Program Structures and Algorithms

INFO6205 Final Project

Under the Supervision of

Prof. Robin Hillyard

Team Members

Amretasre Rengarajan Thiruvengadam – 002762670

Priyal Vimal Gudhka – 002747680

Surbhi Soni – 002969578

Introduction

Aim

The TSP is a classic optimization problem that involves finding the shortest possible route that passes through all given points and returns to the starting point and the objective of the project is to solve the TSP using a dataset that contains CrimeId, Latitude, and Longitude using Christofides algorithm. This algorithm is an efficient algorithm used to find the approximate solution for the traveling salesman problem (TSP). To further optimize the solution, various Strategic and Tactical Optimization methods are used. The Strategic optimization methods include simulated annealing, ant colony optimization, genetic algorithms and Tactical optimization methods include random swapping, 2 opt and/or 3 opt. Out of which 2-Opt, 3-Opt, Simulated annealing and ant colony optimization is being utilized. By combining the Christofides algorithm with various Strategic and Tactical Optimization methods, the aim is to find the most efficient route possible that passes through all given points and returns to the starting point. The final solution is expected to significantly reduce the overall distance traveled and improve the efficiency of the route.

The Repository link for the solution is: Team 20 GitHub URL

Approach

The data set is extracted from a csv file and is converted into a Vertex class that includes the crime ID (used the last five hexadecimal values of the crime id) as the vertex ID and latitude and longitude as X_ver and Y_ver, respectively. This conversion facilitates the preparation of a distance matrix.

Once the data set is converted into Vertex and added to a List, the distance matrix is prepared. The Minimum Spanning Tree (MST) is created using the Prims algorithm, which selects the edge with the minimum weight and adds it to the growing tree until all vertices are connected. The Prims algorithm is widely used to find the MST in a graph.

Next, the Hierholzer Algorithm is utilized to form the Euler tour. This algorithm creates a cycle that passes through every edge exactly once. The Hierholzer Algorithm is efficient and is widely used to find the Euler tour in a graph.

After the Euler tour is formed, any duplicate points from the final tour are removed to optimize the tour by reducing the overall distance traveled. Finally, the optimized tour is utilized as an input for optimization techniques through a switch case. Based on the parameter passed to the switch case the optimization technique is decided and the input is further optimized.

Once the best total distance is calculated the results are displayed in the form of a graph using JPanel form Java Swing and the graph utilizes the Java AWT for connecting the vertices.

Program

Data Structure

The data structures that are used in the code are as follows:

1. List for calculating the final tour

2. Array list

3. 2 D arrays for the distance matrix

4. Linked lists used in Hierholzer algorithm to form the Euler tour

Classes

The classes defined in the code are as follows:

1. **TSPMainProgram**

   This class is the main entry point for the program and contains the logic for extracting the crime ID, latitude, and longitude data from the dataset. It also calls other classes to perform the TSP algorithm, and once the final tour is computed, it uses the TSPVisualization class to plot the graph and display the final tour.

2. **ChristofidesTour**

   This class is responsible for computing the final tour using the Christofides algorithm. It returns the crime IDs of the final tour and the tour cost.

3. **PrimsAlgorithm**

   This class takes the list of vertices from the Christofides tour and sorts them to form the minimum spanning tree. This is a required step for the Christofides algorithm.

4. **HierholzerAlgorithm**

   With the minimum spanning tree, this class creates the euler tour required for the Christofides algorithm.

5. **TwoOptOptimization**

   Another optimisation class that uses the two opt optimization algorithm to optimize the Christofides tour. This class has methods like three opt, namely a method to calculate the distance and the method to perform two opt swaps.

6. **ThreeOptOptimization**

   Used for optimizing the christofides final tour using the three opt optimisation technique. It contains the method to calculate the total distance and the method to swap the vertices according to the distance.

7. **SimulatedAnnealingOptimization**

   This class implements the Simulated Annealing optimization algorithm to optimize the Christofides tour. It holds properties such as current distance, temperature, and cooling rate, and the methods that can optimize the Christofides tour.

8. **AntColonyOptimization**

   This class implements the Ant Colony Optimization algorithm to optimize the Christofides tour. It has several properties, such as the number of ants, alpha (controlling the influence of pheromone), beta (controlling the influence of distance), rho (pheromone evaporation rate), pheromone deposit factor, and the pheromone matrix. It also contains the methods required to optimize the Christofides tour using the Ant Colony Optimization algorithm.

9. **Benchmark**

   The main program uses this class to find the execution time for each of the algorithms.

