

INFO 6205 Spring 2023 Project

Traveling Salesman

Team Members:

Ramya Devie Saravana Bhava
Varun Shetty

saravanabhava.r@northeastern.edu
shetty.varu@northeastern.edu

002770163
002778981

Introduction

- **Aim:** Implement the TSP (Travelling Salesman Problem) using Christofides algorithm and optimize the output using any two of the tactical (2-opt, 3-opt, random swapping algorithm) and strategic optimization (simulated annealing, ant colony, genetic algorithm).
- **Approach:** The Travelling Salesman Problem has been deemed as an NP-Hard Problem ever since its mathematical formulation in the 19th century. A brute force approach to solve this problem would be to consider each vertex, calculate the distance between this node and every other node, and then pick a route by analyzing the distances between all the available nodes using these same steps repeatedly.

This approach would end up giving us the best route possible. However, this approach would only work if the number of vertexes is low. The number of possible routes this method would have to analyze would be equal to $(N-1)!/2$ which signifies a substantial number for even a relatively small value of N like 12. Therefore, since finding an optimal solution for this problem is practically impossible, we use a different approach which would provide us with a reasonably good solution.

Our approach in this project is to utilize the Christofides Algorithm to find a reasonable solution and then later optimize this solution further using various tactical and strategic optimizing techniques. The solutions that will be provided would not be optimal, however, the length of the routes will be within 50% of the MST (Minimum Spanning Tree) Length which would be an acceptable solution for all practical purposes.

To solve the TSP, our first step would be to implement the Christofides Algorithm to find a route. To achieve this, we begin with creating an Adjacency Matrix which would have a 2-D Array containing the distance between each node. Using this 2D graph, we would proceed to find the Minimum Spanning Tree between the Nodes using the Prim's algorithm. Once we have obtained the MST, we would proceed to find the number of Odd Vertices in this MST along with the Minimum weight Matching Techniques to make sure that each node has an even degree of freedom. Finally, we would find the Eulerian tour of this Graph and find the route which would be an acceptable solution to this problem.

This tour will then further be optimized using various optimizing techniques such as random Swapping algorithms like 2-Opt, 3-Opt along with some strategic Algorithms such as Ant colony Optimization and Simulated Annealing. These techniques will be further discussed below.

Program

- **Data Structures:** Array list, Array, Custom data class (Location)

- **Classes:**

1. Edge
2. Location
3. Christofides
4. TwoOpt
5. ThreeOpt
6. AntColonyOptimization
7. SimulatedAnnealing
8. Prims
9. EdgeWeightedGraph
10. Bag
11. TSP

- **Algorithm:**

The algorithm used to implement TSP here is Christofides algorithm with two different tactical optimization such as 2-opt and 3-opt and with two different strategic optimization such as simulated annealing and ant colony optimization. Our objective here is to observe and compare the different optimizations applied to the TSP (Travelling Salesman Problem) using Christofides algorithm. We find MST of the given graph and then create a subgraph that includes only vertices of odd degrees. After finding minimum weight perfect matching, their edges and MST edges are combined to form a multigraph. Then we find eulerian circuit and convert it into Hamiltonian circuit by skipping repeated vertices.

Now, to optimize the output of the Christofides algorithm, we can use two types of optimization algorithms: tactical and strategic. Tactical optimization algorithms focus on improving the current solution by making local changes, while strategic optimization algorithms explore the search space to find better solutions.

Two common tactical optimization algorithms for the TSP are the 2-opt and 3-opt algorithms. The 2-opt algorithm works by removing two edges from the current solution and reconnecting them in a different way if doing so will shorten the total distance. The 3-opt algorithm works similarly, but it removes three edges and reconnects them in different ways. Simulated annealing works by randomly perturbing the current solution and accepting the perturbation if it improves the solution, but also sometimes accepting perturbations that worsen the solution to escape local optima. Ant colony optimization is inspired by the behavior of ants searching for food. In this algorithm, artificial ants deposit pheromones on the edges they travel, and the probability of choosing an edge is based on the amount of pheromone on the edge.

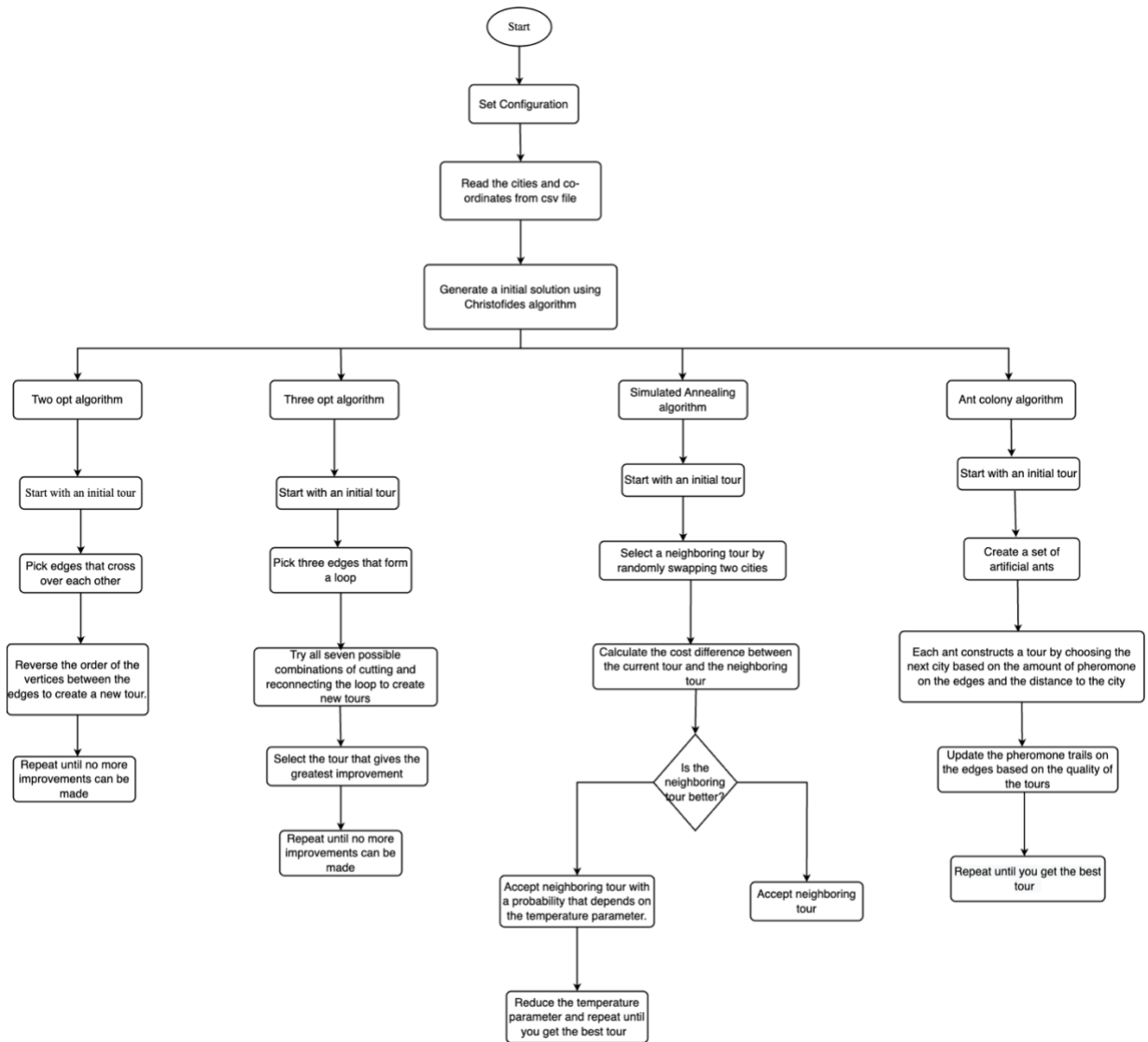
- **Invariants:** Invariant is a function, quantity, or property which remains unchanged when a specified transformation is applied.
 - **TSP:** The Travelling Salesman Project asks for the shortest possible routes that visits a given set of cities and returns to the starting city. The problem is NP-hard, meaning that no polynomial-time algorithm is known that can solve all instances of the problem. Some of the invariants of TSP are as follows.
 - i. Symmetry: The distance between two cities is the same in both directions.
 - ii. Triangle inequality: The distance between two cities is always less than or equal to the sum of the distances between those cities and any other city in the problem. Mathematically, this means that $d(i, j) \leq d(i, k) + d(k, j)$ for all cities i, j , and k .
 - iii. Non-negativity: The distance between any two cities is non-negative.
 - iv. Unique solution: There is only one optimal solution to the TSP.
 - **Christofides algorithm:** The Christofides algorithm is a heuristic algorithm that provides a solution to the TSP that is guaranteed to be within a factor of $3/2$ of the optimal solution. The algorithm has several invariants that are used to ensure the quality of the solution:
 - i. Minimum spanning tree: The first step of the algorithm is to find the minimum spanning tree (MST) of the complete graph of the cities. It is done using Prim's algorithm here.
 - ii. Odd-degree vertices: In the MST, some vertices will have odd degree (i.e., an odd number of edges incident to them). The second step of the algorithm is to find a minimum-weight perfect matching among the vertices with odd degree.
 - iii. Eulerian tour: The third step of the algorithm is to combine the edges of the MST and the minimum-weight perfect matching to form a connected graph that contains each vertex twice (once for the outgoing edge and once for the incoming edge). This graph can be shown to have a Eulerian tour, which is a path that visits each edge exactly once.
 - iv. Shortest path: The fourth and final step of the algorithm is to traverse the Eulerian tour, skipping any vertices that have already been visited, and computing the shortest path between adjacent unvisited vertices. The result is a Hamiltonian cycle that visits each vertex exactly once, and whose length is within a factor of $3/2$ of the optimal solution.

