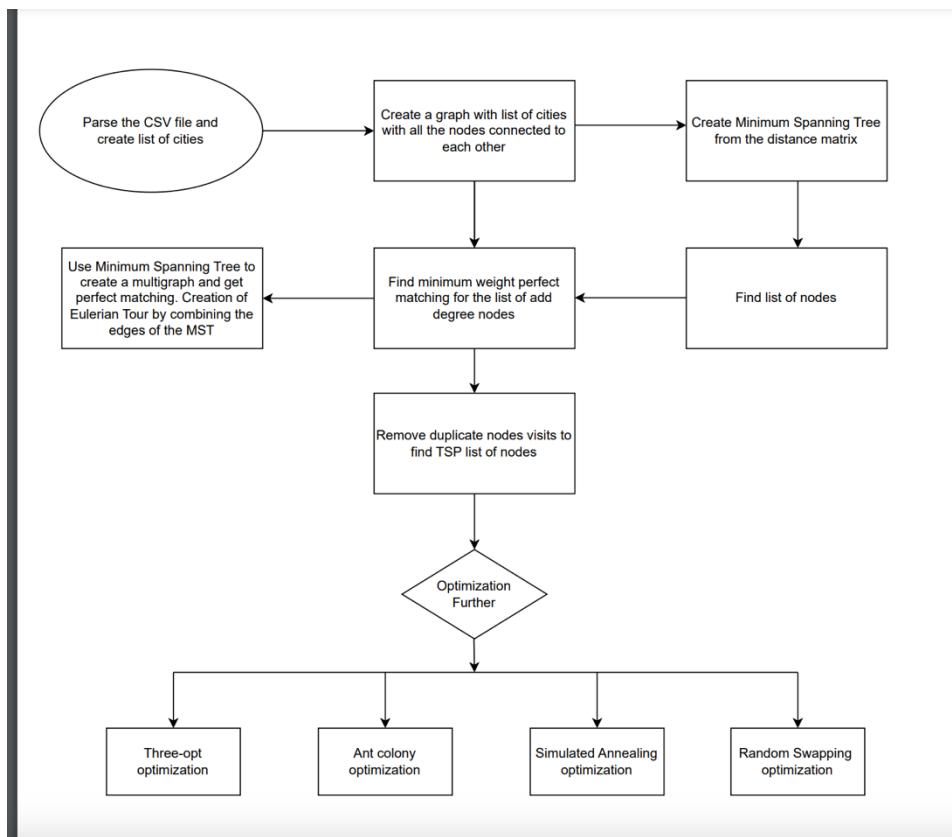
- The temperature T is initialized to 1.0 and decreases with each iteration according to the `coolingRate` parameter, which is set to 0.99995 by default.
- The `currentCost` and `bestCost` variables represent the total length of the `currentSolution` and `bestSolution` tours, respectively.

- The method `swapEdges()` randomly selects two distinct indices i and j in the tour list and reverses the order of the elements in the sublist from i to $j-1$. The resulting tour is guaranteed to have the same cities as the input tour but with a different ordering.
- The `absoluteTemperature` parameter is set to a very small value, indicating that the algorithm will stop when the temperature drops below that value.

AntOptimization.java

- The `buildAntTour` method ensures that each city is visited only once by maintaining a boolean array of visited cities.
- The `optimizeWithAntColony` method updates the best tour and its cost when a new solution is found with a lower cost.
- The `buildAntTour` method uses a probabilistic approach to choose the next city, based on the pheromone and visibility matrices and a random number.