10. **Edge**

    This class represents an edge in the graph, with methods to create the edges between vertices and retrieve the weight or distances between them. It also gives us the owner and child vertex of a particular edge.

11. **GraphPanel**

    The graph panel class is utilized to visualize the final tour using Java Swing and Java AWT. It also contains the parameters to change the visuals of the graph.

12. **DuplicateHelper**

    The duplicateHelper class is used to remove the duplicate vertices from the tour. The run method in this class is used each time the Christofides algorithm forms the final tour.

13. **TSPVisualization**

    The TSPVisualization uses the GraphPanel to print the edges and vertices of the final tour in the JPanel.

14. **Vertex**

    The vertex has properties such as the vertex id which is the crime id, the latitude and longitude as the xVar and yVar respectively and an array list of vertices to keep track of the connected vertices and edge.

15. **VertexComparator**

The VertexComparator class is mainly used to compare the vertices with respect to the edge weight.

Test Classes

The below classes contain the unit testing code for the optimization techniques. It has various methods for checking if all the vertices are present and if the distance optimized using different optimization techniques is less than the distance calculated using Christofides algorithm.

1. **TwoOptOptimizationTest**

2. **ThreeOptOptimizationTest**

3. **SimulatedAnnealingOptimizationTest**

4. **AntColonyOptimizationTest**

Algorithm

1. **Christofides Algorithm**

1.

The Traveling Salesman issue (TSP), a well-known optimization issue in computer science, may be solved using the Christofides method. Finding the quickest route that stops in each city precisely once before returning to the originating city is known as the TSP. The Christofides algorithm is a heuristic algorithm, which means it is not always guaranteed to offer the ideal answer but can provide a good approximation in an acceptable period. The algorithm follows:

1. Create the smallest spanning tree T of a full graph G with n vertices and non-negative edge weights.
2. Establish a minimal weight perfect matching M on the set of vertices in T with an odd degree. A greedy algorithm can be effectively used to do this.
3. T and M's edges may be combined to create the linked graph H, which has each vertex with an even degree.
4. In H, identify a Eulerian circuit—a path that reaches each edge precisely once.
5. The repeated vertices, or vertices that have previously been visited, must be skipped to transform the Eulerian circuit into a Hamiltonian circuit.
6. A tour that visits each vertex precisely once before returning to the beginning vertex is the resultant Hamiltonian circuit.
7. Calculate the total weight of the tour and return it as the output.

The running time of the Christofides algorithm is O (n^2 log n) for a graph with n vertices.

2. Tactical Optimization

a. Two Opt Optimization Algorithm

A local search technique termed two-opt optimization is being used to enhance the result of the Christofides algorithm. The key idea of Two Opt optimization is to link the tour's fragments in such a way as to produce a shorter tour by first iteratively removing two edges from the tour. Up until there are no more improvements left to be made, this procedure is repeated. The technique operates by randomly selecting two edges and then inverting the vertices' locations between the two edges. To generate a shorter tour, the two newly created edges are tested to determine whether they can be switched with the original edges in any way. The process is repeated until no further advancements can be obtained and the new edges are swapped with the original edges if they result in a shorter trip. The Two Opt follows the following steps:

1. Begin with the first Christofides tour.
2. Choose two edges in the tour, such as edges (i,j) and (k,l), wherein i does not equal k and j does not equal l.
3. Calculate the tour's overall distance with the edges (i, j) and (k, l) switched.
4. Swap the edges and update the tour distance if the new tour's overall length is shorter than the old one.
5. Until no further advancements can be achieved or a predefined number of iterations have been made, repeat steps 2-4.

Two Opt optimization algorithm has an $O(n^3)$ time complexity, where n is the number of possible network vertices. This makes it a worthwhile option for small to medium-sized networks, but for bigger graphs, it may become highly inefficient.

b. Three Opt Optimization Algorithm

Similar to Two Opt, Three opt begins with an initial tour and then switches three edges at a time in an effort to discover the better tour. The steps in Three Opt are as follows:

1. Choose the tour's A-B, C-D, and E-F edges such that they form a loop.
2. There are eight alternative ways to join the edges together while still having a legitimate tour. Determine the overall distance for each of these eight arrangements.
3. Choose the arrangement that reduces the distance traveled overall.
4. For each set of three edges in the tour that is possible, repeat the procedures above.
5. Return the final tour and end the algorithm if there is no room for improvement.

