INFO 6205 Spring 2023 Project

Traveling Salesman Problem using Christofides Algorithm

Team Members:

Abhishek Singh (002703946)

Vedantini Gaikwad (002998254)

Meenal Sarwaiya (002737278)

Aim:

The goal of this paper is to examine the Christofides algorithm's effectiveness and that of its optimization techniques, including tactical techniques like 2-opt and 3-opt improvement and strategic techniques like simulated annealing and ant colony optimization. We can find excellent approximations to the traveling salesman issue using these techniques. This study will also examine the drawbacks and difficulties of these approaches and offer suggestions for further investigation.

Approach:

To achieve the goal of this report, we created a Java program using the IntelliJ IDE. The program's goal is to build the Christofides algorithm and use tactical and strategic approaches to optimize it. We employ object-oriented programming (OOP) ideas to make the software more modular, reuseable, and readable.

The program underwent numerous rounds of development. Initially, we created the Vertex, Edge, and Data classes and data structures required to implement the method. The Christofides algorithm and several optimization techniques, such as 2-opt and 3-opt improvement, simulated annealing, and ant colony optimization, were then put into practice.

To ensure the accuracy of the method and the outcomes, we additionally created invariants.
We use a Kaggle dataset with IDs, latitude and longitude values to test the aforementioned techniques. From there, we utilized the Haversine formula to calculate the distance before utilizing the Prims algorithm to determine the MST Value and identify the odd vertices. We align the edges using the perfect matching technique such that they create the least value. The cycle is then found by utilizing

the Euler graph and cycle, as well as the Hamiltonian cycle to eliminate unnecessary vertices. Later on in the tour, the topics of random swap, 2-opt, simulated annealing, and ant colony were covered. We discovered the ideal circuit value with the aid of these algorithms.

Program:

- List
- HashMaps
- Customized Classes (City, Edge)

Classes used :

- City
- DisjointSet
- Edge
- AntColonySolver
- ChristofidesSolver
- SimulatedAnnealingSolver
- ThreeOptSolver
- TwoOptSolver
- CircuitUtil
- DistanceCalculator
- TSPSolution

Algorithm:

I. Christofieds

Christofides' algorithm is a heuristic algorithm for the traveling salesman problem, which is a classic optimization problem in computer science. The goal of the problem is to find the shortest possible route that visits a set of cities and returns to the starting city.

Christofides' algorithm is a greedy algorithm that works in several steps:

1. Start with a complete undirected graph that represents all possible routes between the cities.
2. Use an algorithm like Prim's or Kruskal's to find the minimum spanning tree (MST) of the graph. The MST is the set of edges that connect all the cities together with the minimum possible total edge weight.

3. Identify all the vertices in the MST that have an odd degree (i.e., an odd number of edges connected to them). This is because any complete tour must have an even number of edges incident on each vertex.
4. Find a minimum-weight perfect matching between the odd-degree vertices. A perfect matching is a set of edges that connect all the odd-degree vertices together without any overlap. A minimum-weight perfect matching is one that has the minimum possible total weight.
5. Combine the MST and the minimum-weight perfect matching to form a connected graph that contains every vertex and has all vertices with even degree. This can be done by adding the edges of the minimum-weight perfect matching to the MST, creating a graph with every vertex having an even degree.
6. Find an Eulerian circuit (i.e., a tour that visits every edge exactly once) in the connected graph from step 5.
7. Remove any repeated vertices in the circuit to obtain the final tour.

   The algorithm has a worst-case running time of O(n^3), where n is the number of cities, due to the need to find the minimum-weight perfect matching. However, in practice, it often performs well and produces a tour that is very close to optimal in terms of length.

II. 2-Opt Algorithm:

The 2-opt algorithm is a heuristic algorithm used for solving the traveling salesman problem (TSP), which is a classic optimization problem in computer science. The goal of the TSP is to find the shortest possible route that visits a set of cities and returns to the starting city.

The 2-opt algorithm works in the following way:

1. Start with an initial tour of the cities. This can be any tour, such as a random tour or a tour produced by another algorithm.
2. Choose two edges in the tour and consider reversing the order of the cities between them. This will create a new tour.
3. Calculate the length of the new tour and compare it to the length of the original tour. If the new tour is shorter, accept it as the new tour. Otherwise, keep the original tour.
4. Repeat steps 2-3 for all possible pairs of edges in the tour.
5. If no improvement is possible, terminate the algorithm and output the current tour as the solution.

The name "2-opt" comes from the fact that the algorithm considers reversing two edges at a time. This is a local search algorithm, meaning that it makes small changes to the current solution in the hope of finding a better solution. It is a simple and efficient algorithm that can improve the quality of an initial solution, but it does not guarantee an optimal solution.

The 2-opt algorithm can be extended to k-opt, where k is the number of edges that are reversed at a time. However, larger values of k may require more computation time and may not always result in better solutions.

III. 3-Opt Algorithm

The 3-opt algorithm is a heuristic algorithm used for solving the traveling salesman problem (TSP), which is a classic optimization problem in computer science. The goal of the TSP is to find the shortest possible route that visits a set of cities and returns to the starting city.