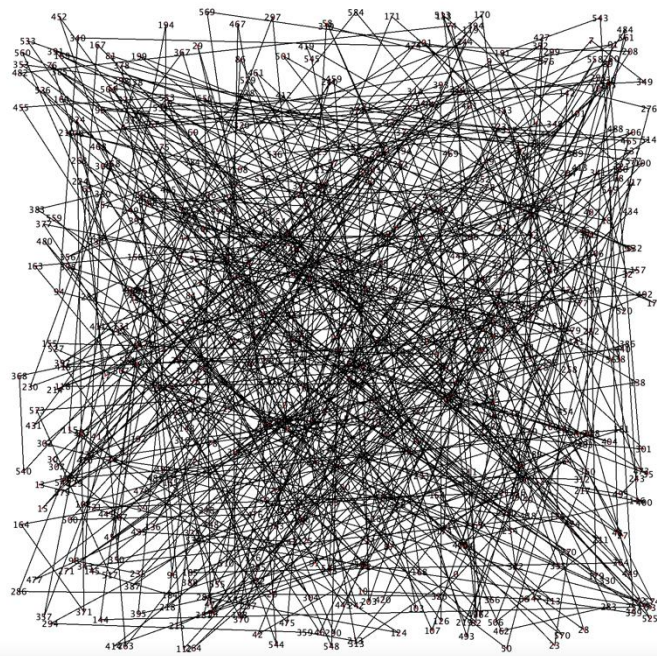
Flow Charts (inc. UI Flow):



Observations & Graphical Analysis:

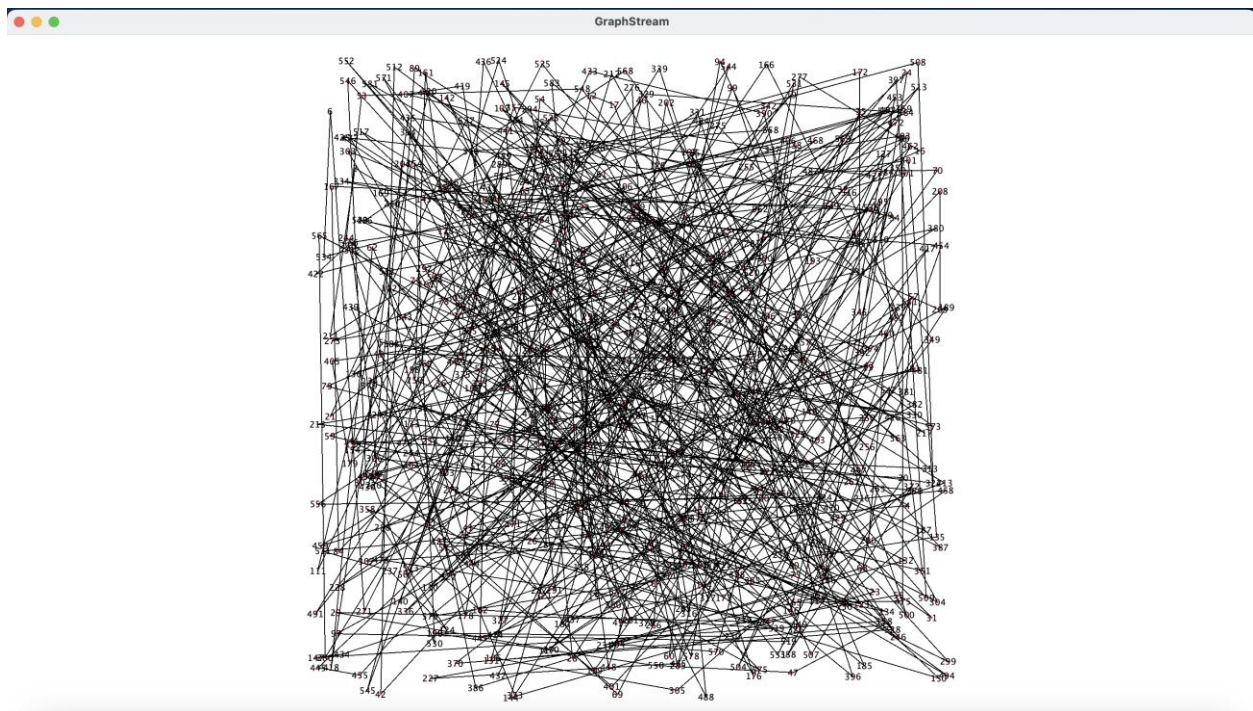
Three OPT:

If the new tour has a shorter distance, the edges are reversed and the tour is updated. This process is repeated until no further improvements can be made. The "reverse" method is used to reverse a sequence of edges in the tour, and the "calculateDistance" method is used to calculate the total distance of a tour. The "tourLength" method is also provided to calculate the total length of a tour.