The size of the input graph or the number of vertices in the graph affects how time-consuming the three opt optimization process is. The temporal complexity is typically $O(n^3)$, where n is the number of vertices. The implementation and particulars of the problem being solved may have an impact on the real-time complexity. The technique is often only used for small to medium-sized issues since it might take a long time to run for big values of n.

3. Strategic Optimization

a. Simulated Annealing Optimization Algorithm

A probabilistic optimization approach called Simulated Annealing is used to determine a cost function's global minimum or maximum. It is modeled after the physical annealing of solids, which involves heating the substance to high temperatures and then gradually cooling it to a more stable condition. These steps can be used to develop the Simulated Annealing Optimization Algorithm for the TSP problem:

1. Calculate the cost of an initial tour T first.
2. Pick i and j as two random cities.
3. Reverse the order of the cities between i and j in T to create a new tour T'.
4. Compute a T' cost calculation.
5. Accept the new tour T' as the current tour T if the cost of T' is less than the cost of T.
6. Calculate the acceptance probability using the cost difference and a temperature parameter if the cost of T' is larger than the cost of T.
7. Accept the new trip T' as the old tour T with a probability proportional to the acceptance probability.
8. Till a stopping requirement, such as a maximum number of iterations or a minimum temperature, is satisfied, repeat steps 2 through 7.

The acceptance probability is calculated using the below formula: P = exp (-delta / temperature) where P is the acceptance probability, delta is the change in the objective function value (in this case, the total distance) between the current and new solution, and temperature is the current temperature of the system. The TSP issue may sometimes be solved using the Simulated Annealing Optimization Algorithm; however, the choice of beginning temperature, cooling schedule, and acceptance probability function can have a significant impact on how well the algorithm performs.

b. Ant Colony Optimization Algorithm

Ant Colony Optimization is a population-based algorithm, meaning that multiple solutions are constructed simultaneously. The algorithm can be customized with several parameters, including the number of ants, the pheromone evaporation rate, and the importance of the pheromone trail versus the distance in the probabilistic rule.

The basic idea of ACO is to model the problem as a graph, where the nodes represent the cities or customers, and the edges represent the routes between them. The ants are then simulated as they traverse the graph, depositing pheromones along the edges that they travel.

The pheromones serve as a form of communication among the ants, as they can follow the paths with higher pheromone levels. Over time, the pheromone trails will be reinforced on the shortest paths, leading to the convergence of the solution towards the optimal solution.

Invariants

1. **Christofides Algorithm**

   a. **Minimum Spanning Tree**

   The algorithm starts by constructing a minimum spanning tree (MST) of the input graph. The MST is a tree that connects all vertices with the minimum possible total edge weight.

   b. **Matching**

   After constructing the MST, the algorithm creates a matching (a set of edges with no common vertices) from the vertices with odd degree in the MST. This matching is guaranteed to have the minimum possible total weight.

   c. **Eulerian Circuit**

   The algorithm then creates a Eulerian circuit (a circuit that visits each edge exactly once) by combining the MST and the matching. This circuit may not be the shortest Hamiltonian circuit, but it is guaranteed to exist.

   d. **Shortest Hamiltonian Circuit**

   Finally, the algorithm improves the Eulerian circuit to a Hamiltonian circuit (a circuit that visits each vertex exactly once) by shortcutting the circuit. The shortcutting process replaces some sub paths of the circuit with shorter paths, resulting in a shorter Hamiltonian circuit. This process is repeated until no more improvements can be made, at which point the algorithm outputs the resulting Hamiltonian circuit.

2. **Two Opt Optimization**

   a. The start and end vertices of the partial tour being reversed should remain the same throughout the algorithm.
   b. The tour should always remain valid. Specifically, each vertex should be visited only once, and the resulting path should form a closed loop.

3. **Three Opt Optimization**

   a. The edges must form a cycle that visits all vertices exactly once.
   b. The three edges being considered for swapping must form a valid triangle.
   c. The three edges being removed must not be adjacent to each other.
   d. The new edges being added must not create a cycle that visits any vertex more than once.
   e. The new edges being added must form a cycle that visits all vertices exactly once.
   f. The newly added edges must create a cycle that goes over each vertex precisely once.

4. **Ant Colony Optimization**

   a. **Pheromone Trails:** The ants in ACO leave behind pheromone trails as they look for food. The quality of the solution identified is inversely correlated with the quantity of pheromone placed on a certain edge. As a result, additional ants are drawn to edges with greater pheromone concentrations, increasing the likelihood that they would choose better solutions. This creates a positive feedback cycle.

b. **Iterations:** Until a stopping requirement is satisfied, ACO builds and enhances potential solutions repeatedly. The algorithm can be stopped after a predetermined number of iterations or after a predetermined period. The number of iterations and initial solution quality both affect how well the final solution turns out.