All the optimization algorithms have the below invariants in common.

- i. Tour representation: The solution is represented as a tour that visits each vertex exactly once and returns to the starting vertex.
 - ii. Improvement: The 2-opt algorithm only makes moves that reduce the total length of the tour, ensuring that the solution is continuously improving.
- **Two-opt algorithm:** The 2-opt algorithm is then used to optimize the solution obtained from the Christofides algorithm by iteratively removing two edges from the tour and reconnecting the resulting subpaths in a way that reduces the total length of the tour. The invariants used in the 2-opt algorithm are:
 - i. Validity: At any point during the optimization process, the solution must still satisfy the invariants of the TSP.

- **Three-opt algorithm:** The 3-opt algorithm is then used to optimize the solution obtained from the Christofides algorithm by iteratively removing three edges from the tour and reconnecting the resulting subpaths in a way that reduces the total length of the tour. The invariants used in the 3-opt algorithm are:
 - i. Validity: At any point during the optimization process, the solution must still satisfy the invariants of the TSP.
 - ii. Feasibility: The 3-opt algorithm only makes moves that result in a feasible solution, meaning that the resulting tour is still valid.
- **Simulated Annealing algorithm:** The SA algorithm is then used to optimize the solution obtained from the Christofides algorithm by iteratively changing the tour using random moves, and accepting moves that reduce the total length of the tour or moves that increase the total length of the tour with a certain probability. The invariants used in the SA algorithm are:
 - i. Validity: At any point during the optimization process, the solution must still satisfy the invariants of the TSP.
 - ii. Temperature: The SA algorithm uses a temperature parameter that controls the probability of accepting moves that increase the total length of the tour. The temperature decreases over time, which allows the algorithm to focus on improving the solution as it converges towards the optimal solution.
- **Ant colony optimization:** The ACO algorithm is then used to optimize the solution obtained from the Christofides algorithm by iteratively constructing and improving tours using a population of artificial ants. The invariants used in the ACO algorithm are:
 - i. Pheromone trail: The artificial ants deposit pheromone along the edges they traverse, which creates a trail that influences the behavior of subsequent ants. This allows the algorithm to explore promising areas of the search space and converge towards the optimal solution.
 - ii. Local search: The ACO algorithm can also include a local search heuristic, which allows ants to improve the tours by making small adjustments to the current solution. This can help the algorithm converge towards the optimal solution faster.
 - iii. Parameter tuning: The ACO algorithm uses several parameters that need to be carefully tuned to ensure good performance. These parameters include the pheromone evaporation rate, the pheromone deposit amount, and the balance between exploration and exploitation.

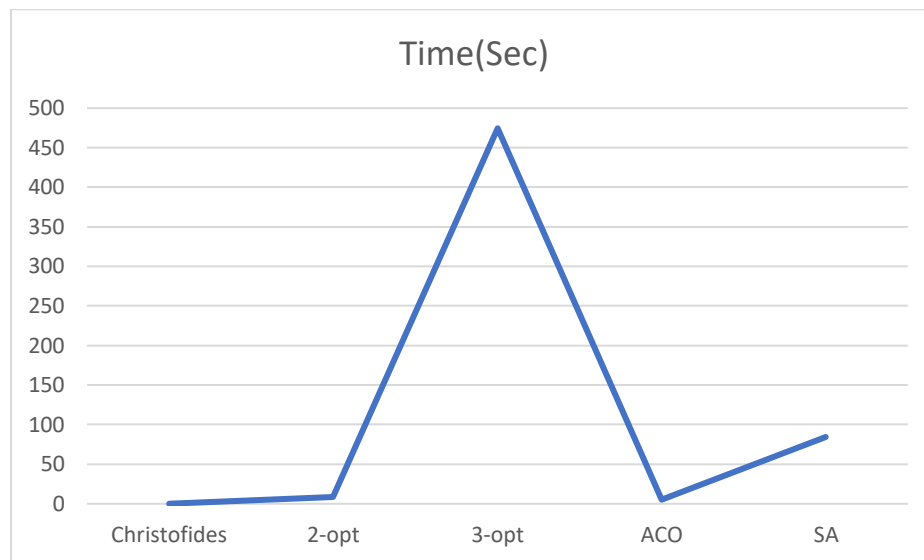
- Flow Chart:



- **Observations & Graphical Analysis:**

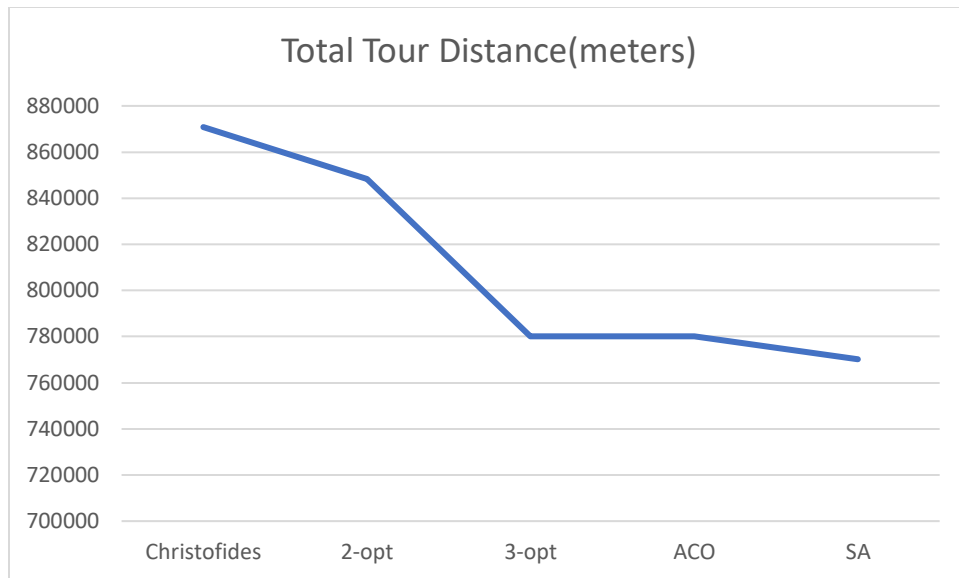
The time taken by three-opt to calculate shortest distance is higher than the other algorithms such as Christofides, two-opt, simulated annealing, ant colony optimization. This happens because worst case time complexity of three-opt is $O(n^3)$ compared to two-opt which has worst case time complexity of $O(n^2)$, so for large values of n (number of cities) three-opt takes more time but provides shortest tour distance compared to two-opt algorithm. When we tried giving the output of two-opt to the three-opt algorithm we were able to reduce the running time.

The 3-opt algorithm considers changes that involve removing and reconnecting three edges at a time. Specifically, the algorithm considers all possible combinations of three edges that can be removed from the tour and then reconnected in a different order. The resulting tour is then evaluated, and if it improves the total tour distance, it is accepted. By considering more possible changes to the tour, the 3-opt algorithm has a greater chance of finding a better solution than the 2-opt algorithm.



Time taken for each algorithm to generate shortest distance for TSP

The total tour distance calculated by each optimization algorithm is compared here. The Christofides algorithm can provide a better approximation than the nearest neighbor heuristic, and then the tactical optimization techniques of 2-opt and 3-opt can further improve the solution obtained from the Christofides algorithm. Similarly, the strategic optimization techniques of simulated annealing and ant colony algorithm can further refine the tour and potentially provide better solutions than the tactical optimization techniques.



Total Tour distance calculated by each algorithm to generate shortest distance for TSP

- Results & Mathematical Analysis:**

```

Projects x Services Files
org.neu.psa.algorithms.genetic.optimizations
AntColonyOptimization.java
SimulatedAnnealing.java

Source History
1 import org.junit.Test;

Output
Run (TSP) x Run (TSP) x

-----< org.neu.psa.TravellingSalesMan >-----
Building TravellingSalesMan 1.0-SNAPSHOT
-----[ jar ]-----

--- exec-maven-plugin:3.0.0:exec (default-cli) @ TravellingSalesMan ---
Vertices: 584
MST Distance: 514981.33100921
### CHRISTOFIDES Elapsed time: 0.068 seconds
Christofides Result Eulerian path: [141, 313, 438, 188, 327, 507, 372, 540, 5, 325, 86, 547, 493, 63, 28, 475, 165, 314, 476, 92, 396, 202, 163, 480, 127, 3, 279, 505, 581, 303, 101, 331, 283, 462, 461, 1
Christofides Result Hash: [ea746, 916a4, c80d0, d15fd, 9a790, 2e6f4, ca21a, c51e6, e93d1, 8064f, d5aeb, ceae8, d7b8b, b8ffc, 328a2, fb049, b5df9, 4bcb7, 0bcd0, cfd89, e5239, afe1b, f00de, 55f0d, 4709f, fd
Christofides Result length of the path: 870814.5346440018 meters

#### 2OPT Elapsed time: 0.45 seconds
Two OPT Route : [141, 313, 438, 188, 327, 507, 372, 540, 547, 493, 63, 28, 475, 165, 314, 476, 92, 202, 396, 86, 325, 5, 0, 368, 106, 305, 7, 140, 112, 62, 418, 221, 363, 121, 401, 542, 173, 308, 35, 196,
Two OPT Route Hash : [ea746, 916a4, c80d0, d15fd, 9a790, 2e6f4, ca21a, c51e6, ceae8, d7b8b, b8ffc, 328a2, fb049, b5df9, 4bcb7, 0bcd0, cfd89, afe1b, e5239, d5aeb, 8064f, e93d1, 47823, 64e7a, 74aa7, b9086,
Two OPT DISTANCE : 848452.6857424903 meters

#### 3 OPT Elapsed time: 474.514 seconds
Three OPT Route : [141, 325, 86, 547, 493, 63, 28, 475, 165, 314, 476, 92, 396, 202, 366, 13, 522, 349, 137, 99, 500, 125, 583, 100, 204, 306, 152, 151, 365, 196, 35, 308, 173, 542, 401, 121, 363, 221, 43
Three OPT Route Hash : [ea746, 8064f, d5aeb, ceae8, d7b8b, b8ffc, 328a2, fb049, b5df9, 4bcb7, 0bcd0, cfd89, e5239, afe1b, f22f2, 6c3a7, 6d4ba, 35c5c, 4972e, 829fb, fcbbe, 9686e, 1eaf0, 497b9, b4447, d7bfd
Three OPT DISTANCE : 780230.8513437398

### ACO Elapsed time: 5.319 seconds
Ant Colony Optimized Tour: [141, 325, 86, 547, 493, 63, 28, 475, 165, 314, 476, 92, 396, 202, 366, 13, 522, 349, 137, 99, 500, 125, 583, 100, 204, 306, 152, 151, 365, 196, 35, 308, 173, 542, 401, 121, 363
Ant Colony Route Hash : [ea746, 8064f, d5aeb, ceae8, d7b8b, b8ffc, 328a2, fb049, b5df9, 4bcb7, 0bcd0, cfd89, e5239, afe1b, f22f2, 6c3a7, 6d4ba, 35c5c, 4972e, 829fb, fcbbe, 9686e, 1eaf0, 497b9, b4447, d7bfd
Ant Colony Optimized Tour Length: 780230.8513437398

### Simulated Annealing Elapsed time: 84.645 seconds
Simulated Annealing Tour: [141, 0, 369, 106, 305, 7, 140, 112, 62, 418, 507, 372, 540, 5, 325, 547, 86, 28, 63, 493, 475, 165, 314, 476, 92, 202, 396, 13, 366, 349, 522, 137, 99, 500, 125, 583, 100, 204,
Simulated Annealing Route Hash : [ea746, 47823, 64e7a, 74aa7, b9086, ceae8, c828a, f37cc, 987b5, d1eaa, 2e6f4, ca21a, c51e6, e93d1, 8064f, ceae8, d5aeb, 328a2, b8ffc, d7b8b, fb049, b5df9, 4bcb7, 0bcd0, cf
Simulated Annealing Tour Length: 770170.309123788

BUILD SUCCESS

```

To compute the MST of connected edge-weighted graph, the Prim's algorithm has a time complexity proportional to $E \log V$, where E is Edges and V is Vertices. In terms of space complexity, the Christofides algorithm requires $O(E + V)$ space to store the graph and the minimum spanning tree.

