

INFO 6205 Spring 2023 Project

Traveling Salesman Problem Solution Using Christofides Algorithm and other optimization techniques

By:

Chandan Anandachari - 002777668

Natarajan Lekshmi Narayana Pillai - 002766033

Chaman Betrabet - 002784662

Aim:

The aim of this report is to investigate the effectiveness of the Christofides algorithm and its optimization methods, including tactical methods such as random swapping and 2-opt improvement, and strategic methods such as simulated annealing and ant colony optimization. Using these methods, we can obtain high-quality approximate solutions to the traveling salesman problem. This report will also explore the limitations and challenges associated with these methods and provide recommendations for future research in this area.

Approach:

We used IntelliJ IDE to develop a Java program to accomplish the aim of this report. The program was designed to implement the Christofides algorithm and optimize it using tactical and strategic methods. To increase modularity, reusability and readability of the program, we use Object-Oriented Programming (OOP) concepts.

The development of the program happened in several stages. First, we designed the data structures and classes necessary to implement the algorithm, such as Vertex, Edge and Data. Next, we implemented the Christofides algorithm and the optimization methods, including random swapping, 2-opt improvement, simulated annealing, and ant colony optimization. We also developed invariants to make sure the algorithm and its results are accurate.

To test the above algorithms we use a kaggle dataset, which had IDs, latitude and longitude values. From there we calculated the distance using the Haversine formula, and then we used Prims algorithm to derive the MST Value to find out the odd vertices. Using perfect matching algorithm, we match the edges, so that they form the minimum value. After that, the cycle is discovered by using the eulerian graph and cycle, and Hamiltonian cycle to remove redundant vertices. This tour was then forwarded to random swap, 2-opt, simulated annealing and ant colony. With the help of these algorithms, we were able to find the optimal tour value.

We analyzed the program's results and used graphical representations to illustrate the program's accuracy and performance.

Overall, the approach we used in this report combined theoretical analysis, programming, and experimentation to achieve the aim of investigating the effectiveness of the Christofides algorithm and its optimization methods in obtaining high-quality approximate solutions to the traveling salesman problem.

Program:

Data Structures used :

- Lists
- HashMaps
- Customized classes (Data, Vertex and Edges)

Classes used :

- Edge: used to store the vertex v1 and vertex v2
- Eulerian Circuit: Used to calculate the eulerian graph and cycle
- Hungarian: Used to calculate the value of minimum cost edges among the odd vertices
- Odd Vertices: Used to calculate the odd vertices in MST
- Prims: Used to calculate the Minimum Spanning Tree
- Ant Colony: Will help in determining the optimized value using the hamiltonian circuit.
- RandomSwap: Used to calculate the optimized value using random swap algorithm.
- TwoOpt: Used to calculate the tour and optimizing the tour length.
- Simulated Annealing: used to calculate the value of the optimized tour.
- CostTraversal: Used to calculate the cost of the entire tour traversal.
- Haversine Distance: Used to calculate the distance between two longitude and latitude
- ReadingData: Used to read values from the csv file.
- MapPanel: Used to plot out the vertices and the edges
- TSP_GUI: Used to add the panel and buttons into a single frame
- Vertex: Used to send the X, Y coordinate and IDs
- ExecutorMain: Used to execute the main function and all the other optimized values.

Algorithm used:**Christofides:**

The Christofides algorithm is a heuristic algorithm that provides an approximate solution to the traveling salesman problem.

The process by which it works is by

1. Finding a minimum spanning tree
2. Creating a subgraph of odd degree vertices
3. Finding a minimum weight perfect matching
4. Combining the minimum spanning tree and the perfect matching that includes all vertices with an even degree
5. Finding an Eulerian circuit (traversing each edge of the graph exactly once)
6. Transforming it into a Hamiltonian circuit (traversing each vertex of the graph exactly once by skipping previously visited vertices)

This algorithm guarantees that the solution will be at most 1.5 times the optimal solution, with a time complexity of $O(N^2 \log(N))$ for a graph with N vertices.

Random Swapping:

This method of tactical optimization involves randomly switching the order of two cities (vertices) in the solution to see whether it improves it. It is a straightforward strategy that has the potential to enhance the solution but also runs the risk of becoming caught in local optima since the algorithm only explores the neighboring solution by swapping two cities.

2-Opt:

This tactical optimization method involves removing two edges from the solution and reconnecting them in a different way, checking if it results in a better solution. It can be more effective in improving the solution, but is more computationally expensive than random swapping.

Simulated Annealing:

The metaheuristic algorithm for strategic optimization is based on the annealing process in metallurgy. It starts with a high temperature and gradually cools down the system, allowing for more exploration at the beginning and more exploitation at the end. It can be useful in finding local optima, but careful parameter adjustment is necessary.

Ant Colony:

This strategic optimization method is based on the swarm intelligence of ants, who use pheromone trails to locate the shortest route between their nest and a food source. By laying virtual pheromone trails on the graph, the algorithm employs a similar method to determine the shortest path. It can be effective in finding good solutions quickly but require careful tuning of its parameters and can get stuck in local optima.

Invariants:

The graph must be undirected and connected

The minimum spanning tree must be connected

The subgraph of odd degree vertices must have an even number of vertices

The edges in the minimum weight perfect matching must connect odd degree vertices