Flow Chart

Below is the flow chart indicating the flow of our program used for calculating optimized distance using various optimization techniques.

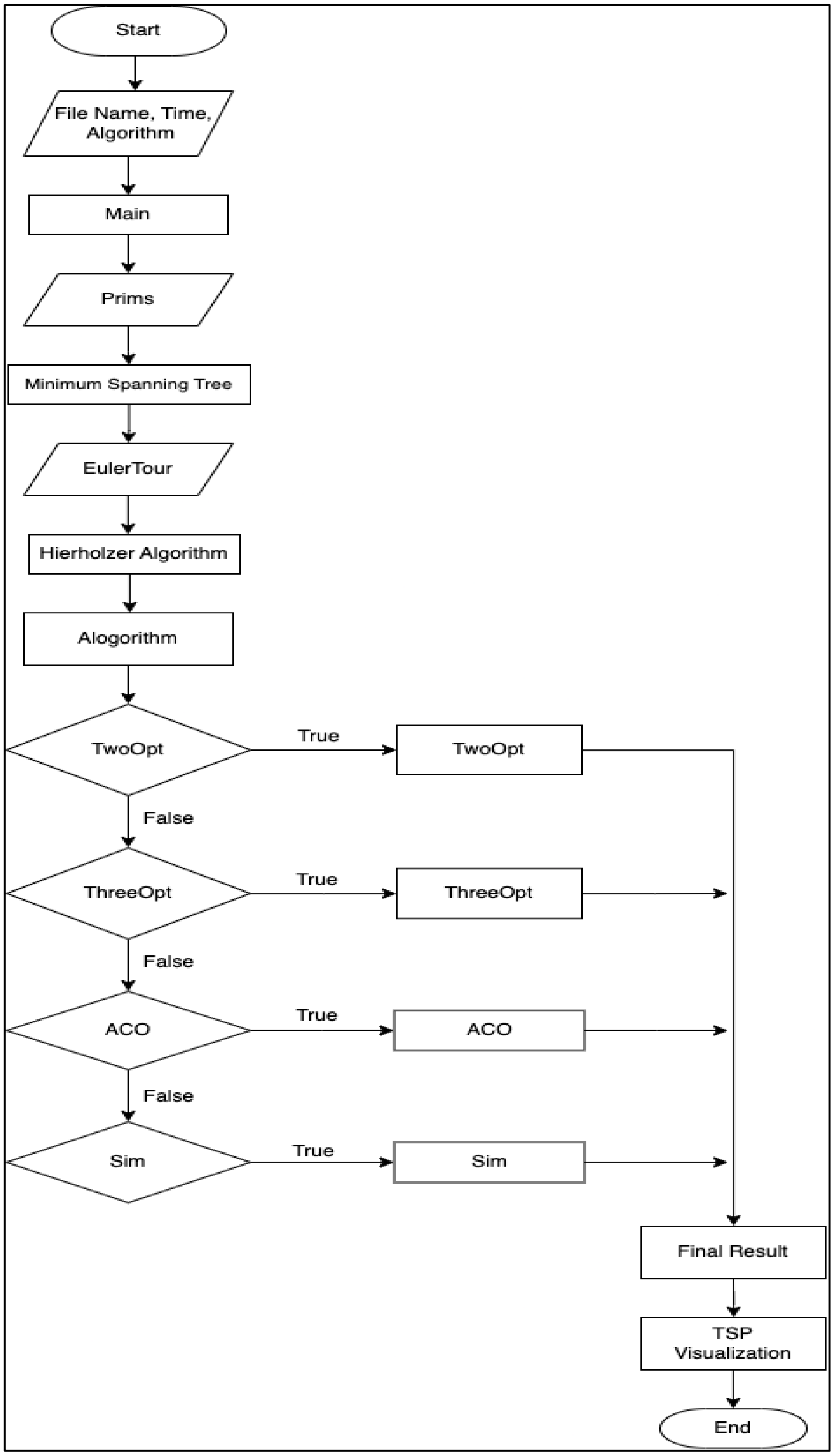
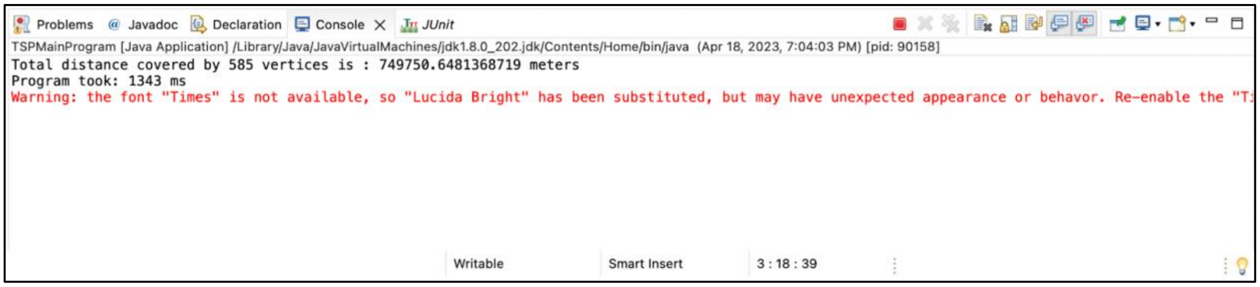