The 2-opt algorithm has a worst-case time complexity of $O(n^2)$, where n is the number of cities, while the 3-opt algorithm has a worst-case time complexity of $O(n^3)$. The 2-opt and 3-opt algorithms require $O(n)$ space to store the current solution.

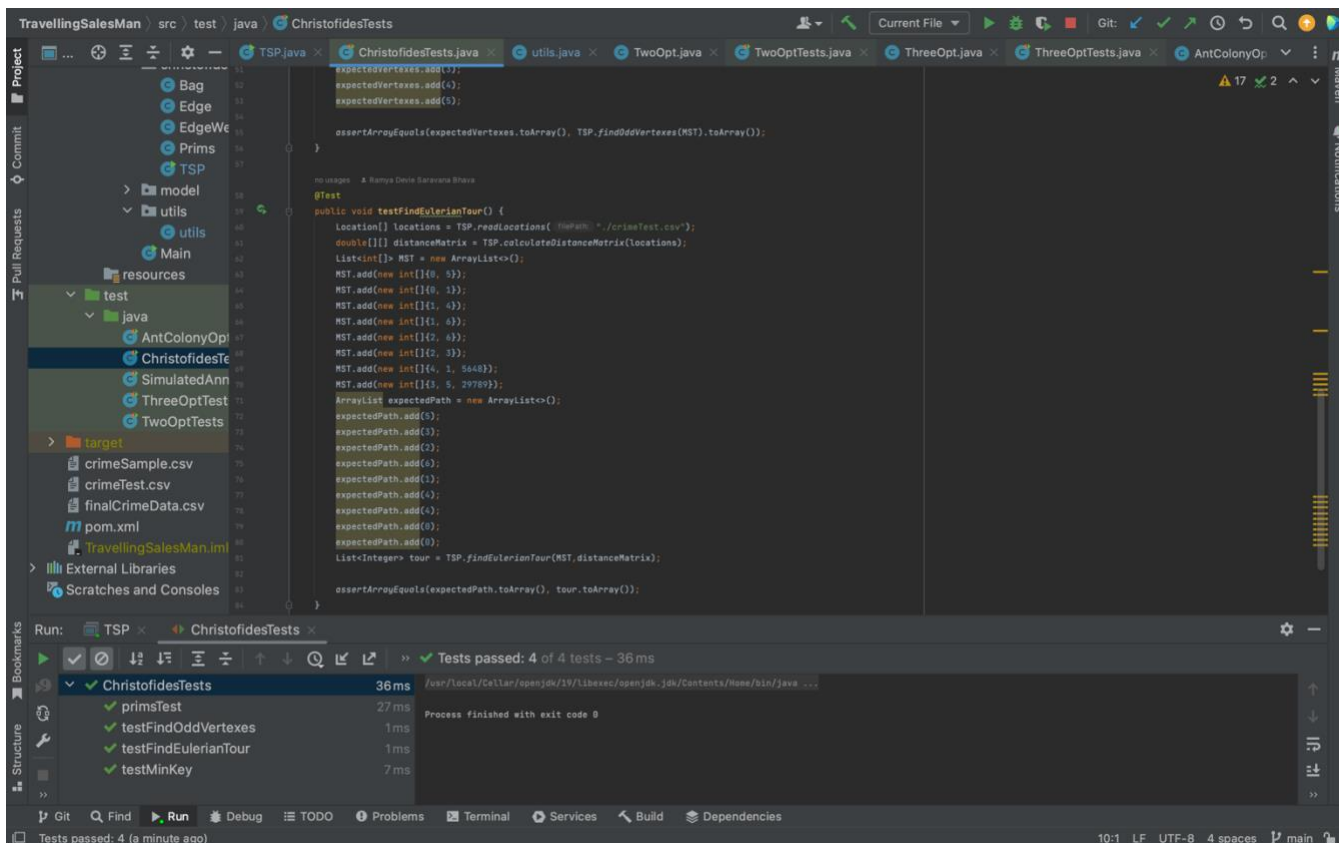
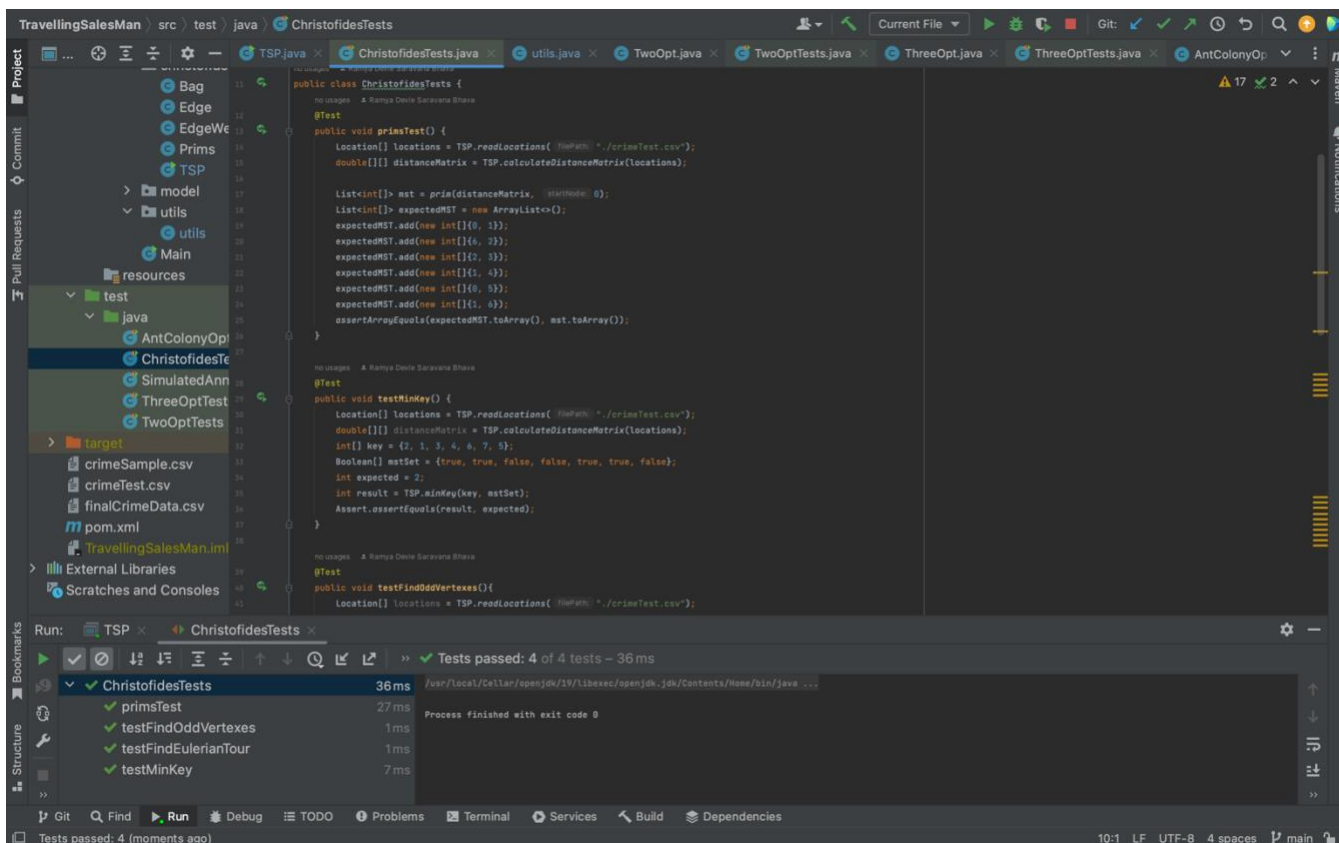
Simulated annealing is a metaheuristic algorithm where the time complexity depends on the number of iterations and the size of the neighborhood. The time complexity can be expressed as $O(kn^2)$, where k is the number of iterations and n is the number of cities. The space complexity of the algorithm is $O(n)$, which is required for storing the tour and $O(1)$ space to store the temperature.