The combined graph must have even degree vertices

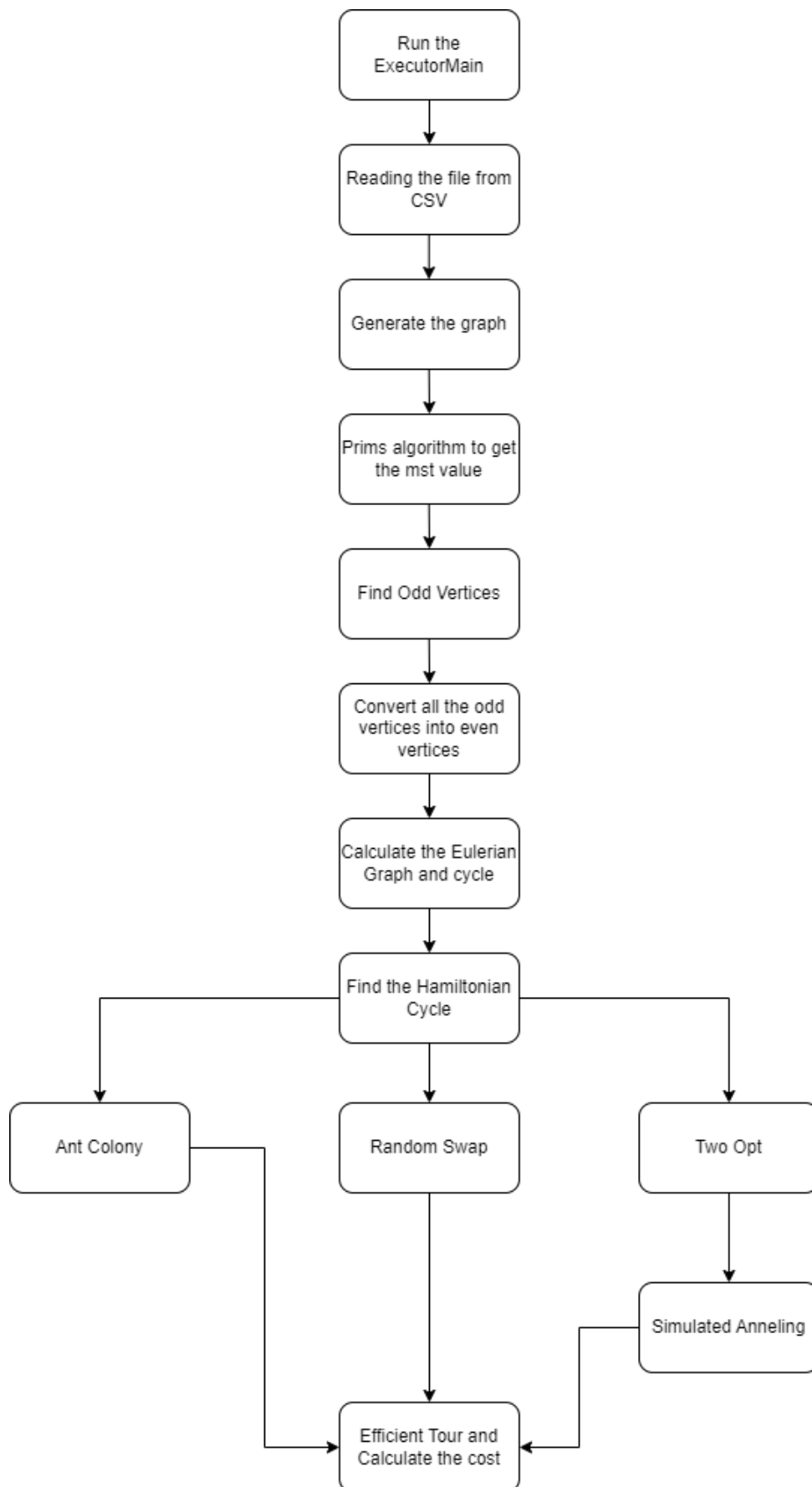
The Eulerian circuit must start and end at the same vertex

Throughout the program, the edges, which are produced by the primality methods used to calculate the Minimum Spanning Tree, will have the same value.

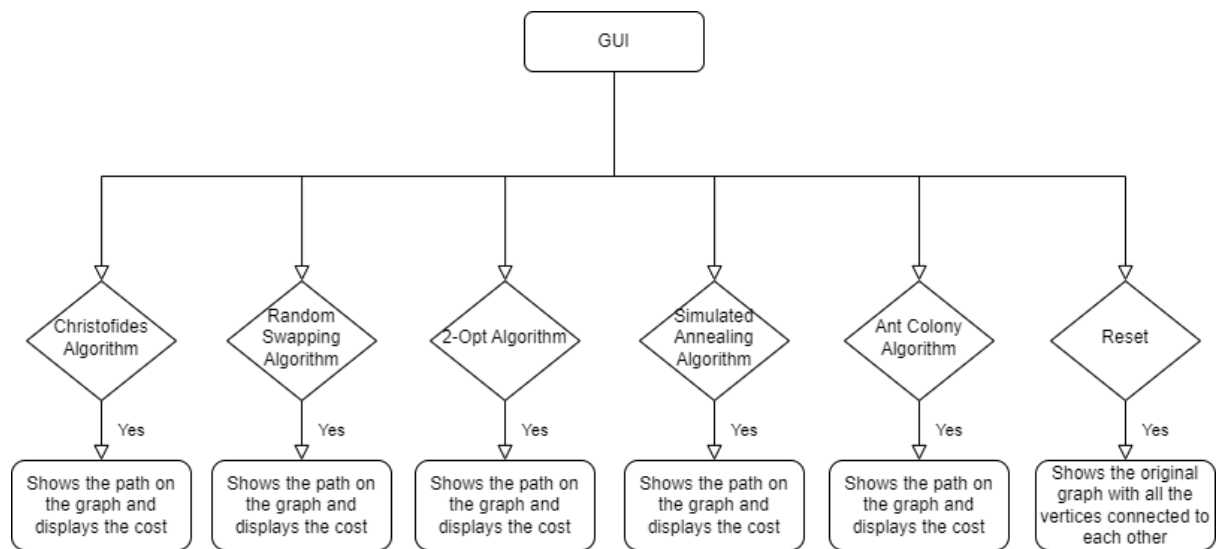
The data that is read from the dataset is moved to the list of type data that is used by the UI to identify the vertices.

Every tour is recorded in a separate list so that it may be used to determine the traversal cost and fed to other algorithms to get the most efficient value.