Fig: Flow Chart

Observations

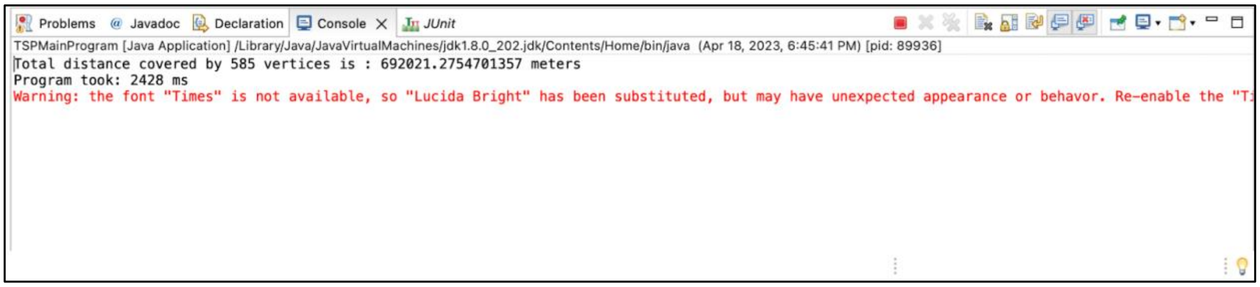
1. **Output of Christofides Algorithm**



```
TSPMainProgram [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java  (Apr 18, 2023, 7:04:03 PM) [pid: 90158]
Total distance covered by 585 vertices is : 749750.6481368719 meters
Program took: 1343 ms
Warning: the font "Times" is not available, so "Lucida Bright" has been substituted, but may have unexpected appearance or behavor. Re-enable the "Ti
```

Fig: Output of Christofides Algorithm

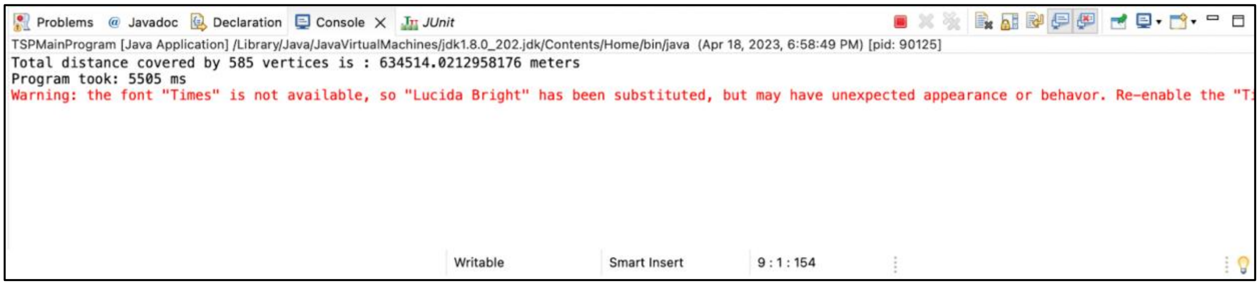
2. **Output of 2-Opt Optimization**



```
TSPMainProgram [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java  (Apr 18, 2023, 6:45:41 PM) [pid: 89936]
Total distance covered by 585 vertices is : 692021.2754701357 meters
Program took: 2428 ms
Warning: the font "Times" is not available, so "Lucida Bright" has been substituted, but may have unexpected appearance or behavor. Re-enable the "Ti
```