The worst-case time complexity of the ACO (ant colony optimization) algorithm for the TSP is $O(n^2m)$, where n is the number of cities and m is the number of iterations. The space complexity is also $O(n^2)$ since the pheromone matrix requires n^2 memory locations. The number of ants used in the algorithm can also impact its running time, as well as the termination criterion for the algorithm. Thus, the actual running time can be affected by several factors and may differ from the theoretical worst-case estimate.

- **Unit tests:**

ChristofidesTests:

- **primsTest:** Loads few samples (7 data points) from crimeTest file and validates the MST generated by prims algorithm that is to be used as a base to find the TPS.
- **testMinKey:** Loads few samples (7 data points) from crimeTest file and validates the minKey method that is used to find MST.
- **testFindEulerianTour:** Loads few samples (7 data points) from crimeTest file and asserts the eulerian tour generated and returned for TSP class.
- **testFindOddVertexes:** Loads few samples (7 data points) from crimeTest file and tests the odd vertexes that is generated by the FindOddVertexes method.

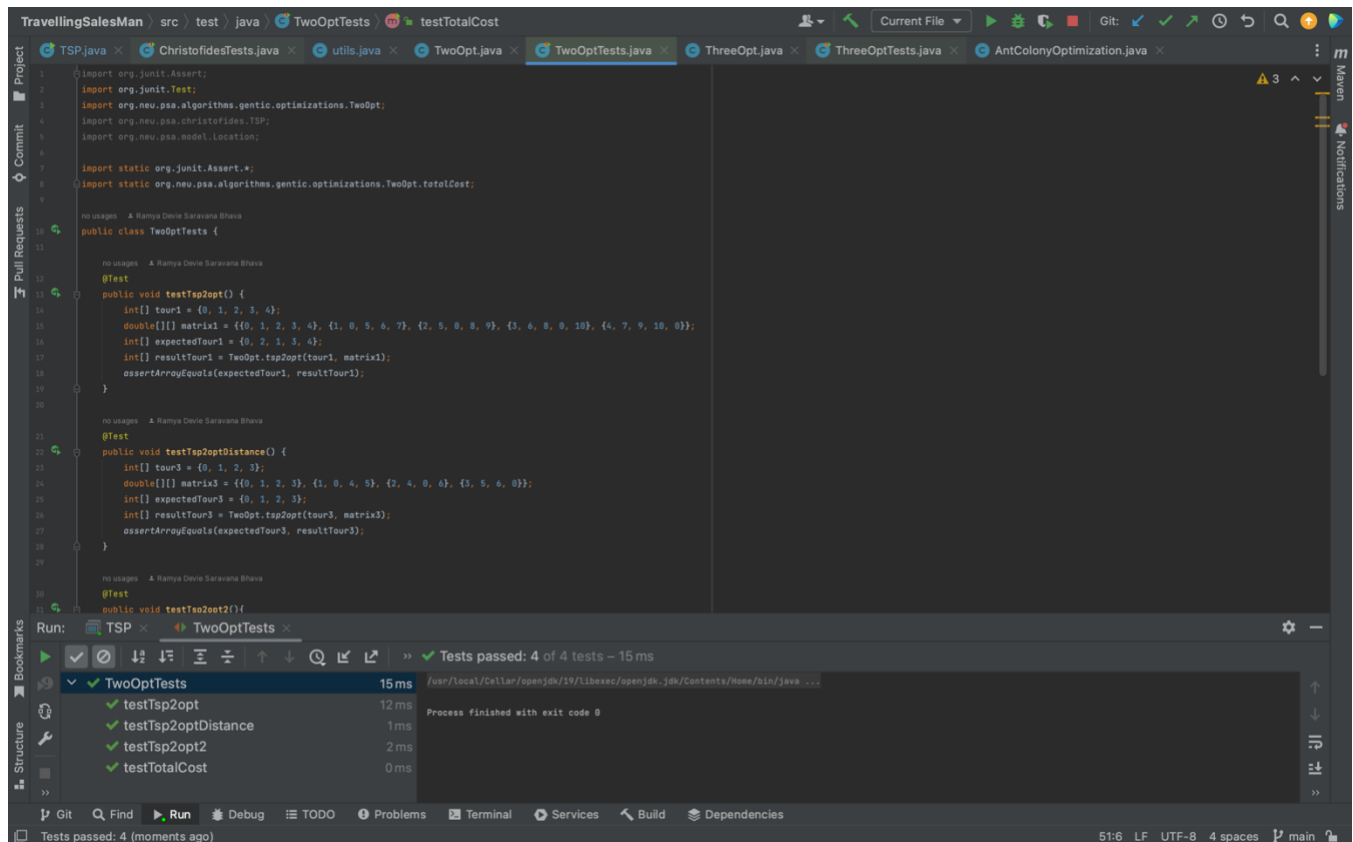


TwoOptTest:

testTsp2Opt: Initializes a small tour in form of an array along with adjacency matrix and asserts that tour generated by TwoOpt algorithm class.

testTotalCost: Initializes a small graph along with adjacency matrix and confirms the total cost i.e., the distance for this graph's TSP.

testTsp2Opt2: Initializes a larger tour in form of an array along with adjacency matrix and asserts that tour generated by TwoOpt algorithm class.



The screenshot shows an IDE with the following components:

- Editor:** Displays the `TwoOptTests.java` file. The code includes imports for `org.junit.Assert`, `org.junit.Test`, `org.neu.psa.algorithms.gentic.optimizations.TwoOpt`, `org.neu.psa.christofides.TSP`, and `org.neu.psa.model.Location`. It also includes static imports for `org.junit.Assert.*` and `org.neu.psa.algorithms.gentic.optimizations.TwoOpt.totalCost`. The class `TwoOptTests` contains three test methods: `testTsp2Opt()`, `testTsp2OptDistance()`, and `testTsp2Opt2()`. Each method initializes a tour and an adjacency matrix, then asserts the result of `TwoOpt.tsp2Opt()` against the expected tour.
- Run Console:** Shows the execution results of the tests. The output is: `Tests passed: 4 of 4 tests - 15 ms`. Below this, a table lists the tests and their durations:

Test Name	Duration
TwoOptTests	15 ms
testTsp2Opt	12 ms
testTsp2OptDistance	1 ms
testTsp2Opt2	2 ms
testTotalCost	0 ms

 The console also shows `Process finished with exit code 0`.
- Bottom Bar:** Displays the status bar with the text `Tests passed: 4 (moments ago)` and the file encoding `51:6 LF UTF-8 4 spaces`.