The 3-opt algorithm is an extension of the 2-opt algorithm, which considers reversing two edges at a time. In the 3-opt algorithm, three edges are considered at a time. The algorithm works in the following way:

1. Start with an initial tour of the cities. This can be any tour, such as a random tour or a tour produced by another algorithm.
2. Choose three non-consecutive edges in the tour and consider all possible ways to reconnect them to form a new tour.
3. Calculate the length of the new tour and compare it to the length of the original tour. If the new tour is shorter, accept it as the new tour. Otherwise, keep the original tour.
4. Repeat steps 2-3 for all possible sets of three non-consecutive edges in the tour.
5. If no improvement is possible, terminate the algorithm and output the current tour as the solution.

The 3-opt algorithm is a local search algorithm, meaning that it makes small changes to the current solution in the hope of finding a better solution. It can be more powerful than the 2-opt algorithm in that it considers a larger number of possible solutions at each step. However, it also requires more computation time.

The 3-opt algorithm can be extended to k-opt, where k is the number of edges that are considered at a time. However, larger values of k may require exponentially more computation time and may not always result in better solutions. Therefore, the choice of k depends on the size of the problem and the available computational resources.

IV. Ant Colony

Ant colony optimization (ACO) is a metaheuristic algorithm that is inspired by the foraging behavior of ants. ACO is a population-based algorithm, meaning that it maintains a population of candidate solutions to the optimization problem being solved.

The algorithm works in the following way:

1. Initialize a population of ants at a random location in the search space. Each ant represents a potential solution to the problem being solved.
2. The ants move around the search space, following a probabilistic decision rule that is based on the amount of pheromone deposited on each edge of the graph. Pheromone is a chemical substance that ants use to communicate with each other.
3. At each step, an ant chooses its next location based on a combination of the pheromone level and the distance to the next location. This decision is based on a probabilistic formula, known as the transition rule, which takes into account the pheromone level and the distance to the next location.
4. Each time an ant moves along a path, it deposits pheromone on the edges it visits, proportional to the quality of the solution it represents.
5. Over time, the pheromone levels on the edges of the graph evolve, with more pheromone being deposited on the edges that are visited more frequently by the ants.
6. The algorithm terminates when a stopping criterion is met, such as a maximum number of iterations or a satisfactory solution is found.

ACO has been shown to be effective in solving a variety of combinatorial optimization problems, such as the traveling salesman problem, the quadratic assignment problem, and the job shop scheduling problem. ACO is particularly useful for problems with large solution spaces, where brute-force methods become impractical.

V. Simulated Annealing

Simulated annealing is a metaheuristic algorithm used to solve combinatorial optimization problems. The algorithm is inspired by the annealing process used in metallurgy, where a metal is heated and cooled slowly to reduce its defects and improve its strength. In simulated annealing, the process of heating and cooling is replaced with a probabilistic search process to find the global optimum of an objective function.

The algorithm works by maintaining a current candidate solution and iteratively moving to a new solution, accepting it with a probability that depends on the difference in the objective function value and a temperature parameter. The temperature parameter controls the acceptance of worse solutions at the beginning of the search when the algorithm is exploring the solution space and allows it to escape from local optima. As the search progresses, the temperature is gradually decreased, and the probability of accepting worse solutions is reduced, allowing the algorithm to converge towards the optimal solution.

The algorithm works in the following way:

1. Initialize a current solution randomly or using a heuristic algorithm.
2. Set an initial temperature value.

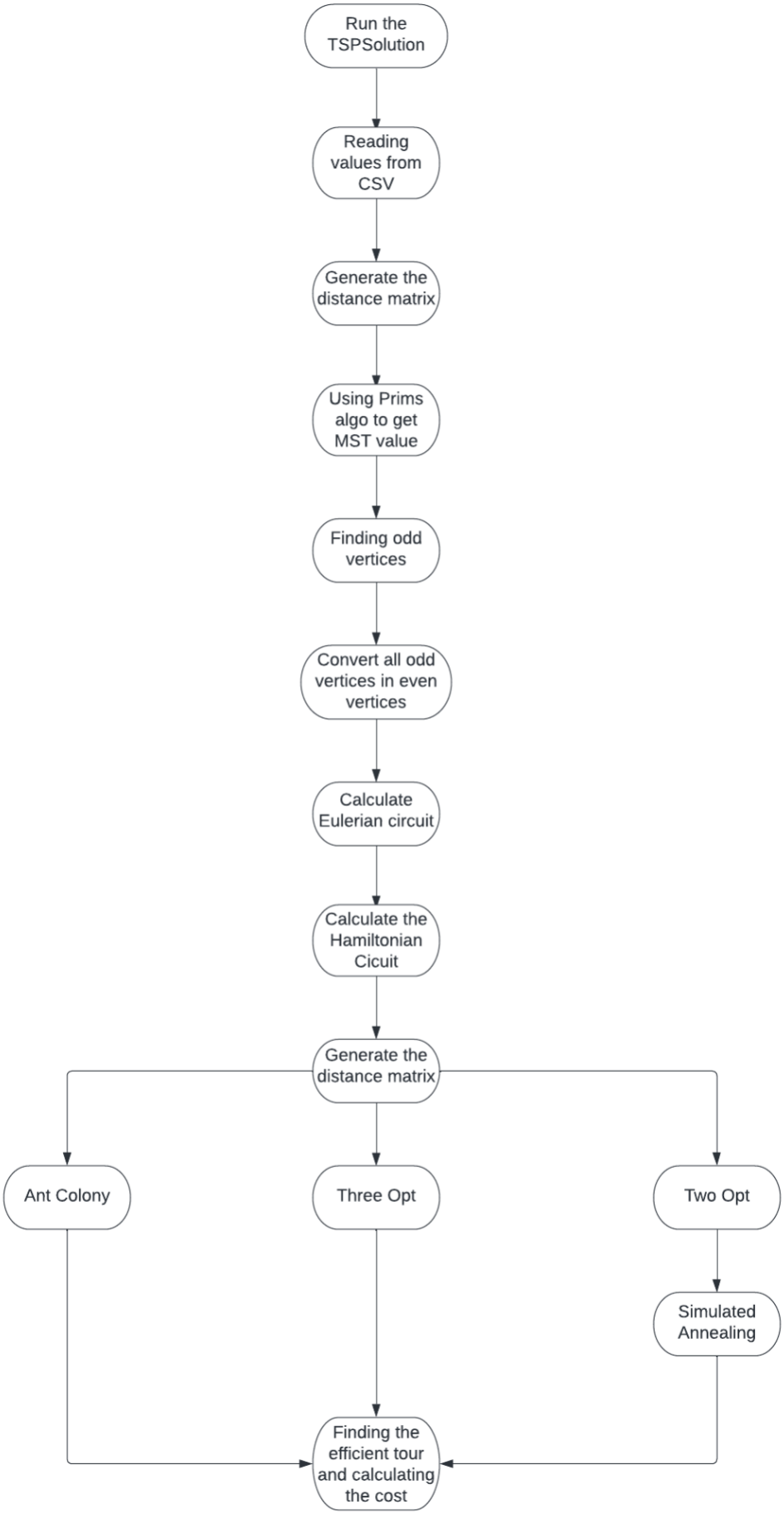
3. Iterate through the following steps until the stopping criterion is met: a. Perturb the current solution to generate a new candidate solution. b. Calculate the change in the objective function value between the current and the new candidate solution. c. If the new solution is better than the current solution, accept it as the new current solution. d. If the new solution is worse than the current solution, accept it with a probability that depends on the temperature and the difference in the objective function value. e. Reduce the temperature according to a cooling schedule.
4. Return the best solution found during the search process.

   Simulated annealing has been successfully applied to a wide range of optimization problems, including the traveling salesman problem, the graph coloring problem, and the knapsack problem. The effectiveness of the algorithm depends on the choice of the cooling schedule and the acceptance probability function, which can be adapted to the problem being solved.

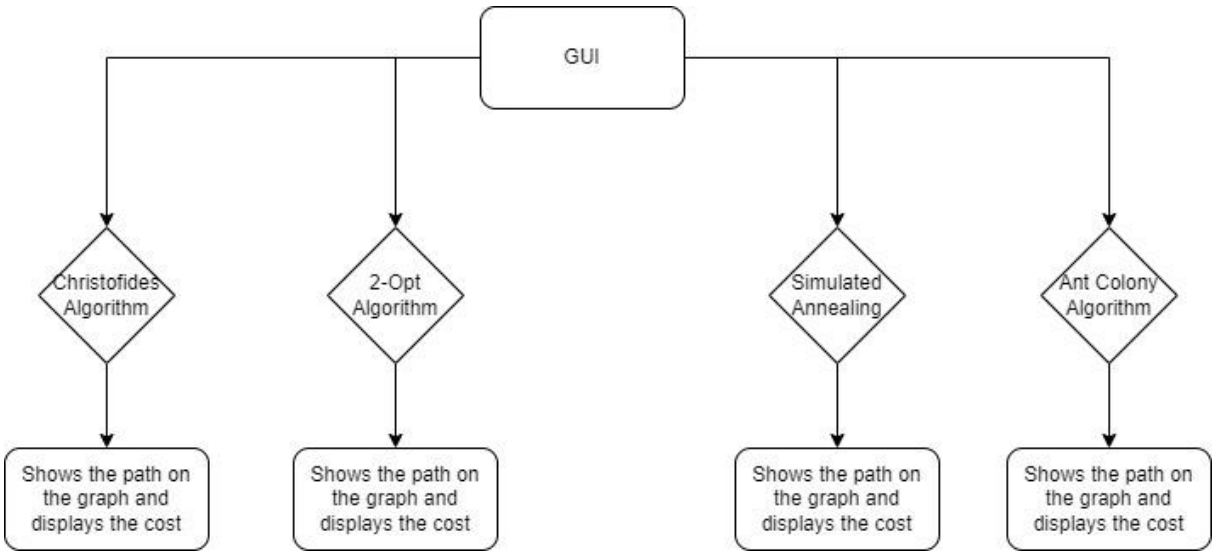
Invariants

- The graph needs to be linked and undirected.
- It is necessary to connect the least spanning tree.
- There must be an even number of vertices in the subgraph of odd degree.
- The edges in the least weight perfect matching must join vertices of an odd degree.
- Vertices in the composite graph must all be of even degree.
- The vertex where the Eulerian circuit begins and ends must be the same
- The edges generated by the primality techniques used to construct the Minimal Spanning Tree will all have the same value throughout the program.
- The information that is taken from the dataset and added to the list of data types utilized by the user interface to identify vertices.
- Each tour is tracked in its own list so that it may be used to calculate the traversal cost and fed to other algorithms to get the most efficient value.