Flow Chart (Algorithms):



Flow Chart (GUI):

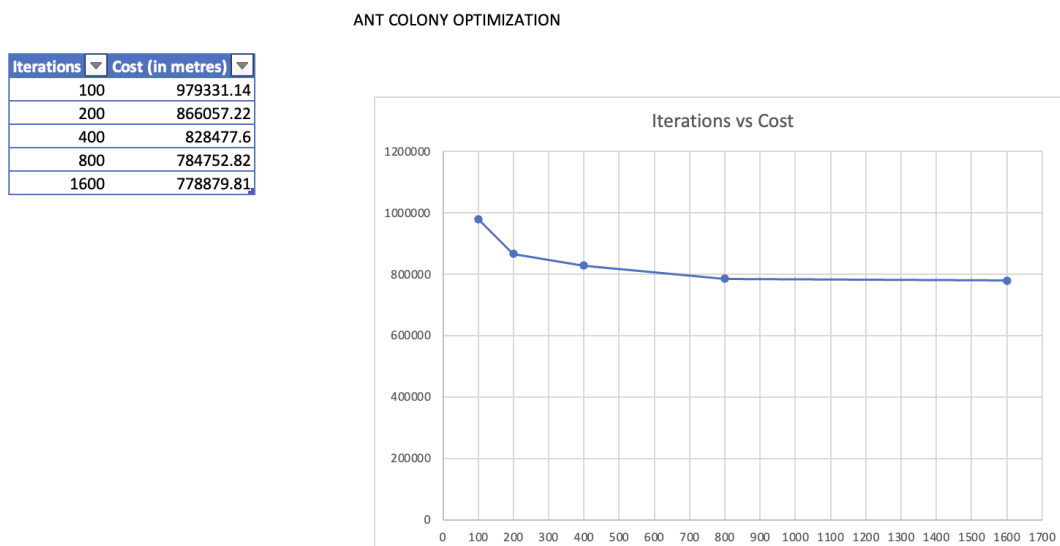


Observations and Graphical Analysis:

Ant colony optimization: We have done our benchmarking for the Ant Colony Optimization, by varying the no. of iterations. We have given the value of the parameters for the formula as follows:

- Alpha (α) – Alpha determines the weight, or priority given to the pheromone trail while the ant chooses the next city. In our benchmarks, we have given a value of 1.0 to alpha
- Beta (β) – Beta determines the weight, or priority given to the heuristic information while the ant chooses the next city. In our benchmarks, we have given a value of 5.0 to beta
- Evaporation is the rate at which the pheromone trail evaporates over time. This parameter is used to avoid the pheromone trail from being too strong and dominating the decision of the ants, leading to suboptimal solutions. A higher value of evaporation means that the pheromone trail evaporates faster. We have given a value of 0.5 for evaporation.
- Q is a parameter that controls the amount of pheromone that an ant deposits on the path. A higher value of Q means that more pheromone is deposited on the path. We have given a value of 500 for Q.

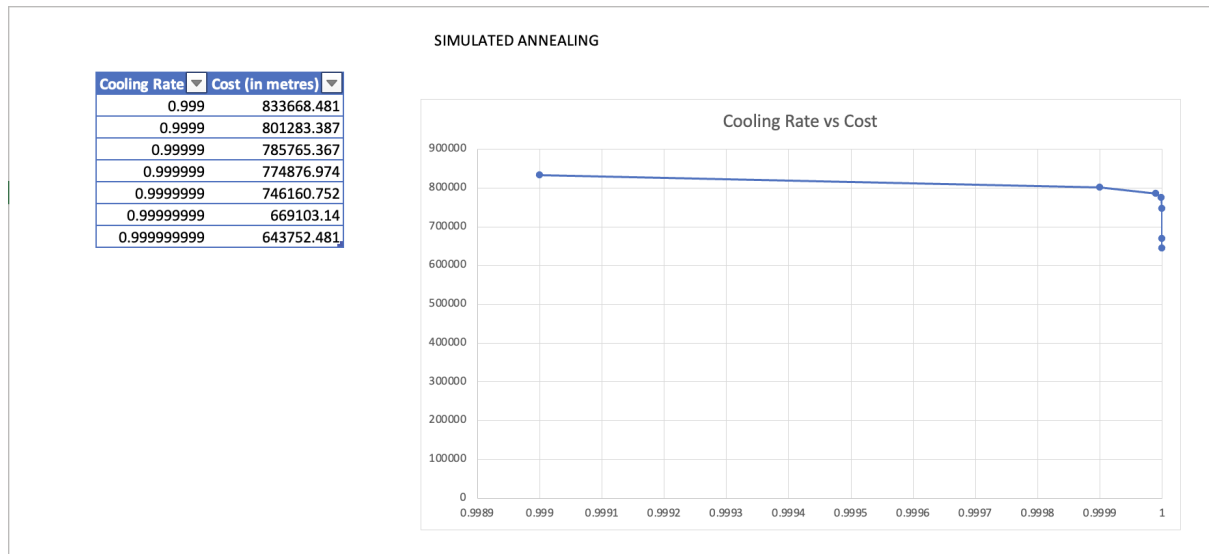
Find below the graph and data showing variation of cost over iterations:



From this, we notice that as we increase the number of iterations, the cost becomes more optimal.

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

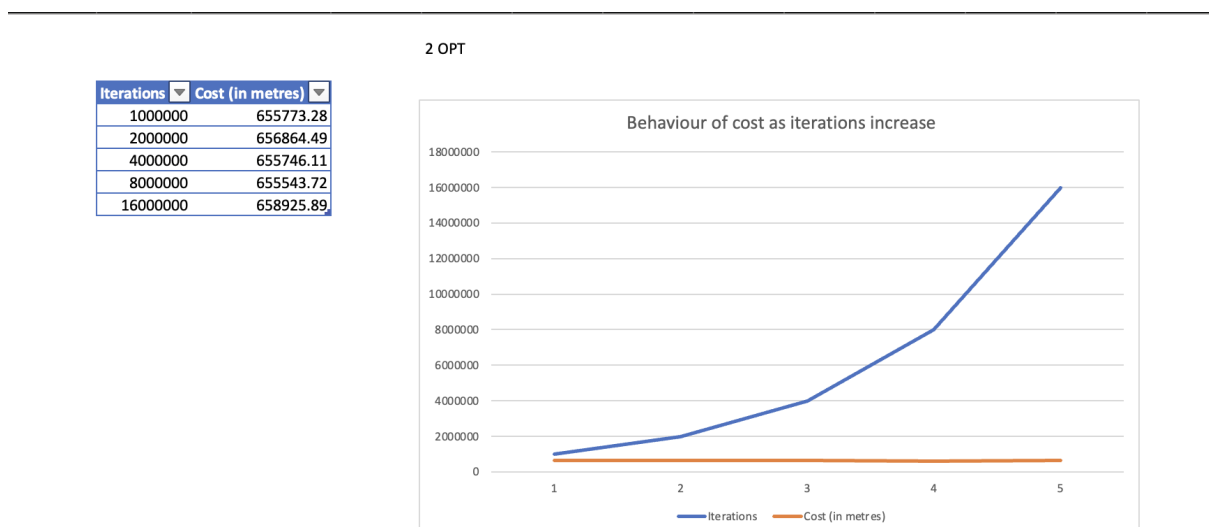
Please find below the data and the graph showing variation of cost over cooling rate:



From this, we notice as we increase the accuracy of the cooling rate, we get a more optimal cost solution in metres.

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

Please find below the data and the graph showing variation of iterations over cost:



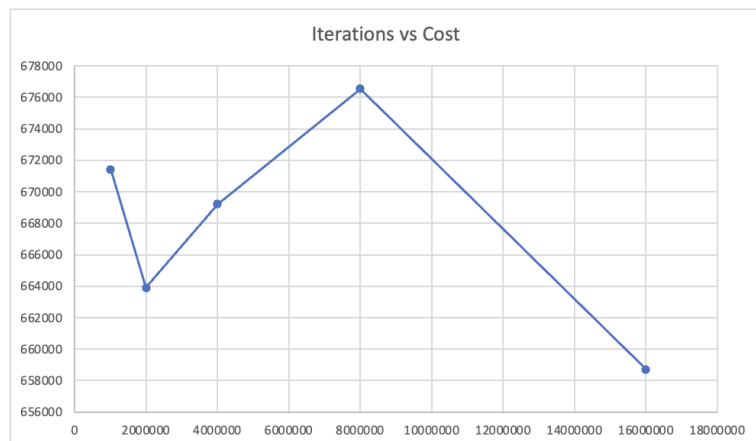
From the data and graph, we notice that as no. of iterations increase, the cost remains constant.

Random swap: For random swap optimization technique, the cost varies slightly over the number of iterations. This is because this technique is randomly swapping two edges and calculating the vertex length adding on to the cost.

Please find below the data and graph for the same:

RANDOM SWAPPING

Iterations	Cost (in metres)
1000000	671432.57
2000000	663891.22
4000000	669215.49
8000000	676543.88
16000000	658719.36



Results and Mathematical Analysis:

We have used a few mathematical formulas for the various algorithms we have implemented. These are explained below:

Haversine formula: We have used the Haversine formula to calculate the distance between the edges of the input file to generate the distance matrix. The Haversine formula is a mathematical formula used in navigation to calculate the distance between two points on the surface of a sphere, such as the Earth. It is commonly used to calculate the distance between two locations given their latitudes and longitudes. The Haversine formula is derived from the law of haversines, which states that for a triangle on the surface of a sphere, the haversine of each angle is equal to the haversine of the sum of the other two angles minus the product of the haversines of those angles. The formula uses the arcsin function to calculate the angle between the two points, and the trigonometric functions sin and cos to convert the latitudes and longitudes to angles. The formula assumes that the Earth is a perfect sphere, which is not entirely accurate, but is generally considered to be accurate enough for most applications.

The Haversine formula is given by:

$$d = 2r \arcsin(\sqrt{\sin^2((\text{lat2}-\text{lat1})/2) + \cos(\text{lat1}) * \cos(\text{lat2}) * \sin^2((\text{lon2}-\text{lon1})/2)})$$

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: One of the optimization techniques we have used to get the optimum TSP solution is Simulated Annealing.

In simulated annealing, the algorithm starts with an initial solution and iteratively generates new candidate solutions by making small random changes to the current solution. The algorithm then evaluates the objective function for each candidate solution and decides whether to accept or reject the new solution based on a probability distribution.

The Metropolis-Hastings algorithm determines the probability distribution for simulated annealing, which is based on the difference between the objective function values of the current and alternative solutions as well as a temperature parameter that drops over time in accordance with a cooling schedule. The cooling schedule is often created to strike a balance between exploration (looking for new solutions) and exploitation (using the existing solution to its fullest extent).

$P(x',x,k,T) = \exp[-(f(x') - f(x))/(kT)]$ 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: One of the optimization techniques we have used to get the optimum TSP solution is the Ant Colony Optimization technique.

The pheromone trail levels for each edge in the graph are calculated as part of the ACO algorithm and updated based on the effectiveness of the ants' solutions. The pheromone level and a heuristic value, which indicates the edge's desirability based on parameters like its length or cost, are then combined to estimate the likelihood that an ant will choose a certain edge.

The formula appears as follows:

$p(i,j)$ 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 α and β 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 Test:

```
import org.junit.Test;
import java.util.ArrayList;
import java.util.List;
import static org.junit.Assert.assertEquals;

public class EulerianCircuit {
    @Test
    public void test1(){
        List<Edge> edgeList=new ArrayList<>();
        edgeList.add(new Edge( edge1: 0, edge2: 1, weight: 10));
        edgeList.add(new Edge( edge1: 1, edge2: 3, weight: 25));
        edgeList.add(new Edge( edge1: 3, edge2: 2, weight: 30));
        edgeList.add(new Edge( edge1: 2, edge2: 0, weight: 15));

        MSTAlgorithms.EulerianCircuit eulerianCircuit=new MSTAlgorithms.EulerianCircuit(edgeList, V: 4);
        List<Integer> circuit=eulerianCircuit.EulerianCircuitAlgorithm();

        assertEquals(edgeList.size(),circuit.size(), delta: 1);
    }
}
```