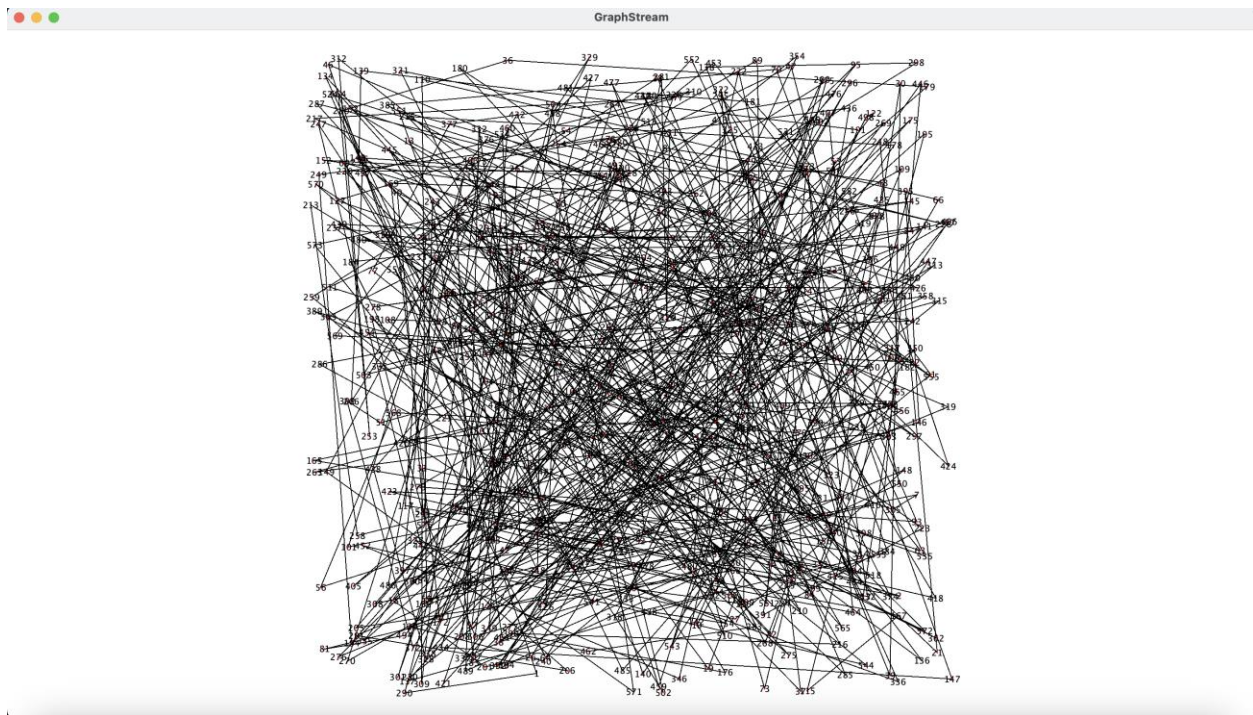
Ant Colony:

The code above implements the Ant Colony Optimization (ACO) algorithm to solve the Traveling Salesman Problem (TSP). The ACO algorithm is a metaheuristic that imitates the behavior of ants to find the shortest path between two points. The algorithm starts by initializing the pheromone and visibility matrices, which store the pheromone and distance information, respectively, for each edge in the graph. Then, a number of ants are randomly placed in the graph, and each ant constructs a path by choosing the next node to visit based on the pheromone and visibility information. This way, over time, the pheromone information becomes concentrated in the best paths, which are more likely to be chosen by future ants.