Fig: Output of 2-Opt

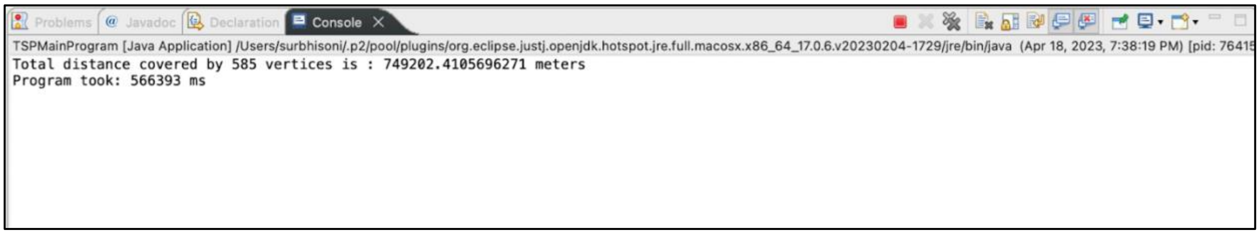
3. **Output of 3-Opt Optimization**



```
TSPMainProgram [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_202.jdk/Contents/Home/bin/java  (Apr 18, 2023, 6:58:49 PM) [pid: 90125]
Total distance covered by 585 vertices is : 634514.0212958176 meters
Program took: 5505 ms
Warning: the font "Times" is not available, so "Lucida Bright" has been substituted, but may have unexpected appearance or behavor. Re-enable the "Ti
```

Fig: Output of 3-Opt

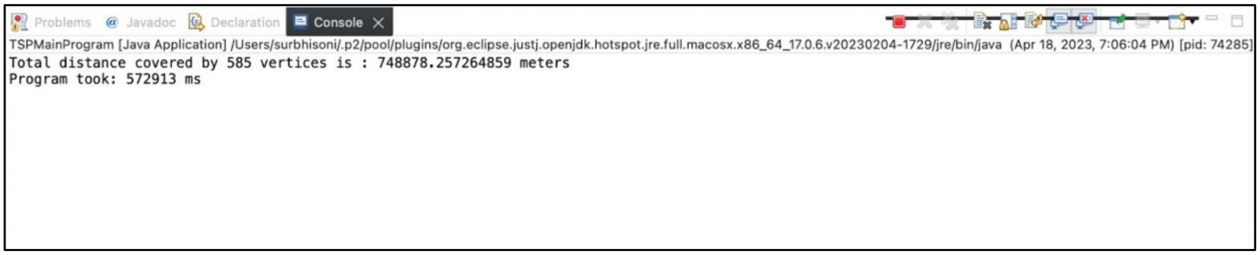
4. **Output of Simulated Annealing Optimization**



```
TSPMainProgram [Java Application] /Users/surbhisoni/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.6.v20230204-1729/jre/bin/java  (Apr 18, 2023, 7:38:19 PM) [pid: 76415
Total distance covered by 585 vertices is : 749202.4105696271 meters
Program took: 566393 ms
```

Fig: Output of Simulated Annealing

5. Output of Ant Colony Optimization



```
TSPMainProgram [Java Application] /Users/surbhisoni/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.6.v20230204-1729/jre/bin/java  (Apr 18, 2023, 7:06:04 PM) [pid: 74285]
Total distance covered by 585 vertices is : 748878.257264859 meters
Program took: 572913 ms
```

Fig: Output of Ant Colony

Graphical Analysis

The table represents the time taken by each of the algorithm:

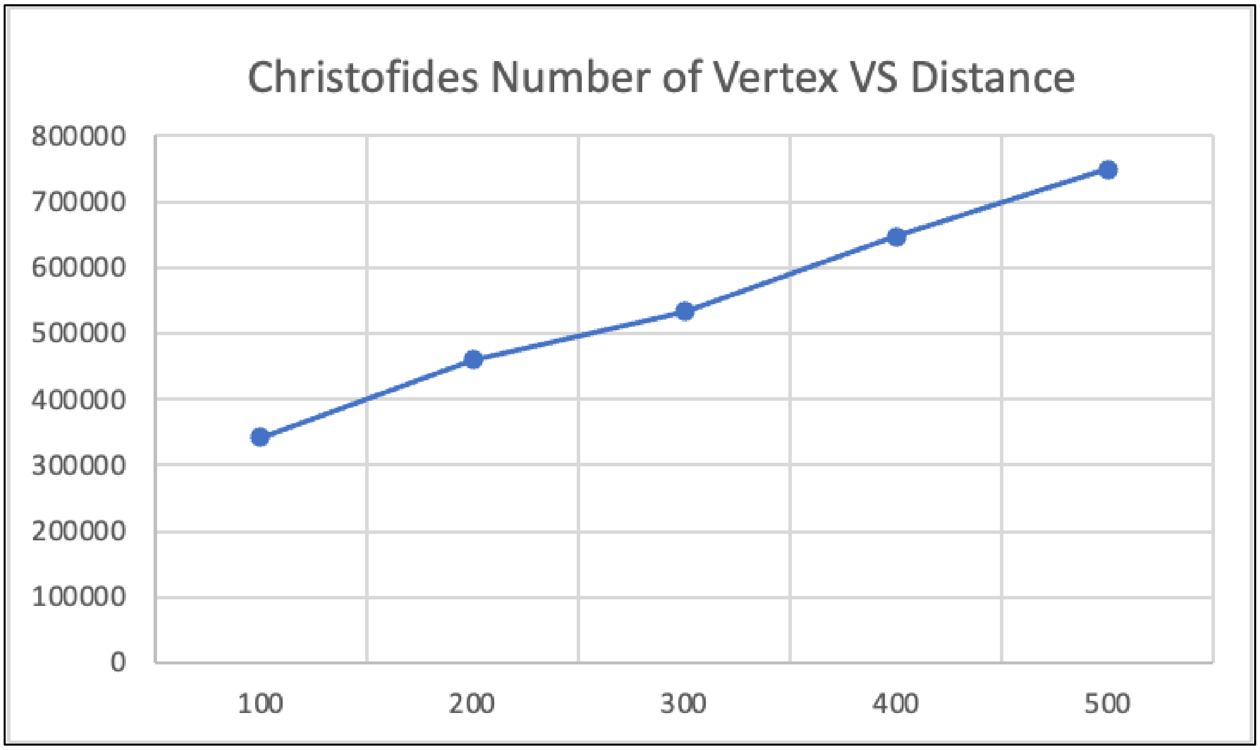
Number of Vertex	Christofides Algorithm	Two Opt Optimization	Three Opt Optimization	Simulated Annealing Optimization	Ant Colony Optimization
99	603	1647	635	651	14669
200	646	1613	850	628	57569
300	665	1639	1066	686	146027
400	654	1633	1557	722	255079
500	677	1684	4133	714	546849

The table below represents the distances obtained by each of the algorithm for various data sets:

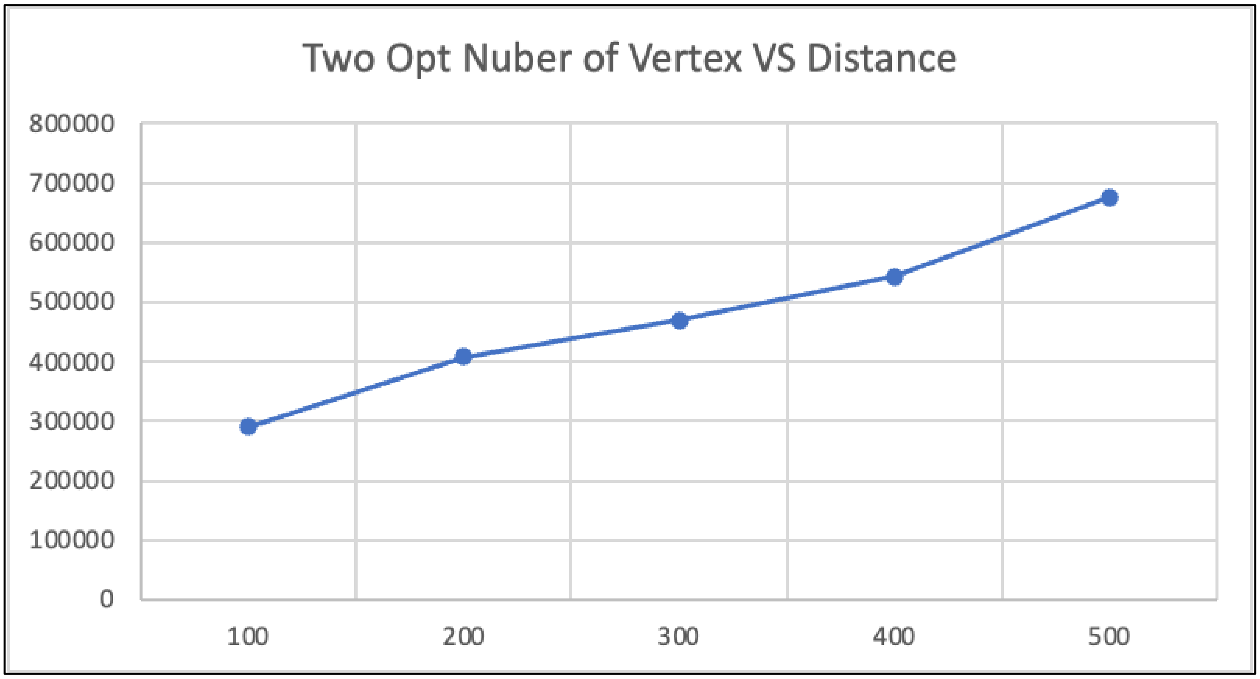
Number of Vertex	Christofides Algorithm	Two Opt Optimization	Three Opt Optimization	Simulated Annealing Optimization	Ant Colony Optimization
99	342602.1646	289505.0755	280897.739	334218.0304	342593.2288
200	459764.5579	407648.7545	394415.5355	455876.0071	459665.6261
300	533123.2458	469524.7351	465380.439	531752.2877	532313.8585
400	647869.3783	543129.2226	545396.8443	645442.084	647602.9051
500	749750.6481	676351.4665	634514.0213	749177.1667	749575.8292