Run: EulerianCircuit

Tests passed: 3 of 3 tests - 2 ms

test1 2 ms
test2 0 ms
test3 0 ms

Process finished with exit code 0

```
import java.util.List;
import static org.junit.Assert.assertEquals;

public class HungarianTest {
    @Test
    public void test1(){
        double graph[][] = {{0, 10, 15, 20},
                             {10, 0, 35, 25},
                             {15, 35, 0, 30},
                             {20, 25, 30, 0}};

        Hungarian hungarian=new Hungarian(graph);
        int a[]=hungarian.execute();

        assertEquals( expected: 4,a.length);
    }

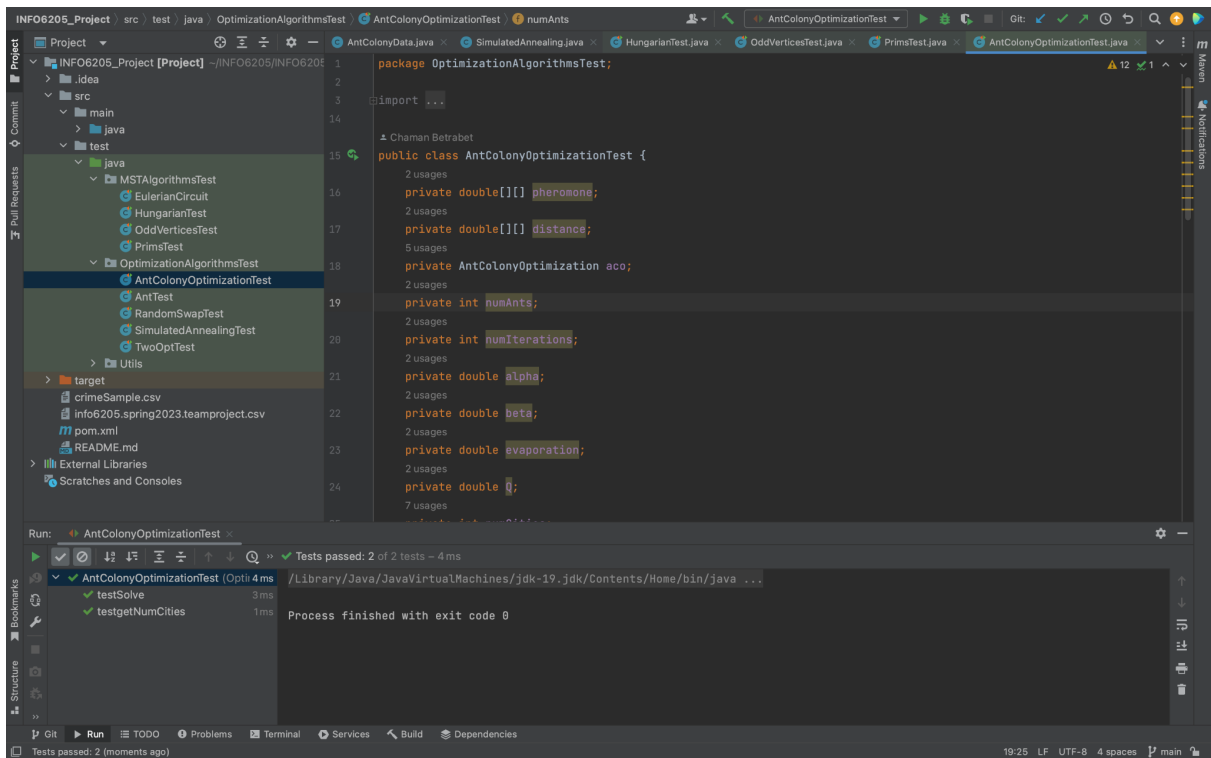
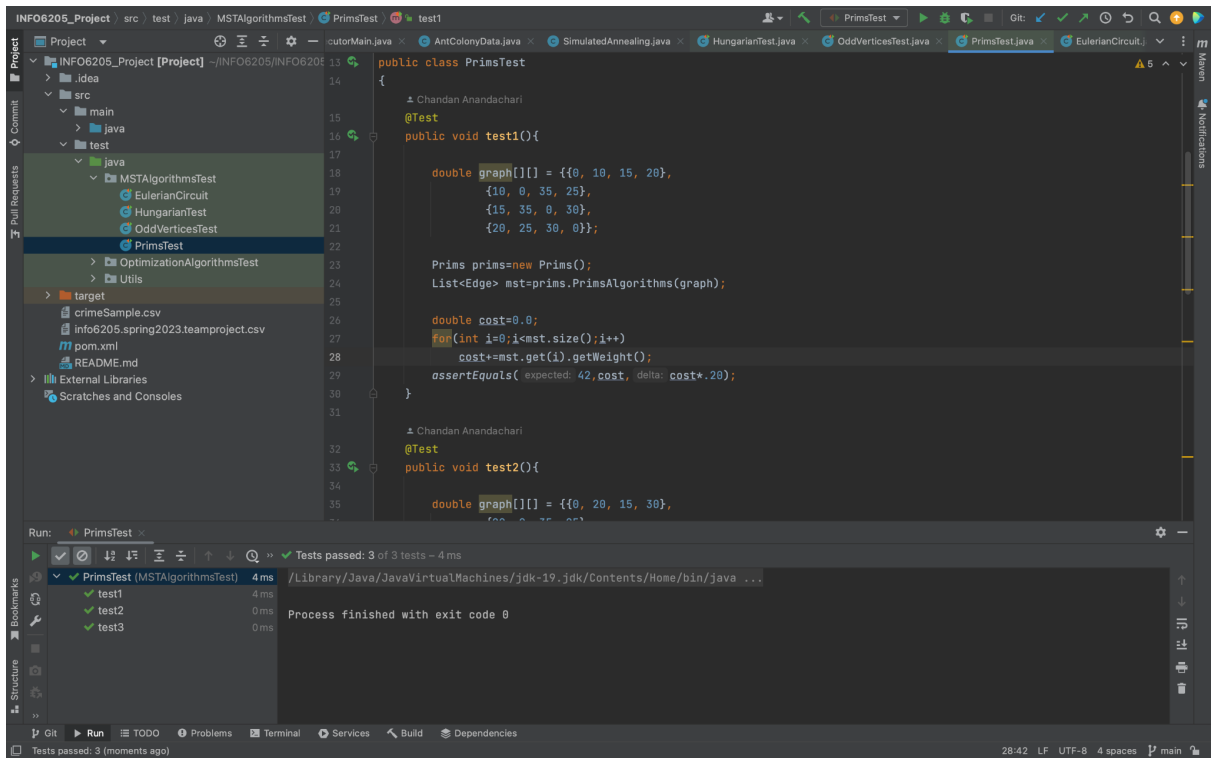
    @Test
    public void test2(){
    }
}
```