The screenshot shows an IDE with the following components:

- Editor:** Displays `TwoOptTests.java` with the following code:

```
int[] expectedTour3 = {0, 1, 2, 3};
int[] resultTour3 = TwoOpt.tsp2opt(tour3, matrix3);
assertArrayEquals(expectedTour3, resultTour3);

@Test
public void testTsp2opt2() {
    int[] tour2 = {0, 1, 2, 3, 4, 5, 6, 7, 8};
    double[][] matrix2 = {{0, 2, 3, 4, 5, 6, 7, 8, 9}, {2, 0, 2, 3, 4, 5, 6, 7, 8}, {3, 2, 0, 2, 3, 4, 5, 6, 7}, {4, 3, 2, 0, 2, 3, 4, 5, 6}, {5, 4, 3, 2, 0, 2, 3, 4, 5}, {6, 5, 4, 3, 2, 0, 2, 3, 4}, {7, 6, 5, 4, 3, 2, 0, 2, 3}, {8, 7, 6, 5, 4, 3, 2, 0, 2}, {9, 8, 7, 6, 5, 4, 3, 2, 0}};
    int[] expectedTour2 = {0, 1, 2, 3, 4, 5, 6, 7, 8};
    int[] resultTour2 = TwoOpt.tsp2opt(tour2, matrix2);
    assertArrayEquals(expectedTour2, resultTour2);

    @Test
    public void testTotalCost() {
        int[] cs = {0, 1, 2, 3};
        double[][] table = {
            {0.0, 2.0, 2.5, 3.0},
            {2.0, 0.0, 1.5, 2.0},
            {2.5, 1.5, 0.0, 1.5},
            {3.0, 2.0, 1.5, 0.0}
        };
        double expected = 8.0;
        double result = totalCost(cs, table);
        assertEquals(expected, result, delta: 0.0001);
    }
}
```
- Run Console:** Shows the test results for `TwoOptTests`:

Test Name	Duration
testTsp2opt	15 ms
testTsp2optDistance	12 ms
testTsp2opt2	1 ms
testTotalCost	2 ms
testTsp3Opt	0 ms

The console also indicates "Tests passed: 4 of 4 tests - 15 ms" and "Process finished with exit code 0".
- Bottom Bar:** Shows the status bar with "51:6 LF UTF-8 4 spaces" and "main".

ThreeOptTests:

testTsp3Opt: Initializes a small tour in form of an array along with adjacency matrix and asserts that tour generated by ThreeOpt algorithm class.

testTotalCost: Initializes a small graph along with adjacency matrix and confirms the total cost i.e., the distance for this graph's TSP.

testTsp3Opt2: Initializes a larger tour in form of an array along with adjacency matrix and asserts that tour generated by ThreeOpt algorithm class.

TravellingSalesMan / src / test / java / ThreeOptTests

TSP.java × utils.java × TwoOpt.java × ThreeOpt.java × ThreeOptTests.java × AntColonyOptimization.java ×

```
import java.util.*;

public class ThreeOptTests {

    @Test
    public void testTsp3Opt() {
        int[] tour1 = {0, 1, 2, 3, 4};
        double[][] matrix1 = {{0, 1, 2, 3, 4}, {1, 0, 5, 6, 7}, {2, 5, 0, 8, 9}, {3, 6, 8, 0, 10}, {4, 7, 9, 10, 0}};
        int[] expectedTour1 = {0, 2, 1, 4, 3};
        int[] resultTour1 = ThreeOpt.threeOpt(tour1, matrix1);
        assertEquals(expectedTour1, resultTour1);
    }

    @Test
    public void testTsp3OptDistance() {
        int[] tour2 = {0, 1, 2, 3, 4, 5, 6, 7, 8};
        double[][] matrix2 = {{0, 2, 3, 4, 5, 6, 7, 8, 9}, {2, 0, 2, 3, 4, 5, 6, 7, 8}, {3, 2, 0, 2, 3, 4, 5, 6, 7}, {4, 3, 2, 0, 2, 3, 4, 5, 6}, {5, 4, 3, 2, 0, 2, 3, 4, 5}, {6, 5, 4, 3, 2, 0, 2, 3, 4}, {7, 6, 5, 4, 3, 2, 0, 2, 3}, {8, 7, 6, 5, 4, 3, 2, 0, 2, 3}, {9, 8, 7, 6, 5, 4, 3, 2, 0, 2, 3}};
        int[] expectedTour2 = {0, 1, 2, 3, 4, 5, 6, 7, 8};
        int[] resultTour2 = ThreeOpt.threeOpt(tour2, matrix2);
        assertEquals(expectedTour2, resultTour2);
    }

    @Test
    public void testTsp3Opt2() {
        int[] tour3 = {0, 1, 2, 3};
        double[][] matrix3 = {{0, 1, 2, 3}, {1, 0, 4, 5}, {2, 4, 0, 6}, {3, 5, 6, 0}};
        int[] expectedTour3 = {0, 1, 2, 3};
        int[] resultTour3 = ThreeOpt.threeOpt(tour3, matrix3);
        assertEquals(expectedTour3, resultTour3);
    }
}
```

Run: TSP × ThreeOptTests ×

Tests passed: 5 of 5 tests – 27 ms

Test	Duration
ThreeOptTests	27 ms
testTsp3Opt	25 ms
testTsp3OptDistance	1 ms
testTsp3Opt2	0 ms
testReverse	0 ms
testTourLength	1 ms

Process finished with exit code 0

Tests passed: 5 (moments ago)

TravellingSalesMan / src / test / java / ThreeOptTests

TSP.java × utils.java × TwoOpt.java × ThreeOpt.java × ThreeOptTests.java × AntColonyOptimization.java ×

```
int[] expectedTour3 = {0, 1, 2, 3};
int[] resultTour3 = ThreeOpt.threeOpt(tour3, matrix3);
assertEquals(expectedTour3, resultTour3);
}

@Test
public void testReverse() {
    int[] tour = {0, 1, 2, 3, 4};
    int i = 1;
    int j = 3;
    int[] expected = {0, 3, 2, 1, 4};
    int[] result = ThreeOpt.reverse(tour, i, j);
    assertEquals(expected, result);
}

@Test
public void testTourLength() {
    int[] tour = {0, 1, 2, 3};
    double[][] dist = {
        {0.0, 2.0, 2.5, 3.0},
        {2.0, 0.0, 1.5, 2.0},
        {2.5, 1.5, 0.0, 1.5},
        {3.0, 2.0, 1.5, 0.0}
    };
    int expected = 7;
    int result = ThreeOpt.tourLength(tour, dist);
    assertEquals(expected, result);
}
}
```

Run: TSP × ThreeOptTests ×

Tests passed: 5 of 5 tests – 27 ms

Test	Duration
ThreeOptTests	27 ms
testTsp3Opt	25 ms
testTsp3OptDistance	1 ms
testTsp3Opt2	0 ms
testReverse	0 ms
testTourLength	1 ms