Flowchart (Algorithm):


Run the TSPSolution
Reading values from CSV
Generate the distance matrix
Using Prims algo to get MST value
Finding odd vertices
Convert all odd vertices in even vertices
Calculate Eulerian circuit
Calculate the Hamiltonian Cicuit
Generate the distance matrix
Ant Colony
Three Opt
Two Opt
Simulated Annealing
Finding the efficient tour and calculating the cost

Flowchart (UI):



Observation and Graphical Analysis:

2-Opt: For 2 Opt optimization techniques, the cost remains constant as the no. of iterations increase. The time complexity for this algorithm is O(n2).

Please find the data and and the graph showing variation of iterations over cost in the excel.

3-Opt: For 3 Opt optimization techniques, the cost remains constant as the no. of iterations increase. The time complexity for this algorithm is O(n2).

Please find the data and and the graph showing variation of iterations over cost in the excel.

Ant colony optimization: By altering the number of iterations, we conducted our benchmarking for the ant colony optimization. The following values have been provided as the formula's parameters::

- While the ant selects the following city, Alpha () controls the weight or importance given to the pheromone trail. We've assigned alpha a value of 1.0 in our benchmarks.
- While the ant selects the next city, Beta () controls the weight or importance given to the heuristic information. We assigned beta a value of 5.0 in our benchmarks.

- The rate of pheromone trail evaporation over time is known as evaporation. This setting prevents the pheromone trail from being too strong and influencing the ants' choices, which could produce less-than-ideal results. The pheromone trail evaporates more quickly when the evaporation value is larger. We have assigned evaporation a value of 0.5.
- The pheromone that an ant deposits on the trail is determined by the parameter Q. More pheromone is deposited on the path when Q is higher in value. For Q, we've given a value of 500.

Find the graph and data showing variation of cost over iterations in the excel.

Simulated Annealing optimization: For simulated annealing optimization, we have done the benchmarking by varying the cooling rate.

Please find the data and the graph showing variation of cost over cooling rate

Result and Mathematical Analysis:

Haversine formula:

In order to create the distance matrix, we calculated the distance between the input file's edges using the Haversine formula. The distance between two places on the surface of a sphere, such as the Earth, can be calculated using the Haversine formula, which is a mathematical formula used in navigation. Given two places' latitudes and longitudes, it is frequently used to determine their distance from one another. The law of haversines, which states that for a triangle on the surface of a sphere, each angle's haversine is equal to the haversine of the sum of the other two angles less the product of the haversines of those angles, is the source of the Haversine formula.

The Haversine formula is given by:
d = 2r arcsin(sqrt(sin^2((lat2-lat1)/2) + cos(lat1) * cos(lat2) * sin^2((lon2-lon1)/2)))
where:
d is the distance between the two points (in the same units as r, typically kilometers or miles)
r is the radius of the sphere (in the same units as d)
lat1 and lat2 are the latitudes of the two points (in degrees)
lon1 and lon2 are the longitudes of the two points (in degrees)

Simulated Annealing:

Simulated annealing is a process where an initial solution is used as a starting point, then iteratively new candidate solutions are generated by making minor random changes to the first solution. The algorithm then analyzes the objective function for each potential solution and, using a probability distribution, determines whether to accept or reject the new answer.

Based on the difference between the objective function values of the current and alternative solutions as well as a temperature parameter that decreases over time in accordance with a cooling schedule, the Metropolis-Hastings algorithm calculates the probability distribution for simulated annealing. The cooling schedule is frequently designed to find a compromise between exploitation (using the existing solution to the fullest degree) and exploration (searching for new options).

P(x',x,k,T) = exp[-(f(x') - f(x))/(kT)] is the mathematical formula for the acceptance probability of a candidate solution with objective function value f(x) and temperature T at iteration k.
where T is the current temperature, k is a constant, x is the current solution, and x' is the potential solution.

Ant Colony Optimization:

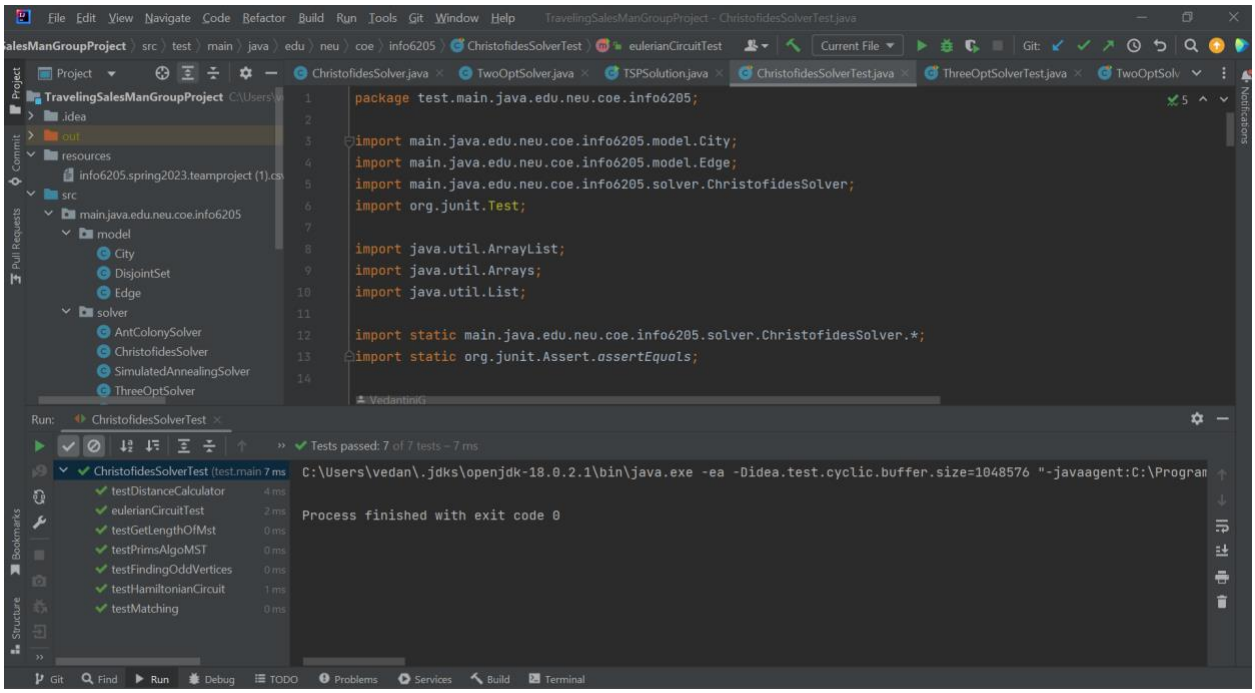
In Ant Colony, the ACO algorithm determines the pheromone trail levels for each edge in the graph and updates them based on how well the ants' solutions work. To determine the likelihood that an ant would select a certain edge, the pheromone level and a heuristic value—which denotes the edge's desirability based on elements like its length or cost—are combined.

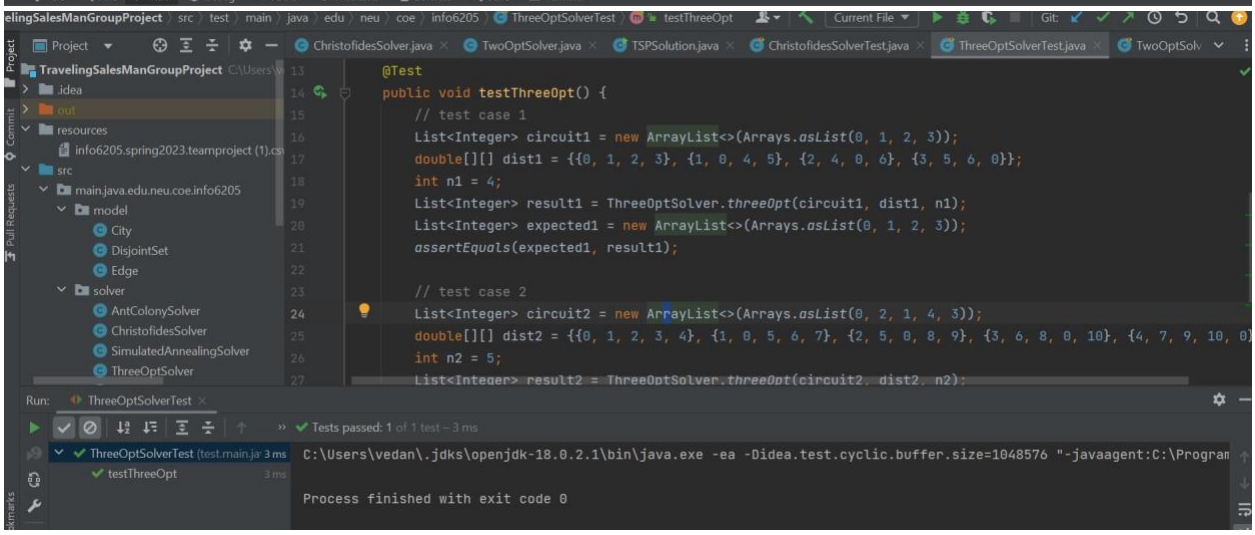
p(i,j) is equal to [(i,j) (i,j)] / [(i,k) (i,k)].