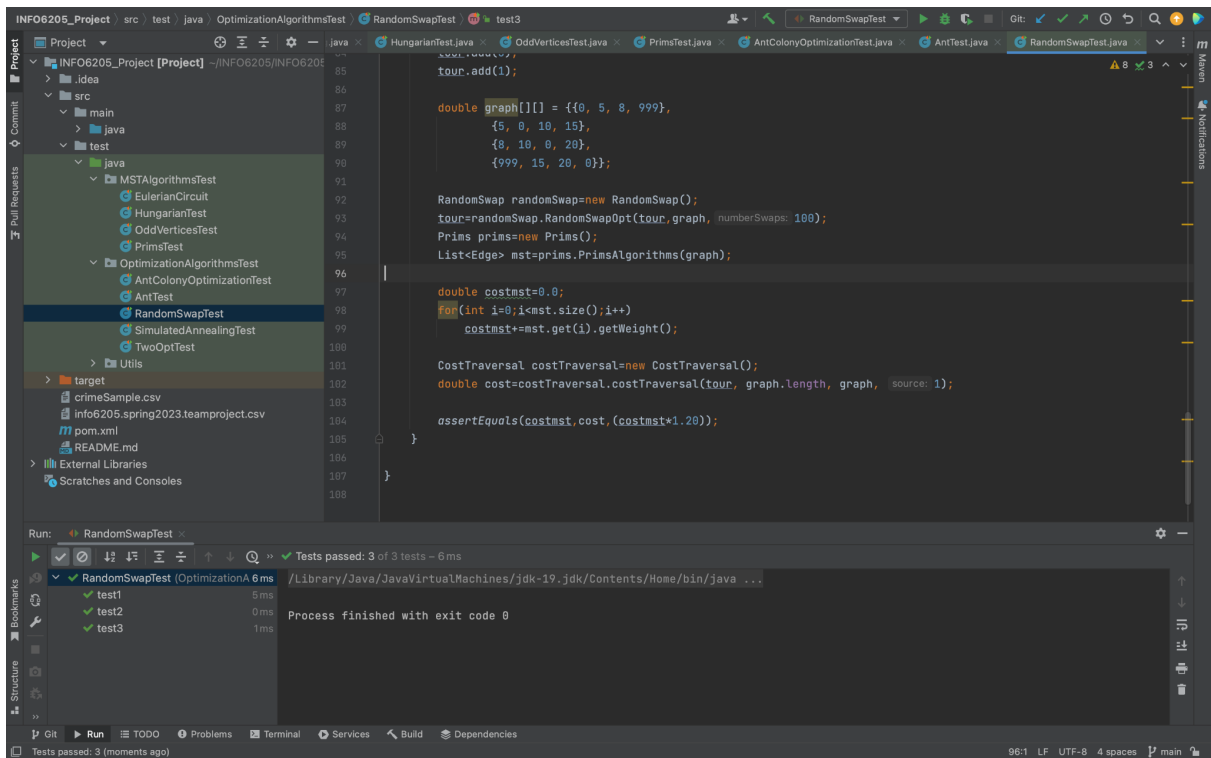
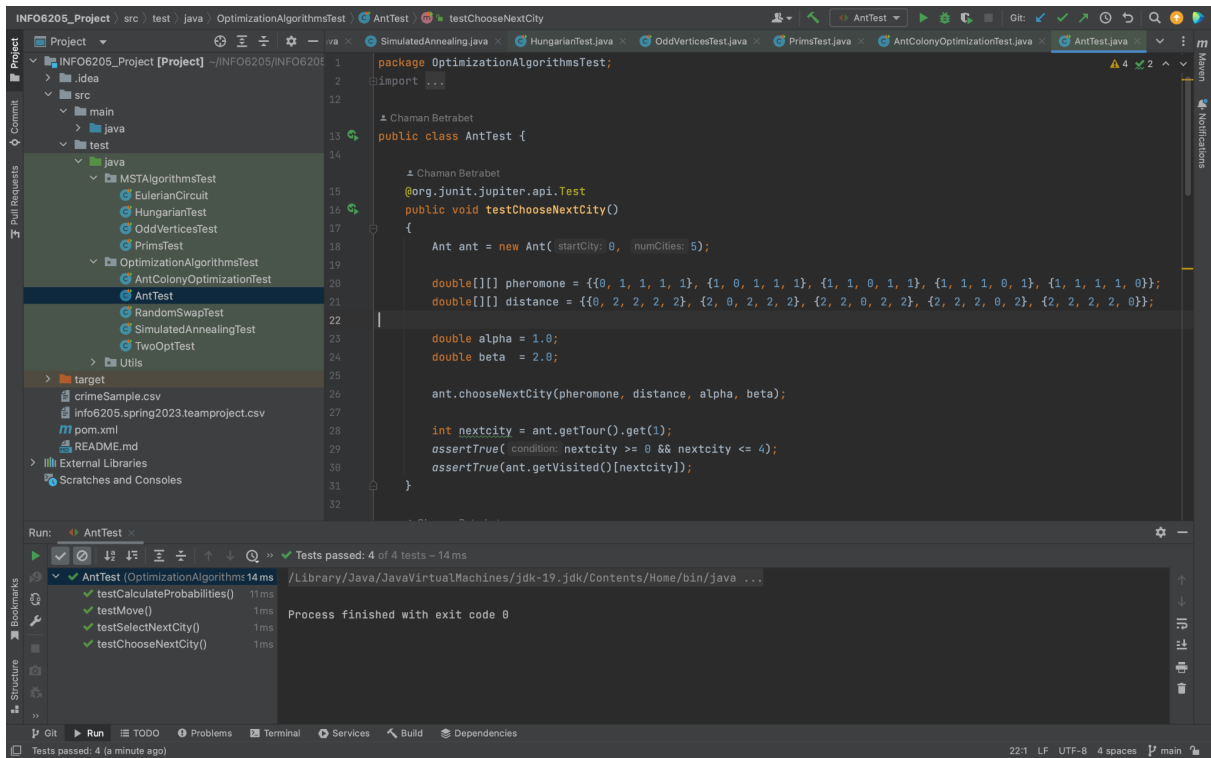
Run: HungarianTest