Process finished with exit code 0

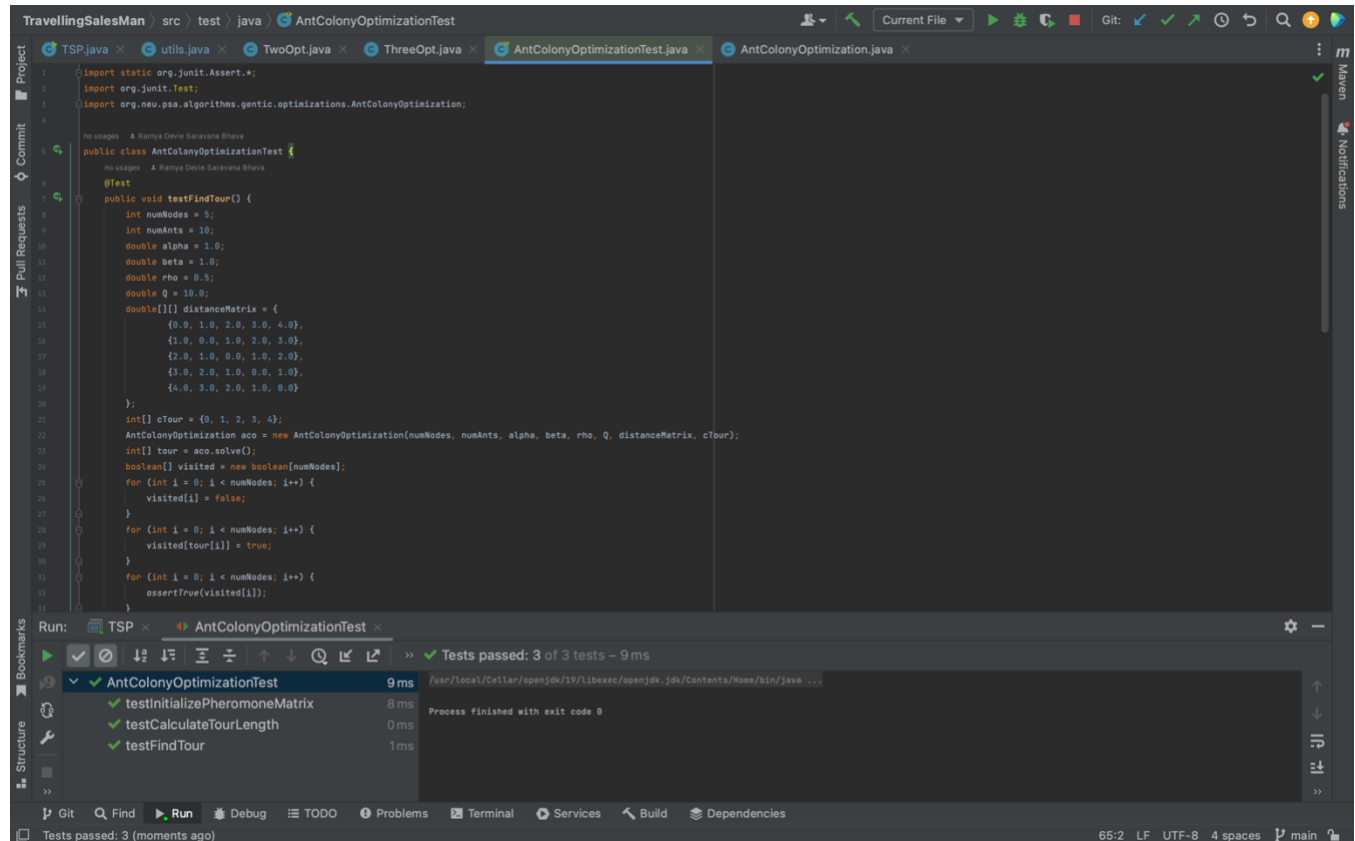
Tests passed: 5 (moments ago)

AntColonyOptimizationTest:

testFindTour: Initializes nodes and the adjacency matrix and validates tour generated and that all nodes are visited.

testCalculateTourLength: Initializes number of nodes in the tour along with the adjacency distance matrix and asserts the value of tour length.

testInitializePheromoneMatrix: Initializes number of nodes and verifies if the matrix values generated by getPheromoneMatrix are as expected.



The screenshot displays an IDE window for the `TravellingSalesMan` project. The `AntColonyOptimizationTest.java` file is open, showing the following code:

```
1 import static org.junit.Assert.*;
2 import org.junit.Test;
3 import org.neu.psa.algorithms.gentic.optimizations.AntColonyOptimization;
4
5 public class AntColonyOptimizationTest {
6     @Test
7     public void testFindTour() {
8         int numNodes = 5;
9         int numAnts = 10;
10        double alpha = 1.0;
11        double beta = 1.0;
12        double rho = 0.5;
13        double Q = 10.0;
14        double[][] distanceMatrix = {
15            {0.0, 1.0, 2.0, 3.0, 4.0},
16            {1.0, 0.0, 1.0, 2.0, 3.0},
17            {2.0, 1.0, 0.0, 1.0, 2.0},
18            {3.0, 2.0, 1.0, 0.0, 1.0},
19            {4.0, 3.0, 2.0, 1.0, 0.0}
20        };
21        int[] cTour = {0, 1, 2, 3, 4};
22        AntColonyOptimization aco = new AntColonyOptimization(numNodes, numAnts, alpha, beta, rho, Q, distanceMatrix, cTour);
23        int[] tour = aco.solve();
24        boolean[] visited = new boolean[numNodes];
25        for (int i = 0; i < numNodes; i++) {
26            visited[i] = false;
27        }
28        for (int i = 0; i < numNodes; i++) {
29            visited[tour[i]] = true;
30        }
31        for (int i = 0; i < numNodes; i++) {
32            assertTrue(visited[i]);
33        }
34    }
35}
```

The `Run` tab at the bottom shows the test results for `AntColonyOptimizationTest`:

Test Method	Duration	Status
testInitializePheromoneMatrix	8 ms	Passed
testCalculateTourLength	0 ms	Passed
testFindTour	1 ms	Passed

The overall test run summary indicates: **Tests passed: 3 of 3 tests - 9 ms**. The process finished with exit code 0.

The screenshot shows an IDE with the following components:

- Editor:** Displays `AntColonyOptimizationTest.java` with two test methods:
 - `testCalculateTourLength()`: Sets `numNodes = 4`, a distance matrix, and a tour. It calls `AntColonyOptimization` and asserts the tour length is 19.0.
 - `testInitializePheromoneMatrix()`: Sets `numNodes = 5`, a distance matrix, and a tour. It calls `AntColonyOptimization` and asserts the initial pheromone values.
- Run Console:** Shows the execution results:
 - Tests passed: 3 of 3 tests - 9ms
 - Test results table:

Test Method	Duration
testInitializePheromoneMatrix	8ms
testCalculateTourLength	0ms
testFindTour	1ms
- Bottom Bar:** Shows "Tests passed: 3 (moments ago)" and "65:2 LF UTF-8 4 spaces | main".