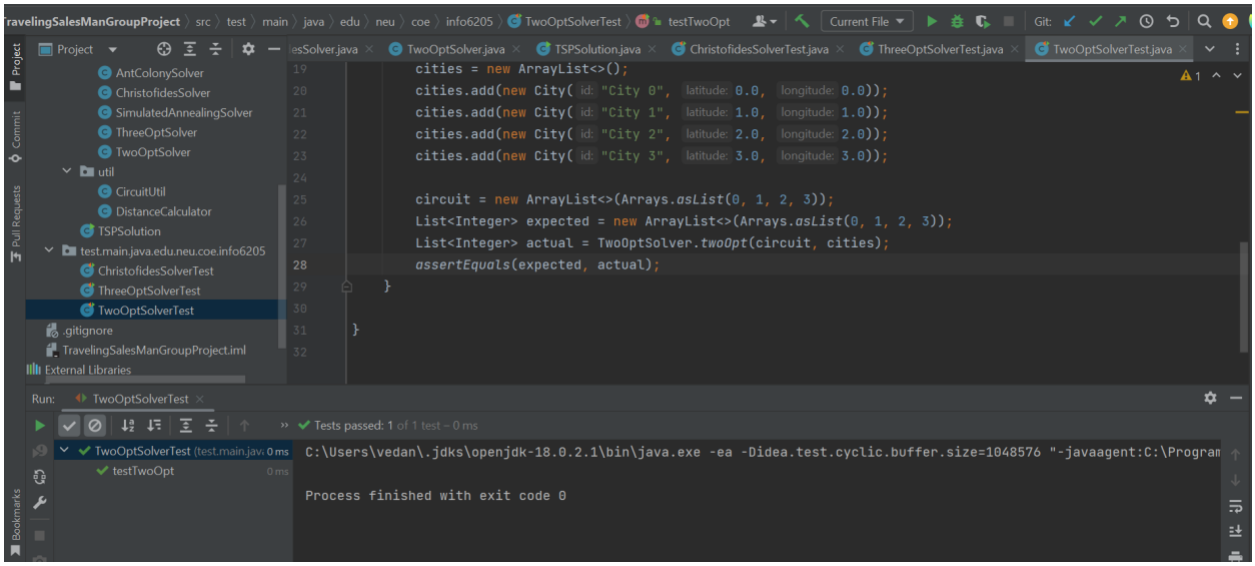
where p(i,j) is the likelihood that an ant would move from node i to node j, (i,j) is the pheromone level on that edge, (i,j) is the edge's heuristic value, and and are parameters that regulate the relative weights of the pheromone and heuristic values, respectively. The probability of all the edges leaving node i are totaled in the denominator.

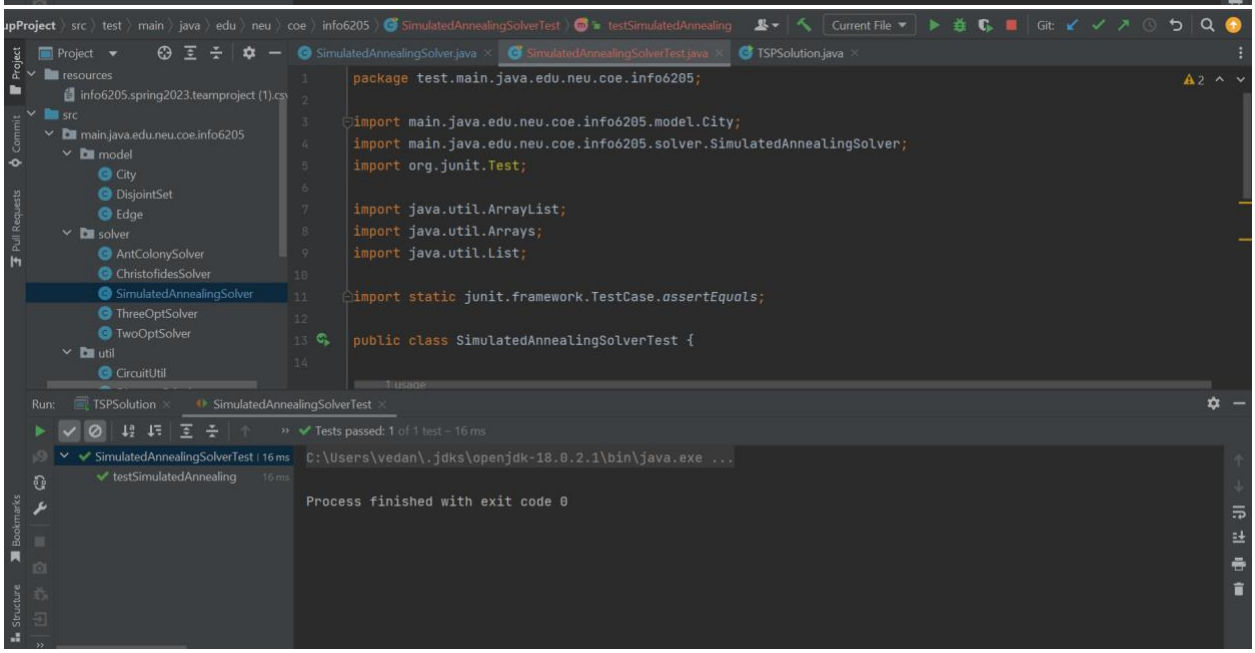
Prims' Algorithm is used for finding out the MST (Minimum spanning tree).
We have used the Hungarian Algorithm to calculate the minimum weight perfect matching.
Christofides algorithm is used to solve TSP.
2 opt and random swap are optimizations we have used to get the most optimal solution for the TSP.
Note: For optimization techniques like Ant Colony Optimization and Simulated Annealing, since the complexity is dependent on input parameters, it is tough to determine the exact time complexity.