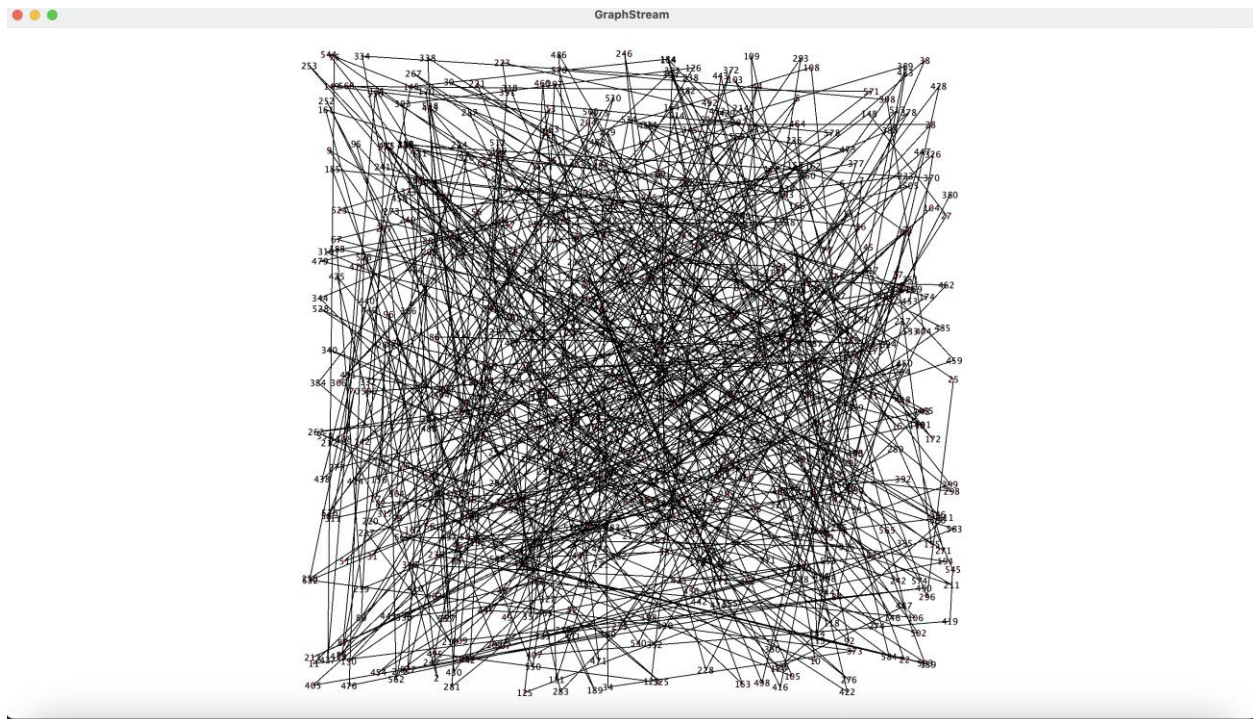
Random Swapping:

OptimizeRandomSwapping contains a static method randomSwapOptimization that takes a tour (represented as a list of integers), a distance matrix, and a number of iterations as input parameters. It returns a new tour that is optimized by randomly swapping two cities and checking if the new tour length is shorter. If it is, the new tour is accepted; otherwise, the swap is reverted. The method tourLength calculates the length of a given tour using the distance matrix.



Simulated Annealing

Simulated annealing is a probabilistic optimization algorithm that is inspired by the physical process of annealing. It starts with an initial solution (in this case, the given tour) and iteratively swaps two edges in the solution, checking if the new solution is better or not. If the new solution is better, it is accepted; otherwise, it may be accepted with a certain probability determined by a temperature parameter that gradually decreases over time. The algorithm eventually converges to a good solution, although it may not be the optimal solution.



Results & Mathematical Analysis:

The total nodes for the minimum spanning tree as per the given dataset is 584. The total cost of the MST is 5,13,000 meters. The cost of the Eulerian Tour is 19,00,000 meters without optimization.

The cost for each mentioned algorithm has been documented below after optimization:

Algorithm	Cost(in meters)	TSP Tour
Three-opt algorithm	10,28,377.84	586
Ant colony optimization	7,01,071.05	586
Simulated annealing	10,28,376	586
Random swapping	18,23,045.25	586

Ratio Comparison :