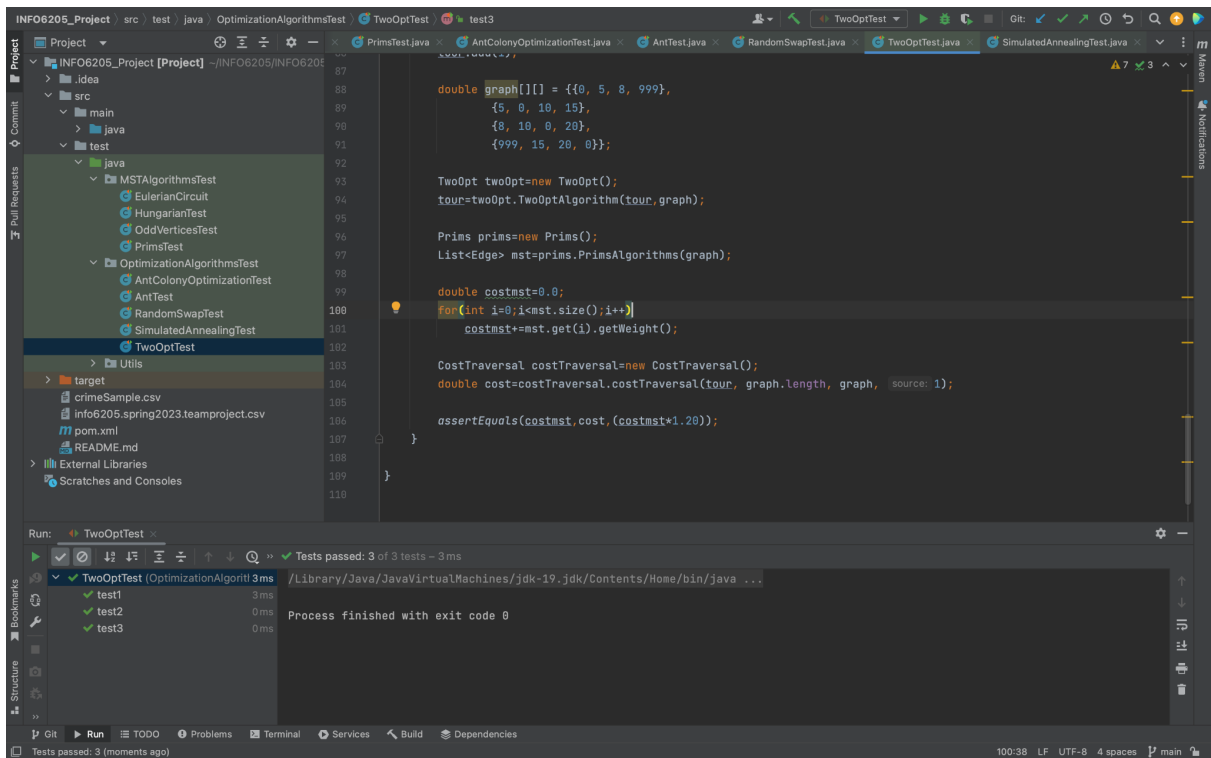
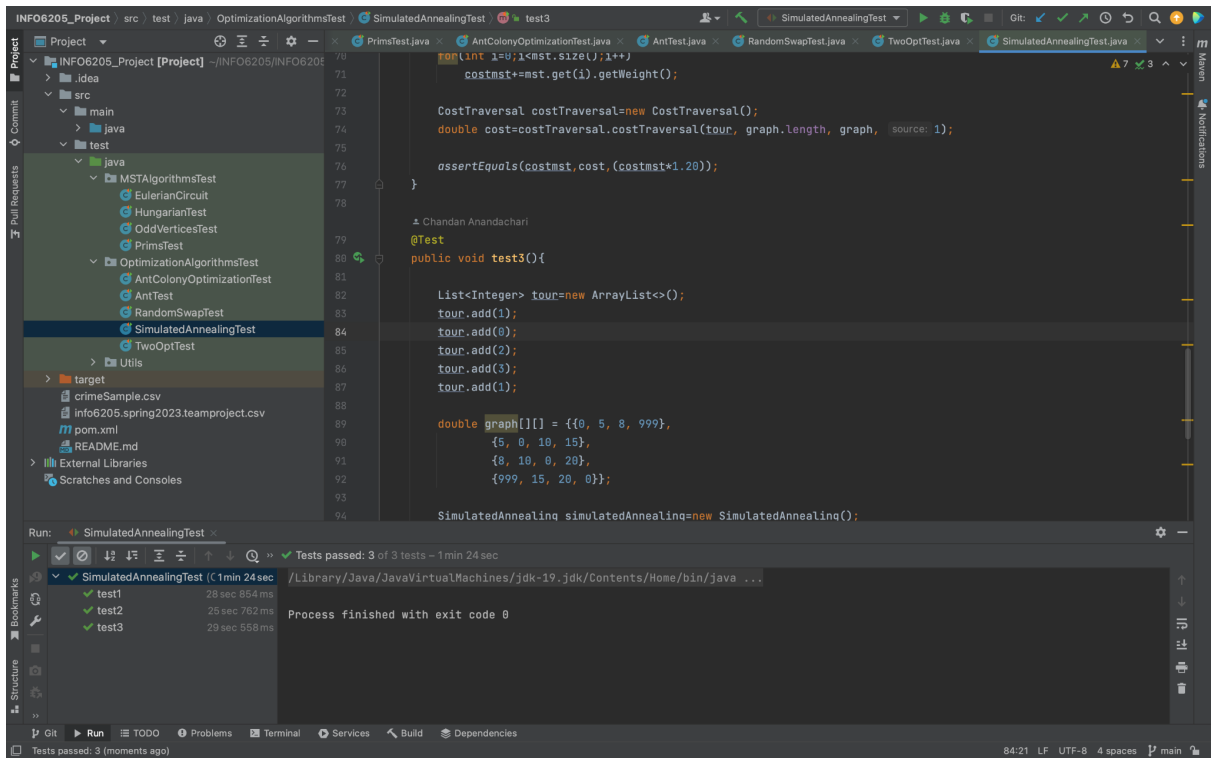
Tests passed: 4 of 4 tests - 1 ms