SimulatedAnnealing:

testOptimizeTour: Sets the nodes, distance matrix, along with the temperature and validates the optimal tour length generated by the algorithm is the same.

testOptimizedTourWithSameNodes: Sets the tour with just one node and asserts that the optimized tour is same when all nodes are same.

testCalculateTourLength: Sets the nodes, distance matrix, along with the temperature and validates the tour length generated by the algorithm is the same.

testSimulatedAnnealingWithZeroInitialTemperature: Sets the nodes, distance matrix, with initial temperate set to zero and validates the tour generated by the algorithm is as expected.

testSimulatedAnnealingWithNonSymmetricDistanceMatrix: Sets the nodes where the distance between nodes is not symmetric and validates the tour generated by the algorithm is as expected.

TravellingSalesMan / src / test / java / SimulatedAnnealingTest

```
private int[] tour;
4 usages
private double initialTemperature;
4 usages
private double[][] distanceMatrix;

@Before
public void setUp() {
    numNodes = 5;
    initialTemperature = 100.0;
    coolingRate = 0.99;
    tour = new int[]{0, 1, 2, 3, 4};
    distanceMatrix = new double[][]{{0, 2, 9, 10, 5}, {2, 0, 6, 4, 8}, {9, 6, 0, 3, 7}, {10, 4, 3, 0, 2}, {5, 8, 7, 2, 0}};
}

@Test
public void testOptimizeTour() {
    SimulatedAnnealing sa = new SimulatedAnnealing(numNodes, distanceMatrix, initialTemperature, coolingRate, tour);
    int[] optimizedTour = sa.optimizeTour();
    double optimizedTourLength = sa.calculateTourLength(optimizedTour);
    assertEquals("expected: 18.0, optimizedTourLength, delta: 0.001");
}

@Test
public void testCalculateTourLength() {
    SimulatedAnnealing sa = new SimulatedAnnealing(numNodes, distanceMatrix, initialTemperature, coolingRate, tour);
    double tourLength = sa.calculateTourLength(tour);
    assertEquals("expected: 18.0, tourLength, delta: 0.001");
}
```

Run: TSP x SimulatedAnnealingTest x

Tests passed: 5 of 5 tests – 651 ms

- ✓ SimulatedAnnealingTest 651 ms
- ✓ testOptimizeTourWithSameNodes 384 ms
- ✓ testCalculateTourLength 1 ms
- ✓ testOptimizeTour 264 ms
- ✓ testSimulatedAnnealingWithZeroInitialTer 1 ms
- ✓ testSimulatedAnnealingWithNonSymmetr 1 ms

Process finished with exit code 0

Tests passed: 5 (a minute ago)

TravellingSalesMan / src / test / java / SimulatedAnnealingTest

```
assertEquals("expected: 18.0, tourLength, delta: 0.001");
}

@Test
public void testOptimizeTourWithSameNodes() {
    // test that optimized tour is the same when all nodes are the same
    int[] sameNodesTour = new int[]{0, 0, 0, 0, 0};
    SimulatedAnnealing sa = new SimulatedAnnealing(numNodes, distanceMatrix, initialTemperature, coolingRate, sameNodesTour);
    int[] optimizedTour = sa.optimizeTour();
    assertEquals(sameNodesTour, optimizedTour);
}

@Test
public void testSimulatedAnnealingWithNonSymmetricDistanceMatrix() {
    double[][] distanceMatrix = {{0.0, 1.0}, {2.0, 0.0}};
    SimulatedAnnealing sa = new SimulatedAnnealing(numNodes: 2, distanceMatrix, initialTemperature: 100.0, coolingRate: 0.95, new int[] {0, 1});
    int[] optimizedTour = sa.optimizeTour();
    assertEquals(optimizedTour, new int[] {0, 1});
}

@Test
public void testSimulatedAnnealingWithZeroInitialTemperature() {
    double[][] distanceMatrix = {{0.0, 1.0}, {1.0, 0.0}};
    SimulatedAnnealing sa = new SimulatedAnnealing(numNodes: 2, distanceMatrix, initialTemperature: 0.0, coolingRate: 0.95, new int[] {0, 1});
    int[] optimizedTour = sa.optimizeTour();
    assertEquals(optimizedTour, new int[] {0, 1});
}
```

Run: TSP x SimulatedAnnealingTest x

Tests passed: 5 of 5 tests – 651 ms

- ✓ SimulatedAnnealingTest 651 ms
- ✓ testOptimizeTourWithSameNodes 384 ms
- ✓ testCalculateTourLength 1 ms
- ✓ testOptimizeTour 264 ms
- ✓ testSimulatedAnnealingWithZeroInitialTer 1 ms
- ✓ testSimulatedAnnealingWithNonSymmetr 1 ms

Process finished with exit code 0

Tests passed: 5 (a minute ago)

- **Conclusion:**

In conclusion, we have implemented the TSP using the Christofides algorithm and optimized the output using various optimization techniques. We started by constructing the minimum spanning tree of the given graph using Prim's algorithm and then used the nearest neighbor heuristic to find an approximate solution for the TSP. Then, we applied the Christofides algorithm to the graph to obtain a better approximation by connecting the odd-degree vertices in the minimum spanning tree using the minimum-weight perfect matching algorithm.

Next, we applied the tactical optimization techniques of two-opt and three-opt to further improve the solution by removing local optima and refining the tour. These techniques were able to significantly reduce the total distance of the tour in a short amount of time. Three-opt algorithm takes more time than two-opt because it considers all possible combinations of three edges that can be removed from the tour and then reconnected in a different order. The resulting tour is then evaluated, and if it improves the total tour distance, it is accepted.

Finally, we implemented the strategic optimization techniques of simulated annealing and ant colony algorithm to refine the tour further and avoid getting trapped in local optima. Simulated annealing is a metaheuristic that uses random search to escape local optima and improve the overall solution, while the ant colony algorithm mimics the behavior of ants in finding the shortest path between their colony and food source by depositing pheromones.

Overall, these results suggest that the choice of algorithm plays a significant role in the solution to the traveling salesman problem. While some algorithms may produce better results in certain situations, others may perform better in different circumstances. Therefore, researchers and practitioners must carefully evaluate different algorithms and select the most appropriate one for their particular use case. Among all the optimization algorithms (two-opt, three-opt, ant colony, simulated annealing) used here, simulated annealing algorithm provides the shortest distance for the data set provided.

- **References:**

1. https://www.youtube.com/watch?v=GiDsjiBOVoA&ab_channel=Reducible
2. <https://bank.engzenon.com/tmp/5e7f6ee5-d4dc-4aa8-9b0a-42d3c0feb99b/6062caf3-c600-4fc2-b413-4ab8c0feb99b/Algorithms-4th-Edition.pdf>
3. https://en.wikipedia.org/wiki/Christofides_algorithm
4. https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms
5. https://en.wikipedia.org/wiki/Simulated_annealing
6. <https://en.wikipedia.org/wiki/2-opt>
7. <https://en.wikipedia.org/wiki/3-opt>
8. <https://ieeexplore.ieee.org/document/5200345>
9. <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=995530>
10. https://www.researchgate.net/profile/Mauro-Birattari/publication/220380876_On_the_Invariance_of_Ant_Colony_Optimization/links/547b4b9b0cf293e2da2d7098/On-the-Invariance-of-Ant-Colony-Optimization.pdf