Ratio for Minimum spanning tree/ Three-opt: 1.73

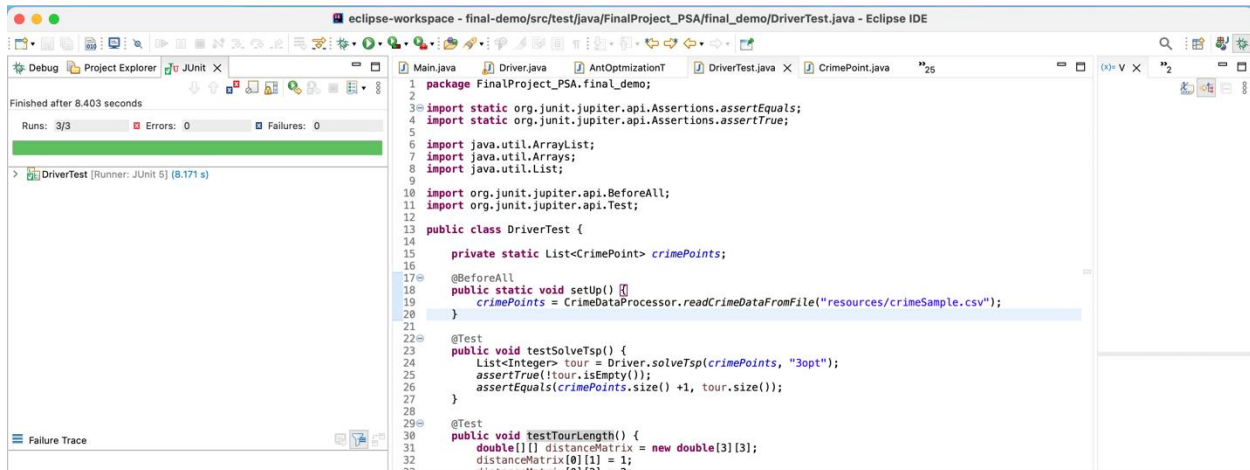
Ratio for Minimum spanning tree / Random swapping : 1.27

Ratio for Minimum spanning tree / Simulated annealing : 1.706

Ratio for Minimum spanning tree / Ant Colony : 1.73

Unit tests:

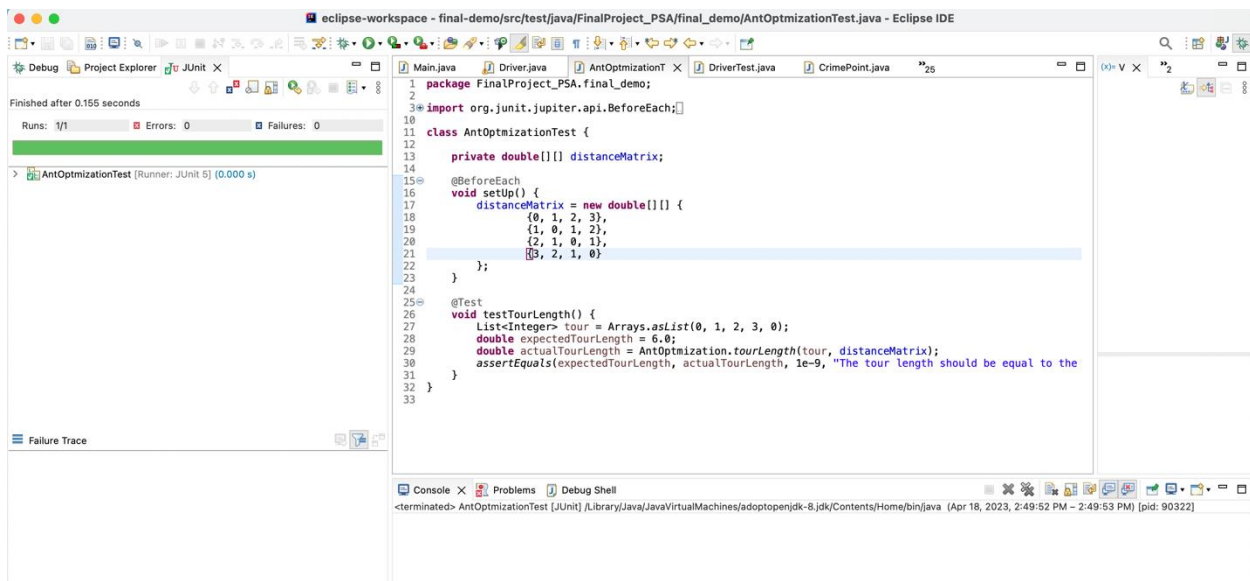
DriverTest.java



The screenshot shows the Eclipse IDE with the file `DriverTest.java` open. The left sidebar displays the JUnit test runner results, indicating that the tests passed successfully after 8.403 seconds. The main editor shows the source code of `DriverTest.java`, which includes imports for JUnit and utility classes, and two test methods: `testSolveTsp` and `testTourLength`.

```
1 package FinalProject_PSA.fina_demo;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4 import static org.junit.jupiter.api.Assertions.assertTrue;
5
6 import java.util.ArrayList;
7 import java.util.Arrays;
8 import java.util.List;
9
10 import org.junit.jupiter.api.BeforeAll;
11 import org.junit.jupiter.api.Test;
12
13 public class DriverTest {
14
15     private static List<CrimePoint> crimePoints;
16
17     @BeforeAll
18     public static void setUp() {
19         crimePoints = CrimeDataProcessor.readCrimeDataFromFile("resources/crimeSample.csv");
20     }
21
22     @Test
23     public void testSolveTsp() {
24         List<Integer> tour = Driver.solveTsp(crimePoints, "3opt");
25         assertTrue(!tour.isEmpty());
26         assertEquals(crimePoints.size() + 1, tour.size());
27     }
28
29     @Test
30     public void testTourLength() {
31         double[][] distanceMatrix = new double[3][3];
32         distanceMatrix[0][1] = 1;
33         distanceMatrix[0][2] = 2;
```

AntOptimizationTest.java



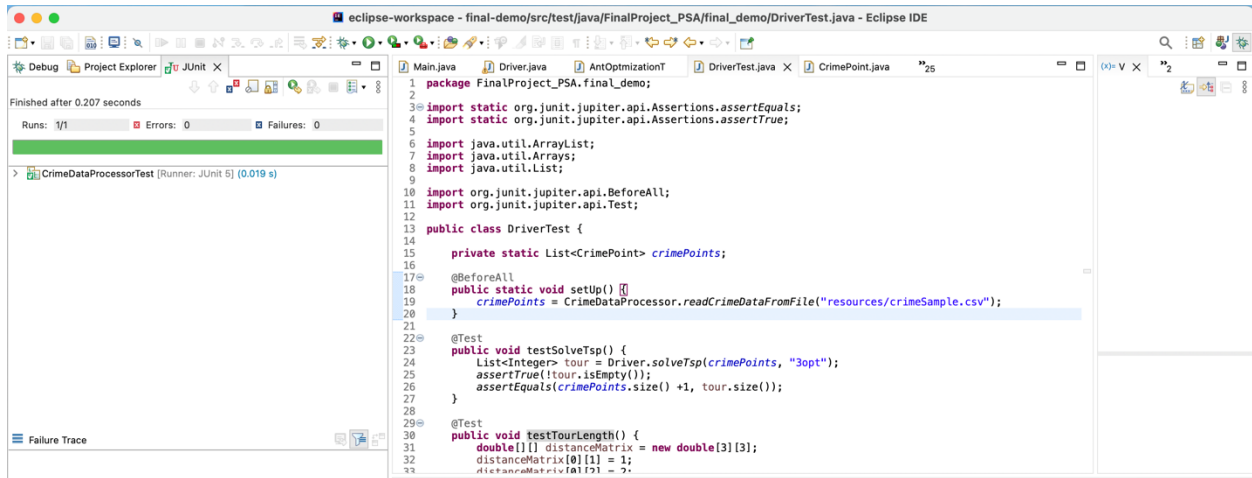
The screenshot shows the Eclipse IDE with the file `AntOptimizationTest.java` open. The left sidebar displays the JUnit test runner results, indicating that the tests passed successfully after 0.155 seconds. The main editor shows the source code of `AntOptimizationTest.java`, which includes imports for JUnit and utility classes, and two test methods: `setUp` and `testTourLength`. The `setUp` method initializes a `distanceMatrix` with specific values, and the `testTourLength` method tests the `AntOptimization.tourLength` method.

```
1 package FinalProject_PSA.fina_demo;
2
3 import org.junit.jupiter.api.BeforeEach;
4
5 class AntOptimizationTest {
6
7     private double[][] distanceMatrix;
8
9     @BeforeEach
10     void setUp() {
11         distanceMatrix = new double[][] {
12             {0, 1, 2, 3},
13             {1, 0, 1, 2},
14             {2, 1, 0, 1},
15             {3, 2, 1, 0}
16         };
17     }
18
19     @Test
20     void testTourLength() {
21         List<Integer> tour = Arrays.asList(0, 1, 2, 3, 0);
22         double expectedTourLength = 6.0;
23         double actualTourLength = AntOptimization.tourLength(tour, distanceMatrix);
24         assertEquals(expectedTourLength, actualTourLength, 1e-9, "The tour length should be equal to the");
25     }
26 }
27
28
29
30
31
32
33
```