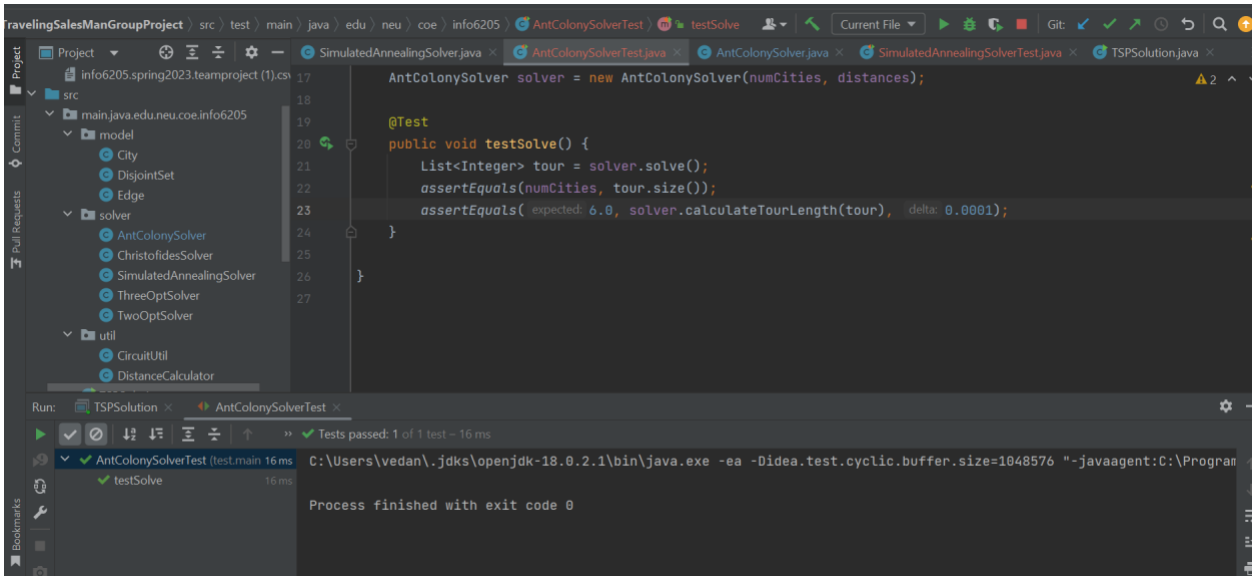
Unit Tests Screenshot:

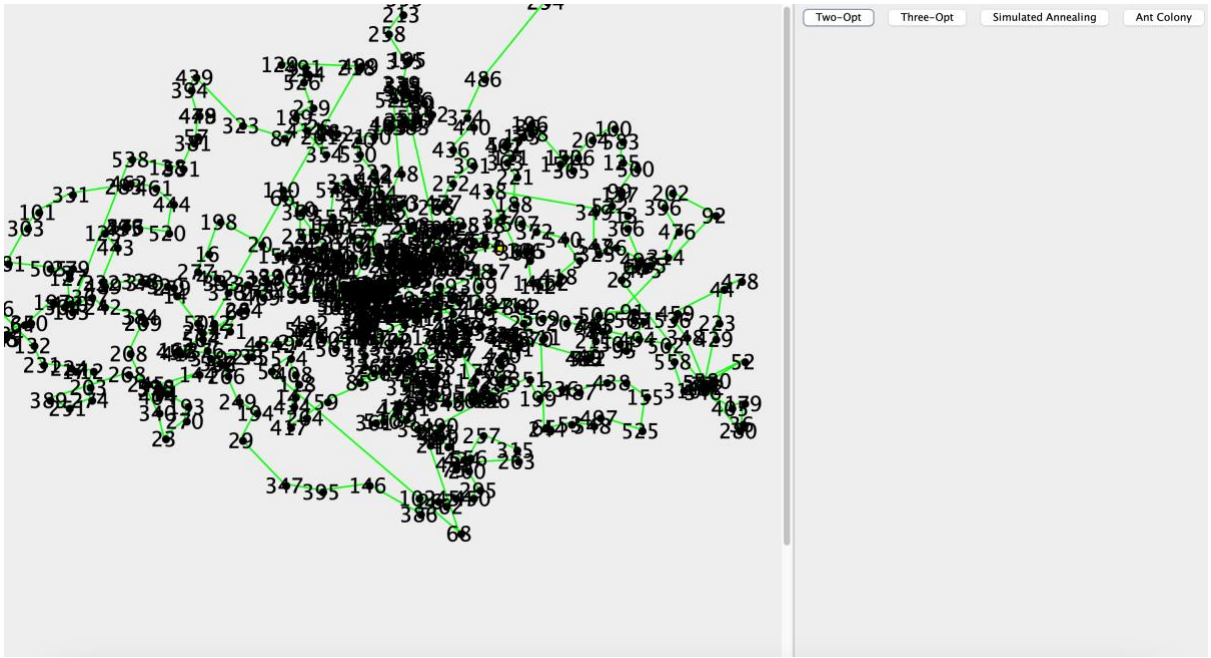





Project
AntColonySolver
ChristofidesSolver
SimulatedAnnealingSolver
ThreeOptSolver
TwoOptSolver
util
CircuitUtil
DistanceCalculator
TSPSolution
test.main.java.edu.neu.coe.info6205
ChristofidesSolverTest
ThreeOptSolverTest
TwoOptSolverTest
.gitignore
TravelingSalesManGroupProject.iml
External Libraries
TwoOptSolver.java
TSPSolution.java
ChristofidesSolverTest.java
ThreeOptSolverTest.java
TwoOptSolverTest.java
cities = new ArrayList<>();
cities.add(new City(id: "City 0", latitude: 0.0, longitude: 0.0));
cities.add(new City(id: "City 1", latitude: 1.0, longitude: 1.0));
cities.add(new City(id: "City 2", latitude: 2.0, longitude: 2.0));
cities.add(new City(id: "City 3", latitude: 3.0, longitude: 3.0));
circuit = new ArrayList<>(Arrays.asList(0, 1, 2, 3));
List<Integer> expected = new ArrayList<>(Arrays.asList(0, 1, 2, 3));
List<Integer> actual = TwoOptSolver.twoOpt(circuit, cities);
assertEquals(expected, actual);
}
}
Run: TwoOptSolverTest
Tests passed: 1 of 1 test – 0 ms
TwoOptSolverTest (test.main.jav: 0 ms
testTwoOpt 0 ms
C:\Users\vedan\.jdks\openjdk-18.0.2.1\bin\java.exe -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program
Process finished with exit code 0


Project
resources
info6205.spring2023.teamproject (1).csv
src
main.java.edu.neu.coe.info6205
model
City
DisjointSet
Edge
solver
AntColonySolver
ChristofidesSolver
SimulatedAnnealingSolver
ThreeOptSolver
TwoOptSolver
util
CircuitUtil
SimulatedAnnealingSolver.java
SimulatedAnnealingSolverTest.java
TSPSolution.java
package test.main.java.edu.neu.coe.info6205;
import main.java.edu.neu.coe.info6205.model.City;
import main.java.edu.neu.coe.info6205.solver.SimulatedAnnealingSolver;
import org.junit.Test;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import static junit.framework.TestCase.assertEquals;
public class SimulatedAnnealingSolverTest {
1 usage
Run: TSPSolution
SimulatedAnnealingSolverTest
Tests passed: 1 of 1 test – 16 ms
SimulatedAnnealingSolverTest (16 ms
testSimulatedAnnealing 16 ms
C:\Users\vedan\.jdks\openjdk-18.0.2.1\bin\java.exe ...
Process finished with exit code 0


Project
info6205.spring2023.teamproject (1).csv
src
main.java.edu.neu.coe.info6205
model
City
DisjointSet
Edge
solver
AntColonySolver
ChristofidesSolver
SimulatedAnnealingSolver
ThreeOptSolver
TwoOptSolver
util
CircuitUtil
DistanceCalculator
SimulatedAnnealingSolver.java
AntColonySolverTest.java
AntColonySolver.java
SimulatedAnnealingSolverTest.java
TSPSolution.java
AntColonySolver solver = new AntColonySolver(numCities, distances);
@Test
public void testSolve() {
List<Integer> tour = solver.solve();
assertEquals(numCities, tour.size());
assertEquals(expected: 6.0, solver.calculateTourLength(tour), delta: 0.0001);
}
}
Run: TSPSolution
AntColonySolverTest
Tests passed: 1 of 1 test – 16 ms
AntColonySolverTest (test.main 16 ms
testSolve 16 ms
C:\Users\vedan\.jdks\openjdk-18.0.2.1\bin\java.exe -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program
Process finished with exit code 0



Conclusion:

In order to obtain accurate approximations of the solutions to the traveling salesman problem, the effectiveness of the Christofides algorithm and its optimization methods, including tactical methods like random swapping and 2-opt improvement and strategic methods like simulated annealing and ant colony optimization, were investigated in this report.

Our research shows that the Christofides method produced good quality approximations with a worst-case performance guarantee of 1.5 times the optimum solution. The quality of the answers was further improved by the use of optimization techniques including 2-opt improvement and random swapping. Simulated annealing and ant colony optimization were also successful in boosting the quality of the solutions, despite the fact that they required careful parameter tuning.

Using the simulated annealing optimization approach, we discovered that the quality of the solutions was greatly improved by the two-opt enhancement technique. This approach enabled escape from the local optima, enhanced the quality of the solutions, and allowed for a sufficient exploration of the solution space.

Our Java program implementation provides a practical tool for tackling the problem, and further research into optimization methods and their combination may produce even better results.

References:

- "Two-opt heuristic" by C. Helsgaun, in Wiley Encyclopedia of Operations Research and Management Science (2011). This article provides a comprehensive overview of the 2-opt algorithm and its variants, including pseudocode and examples.
- "The traveling salesman problem: a computational study" by D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, in Princeton University Press (2007). This book provides an in-depth study of the TSP and includes a detailed description of the 3-opt algorithm, along with pseudocode and examples.
- "Ant Colony Optimization: A Review" by R. Socha and M. Dorigo, in Swarm Intelligence (2008). This review article provides an overview of the development of ACO over the years, including its variants, applications, and theoretical analysis.
- "Simulated annealing: theory and applications" by S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, in Science (1983). This paper presents the original simulated annealing algorithm and demonstrates its effectiveness on various optimization problems.

Youtube Videos:

- https://www.youtube.com/watch?v=GiDsjIBOVoA&ab_channel=Reducible
- https://www.youtube.com/watch?v=wsEzZ4F_bS4&ab_channel=Rudidev
- https://www.youtube.com/watch?v=oXb2nC-e_EA&t=472s&ab_channel=LearnbyExample