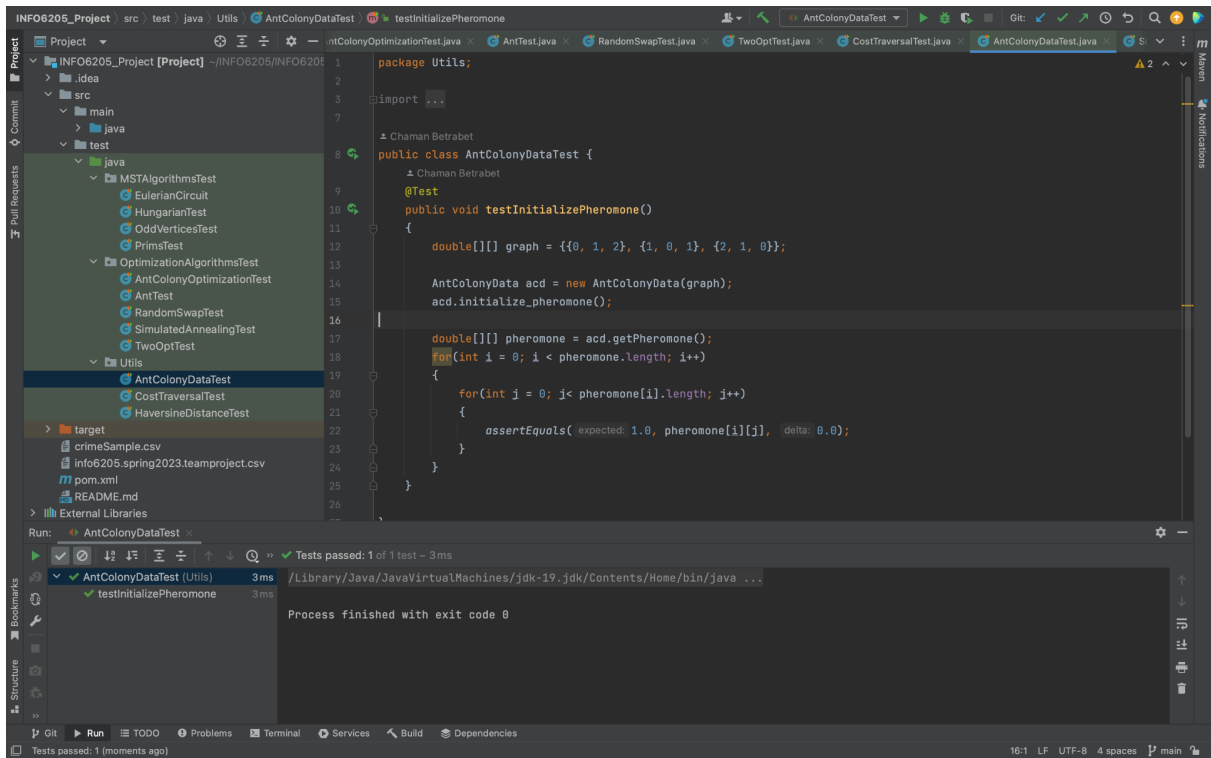
test1 1 ms
test2 0 ms
test3 0 ms
test4 0 ms

Process finished with exit code 0









Conclusion:

In this report, we investigated the effectiveness of the Christofides algorithm and its optimization methods, including tactical methods such as random swapping and 2-opt improvement, and strategic methods such as simulated annealing and ant colony optimization, in obtaining high-quality approximate solutions to the traveling salesman problem.

Based on our findings, the Christofides method produced high-quality approximate solutions with a worst-case performance guarantee of 1.5 times the optimal solution. The addition of optimization methods such as random swapping and 2-opt improvement further improved the solution quality. Simulated annealing and ant colony optimization were also effective in improving the solution quality but required careful parameter tuning.

In order to assure the program's accuracy, we used object oriented programming principles and unit tests to develop the Christofides algorithm and its optimization methods in Java. We generated datasets of various sizes to test the performance and accuracy of the program.

We discovered that the 2-opt improvement technique within the simulated annealing optimization method provided significant improvements in the solution quality. This approach was able to escape from the local optima and enhance the solution quality while still allowing for sufficient exploration of the solution space.

In conclusion, our program implementation in Java provides a practical tool for solving the problem, and future research optimization methods and their combination to achieve even better results.

References:

1. Reducible. (2022, July 26). The Traveling Salesman Problem: When Good Enough Beats Perfect [Video]. Retrieved from https://www.youtube.com/watch?v=GiDsjiBOVoA&ab_channel=Reducible
2. de Oliveira da Costa, P. R., Rhuggenaath, J., Zhang, Y., & Akcay, A. (2021). Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. IEEE Transactions on Neural Networks and Learning Systems, 32(4), 1698-1709. <http://proceedings.mlr.press/v129/costa20a/costa20a.pdf>
3. MathWorks. (n.d.). Minimization Using Simulated Annealing Algorithm. Retrieved April 16, 2023, from <https://www.mathworks.com/help/gads/simulated-annealing-examples.html>
4. Rudidev. (2020, November 2). 2-opt explained | TSP optimization tutorial and visualization [Video]. Retrieved from https://www.youtube.com/watch?v=wsEzZ4F_bS4&ab_channel=Rudidev
5. LearnByExample. (2021, May 18). Solving the Travelling Salesman Problem using Ant Colony Optimization [Video]. Retrieved from https://www.youtube.com/watch?v=oXb2nC-e_EA&t=472s&ab_channel=LearnByExample