The bottom of the screenshot shows the Console window with the following output:

```
<terminated> AntOptimizationTest [JUnit] /Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/bin/java (Apr 18, 2023, 2:49:52 PM - 2:49:53 PM) [pid: 90322]
```

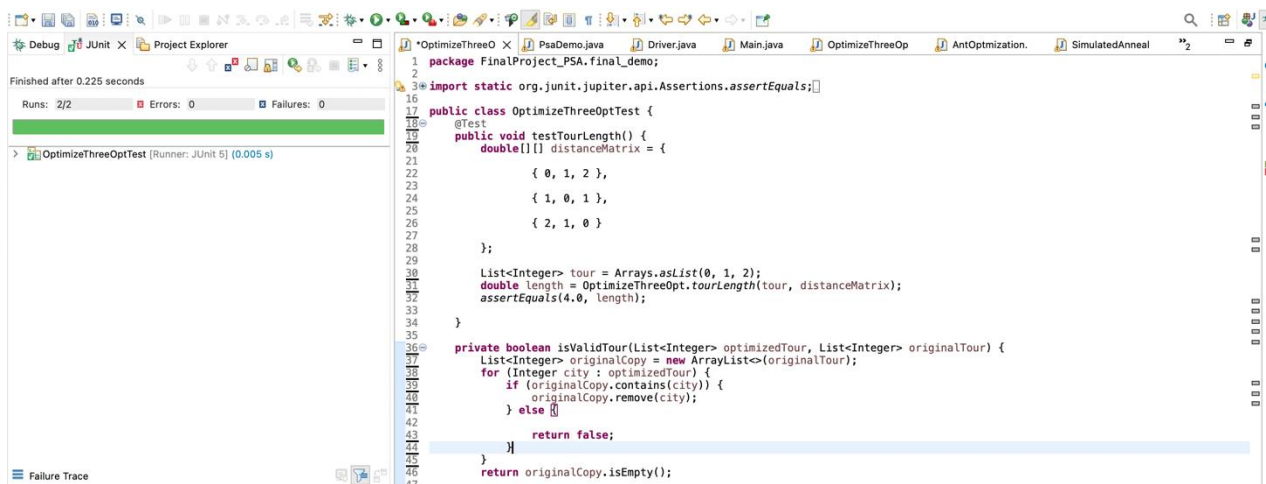
CrimeDataProcessorTest.java



The screenshot shows the Eclipse IDE with the file `CrimeDataProcessorTest.java` open. The left sidebar displays the JUnit runner results, indicating a successful run of `CrimeDataProcessorTest` in 0.019 seconds. The main editor shows the following Java code:

```
1 package FinalProject_PSA.fina_demo;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4 import static org.junit.jupiter.api.Assertions.assertTrue;
5
6 import java.util.ArrayList;
7 import java.util.Arrays;
8 import java.util.List;
9
10 import org.junit.jupiter.api.BeforeAll;
11 import org.junit.jupiter.api.Test;
12
13 public class DriverTest {
14
15     private static List<CrimePoint> crimePoints;
16
17     @BeforeAll
18     public static void setUp() {
19         crimePoints = CrimeDataProcessor.readCrimeDataFromFile("resources/crimeSample.csv");
20     }
21
22     @Test
23     public void testSolveTsp() {
24         List<Integer> tour = Driver.solveTsp(crimePoints, "3opt");
25         assertTrue(!tour.isEmpty());
26         assertEquals(crimePoints.size() + 1, tour.size());
27     }
28
29     @Test
30     public void testTourLength() {
31         double[][] distanceMatrix = new double[3][3];
32         distanceMatrix[0][1] = 1;
33         distanceMatrix[1][2] = 2;
```

OptimizeThreeOptTest.java



The screenshot shows the Eclipse IDE with the file `OptimizeThreeOptTest.java` open. The left sidebar displays the JUnit runner results, indicating a successful run of `OptimizeThreeOptTest` in 0.005 seconds. The main editor shows the following Java code:

```
1 package FinalProject_PSA.fina_demo;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5 public class OptimizeThreeOptTest {
6
7     @Test
8     public void testTourLength() {
9         double[][] distanceMatrix = {
10             { 0, 1, 2 },
11             { 1, 0, 1 },
12             { 2, 1, 0 }
13         };
14
15         List<Integer> tour = Arrays.asList(0, 1, 2);
16         double length = OptimizeThreeOpt.tourLength(tour, distanceMatrix);
17         assertEquals(4.0, length);
18     }
19
20     private boolean isValidTour(List<Integer> optimizedTour, List<Integer> originalTour) {
21         List<Integer> originalCopy = new ArrayList<>(originalTour);
22         for (Integer city : optimizedTour) {
23             if (originalCopy.contains(city)) {
24                 originalCopy.remove(city);
25             } else {
26                 return false;
27             }
28         }
29         return originalCopy.isEmpty();
30     }
31 }
```

OptimizeRandomSwappingTest.java

```
1 package FinalProject_PSA.fina_demo;
2
3 import org.junit.jupiter.api.Test;
4
5 public class OptimizeRandomSwappingTest {
6
7     @Test
8     public void testTourLength() {
9         double[][] distanceMatrix = {
10             {0, 1, 2},
11             {1, 0, 1},
12             {2, 1, 0}
13         };
14         List<Integer> tour = Arrays.asList(0, 1, 2);
15
16         double length = OptimizeRandomSwapping.tourLength(tour, distanceMatrix);
17         assertEquals(4.0, length);
18     }
19
20     @Test
21     public void testRandomSwapOptimization() {
22         double[][] distanceMatrix = {
23             {0, 1, 2},
24             {1, 0, 1},
25             {2, 1, 0}
26         };
27         List<Integer> tour = Arrays.asList(0, 1, 2);
28         int iterations = 100;
29         List<Integer> optimizedTour = OptimizeRandomSwapping.randomSwapOptimization(tour, distanceMatrix,
30             iterations);
31         assertTrue(isValidTour(optimizedTour, distanceMatrix));
32     }
33 }
```

SimulatedAnnealingTest.java

```
1 package FinalProject_PSA.fina_demo;
2
3 import org.junit.jupiter.api.Test;
4
5 public class SimulatedAnnealingTest {
6
7     @Test
8     public void testTourLength() {
9         double[][] distanceMatrix = {
10             {0, 1, 2},
11             {1, 0, 1},
12             {2, 1, 0}
13         };
14         List<Integer> tour = Arrays.asList(0, 1, 2);
15
16         double length = SimulatedAnnealing.tourLength(tour, distanceMatrix);
17         assertEquals(4.0, length);
18     }
19
20     @Test
21     public void testOptimizeWithSimulatedAnnealing() {
22         double[][] distanceMatrix = {
23             {0, 1, 2},
24             {1, 0, 1},
25             {2, 1, 0}
26         };
27         List<Integer> tour = Arrays.asList(0, 1, 2);
28         List<Integer> optimizedTour = SimulatedAnnealing.optimizeWithSimulatedAnnealing(tour, distanceMatrix,
29             iterations);
30         assertTrue(isValidTour(optimizedTour, distanceMatrix));
31     }
32 }
```

Conclusion:

In conclusion, the Traveling Salesman Problem is a challenging algorithm that seeks to find the shortest tour of n cities or points in a two-dimensional space. The complexity of deterministic solutions is $O(k!)$, which makes it an NP-hard problem. The Christofides algorithm is a popular approach used to solve the TSP, and it is often combined with other optimization methods such

as random swapping, 2-opt and/or 3-opt improvement, simulated annealing, ant colony optimization, genetic algorithms, among others. By utilizing these techniques, it is possible to improve the quality of solutions and reduce the computational complexity of the TSP. Despite the challenges associated with the TSP, it remains an important problem in operations research and computer science, with numerous practical applications in logistics, transportation, and network optimization.

References:

https://www.youtube.com/watch?v=oXb2nC-e_EA

[GeeksforGeeks | A computer science portal for geeks](#)

[java - Simulated Annealing TSP - Stack Overflow](#)

[The Traveling Salesman Problem: When Good Enough Beats Perfect](#)

[GraphStream - GraphStream - A Dynamic Graph Library \(graphstream-project.org\)](#)