Number of Vertices versus Distances Graphs

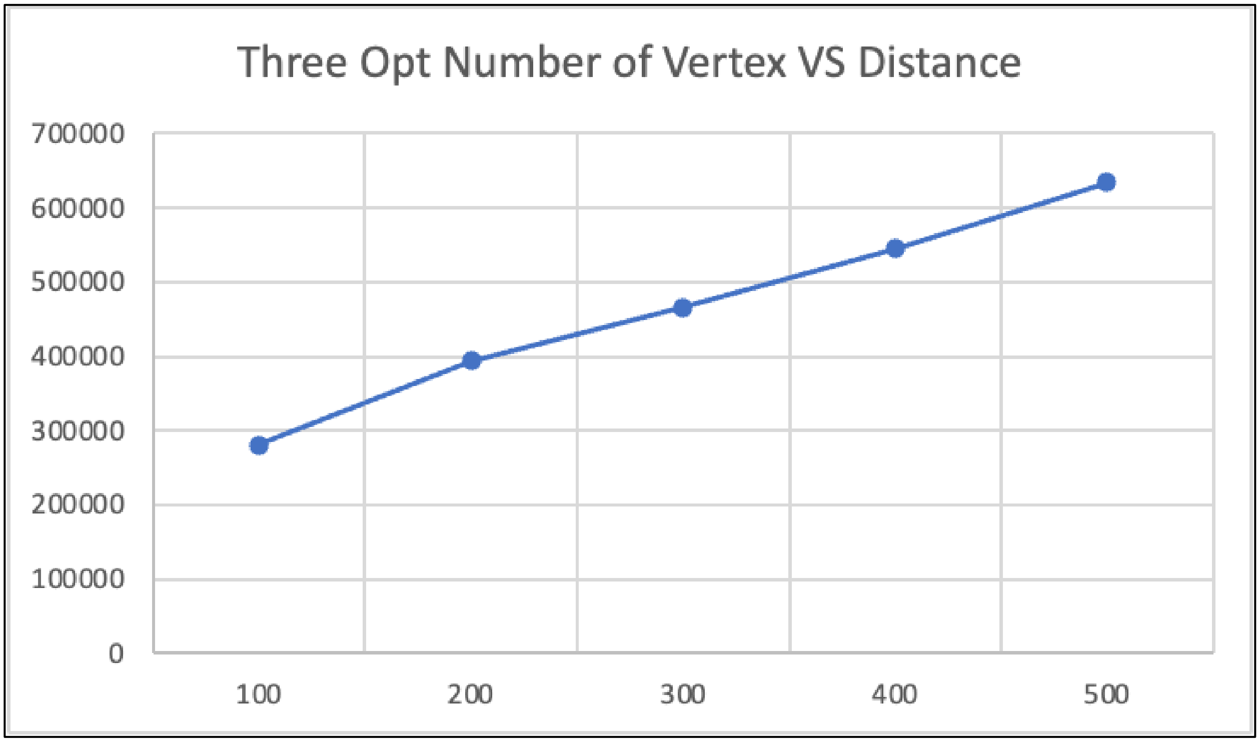
1. **Christofides Algorithm**



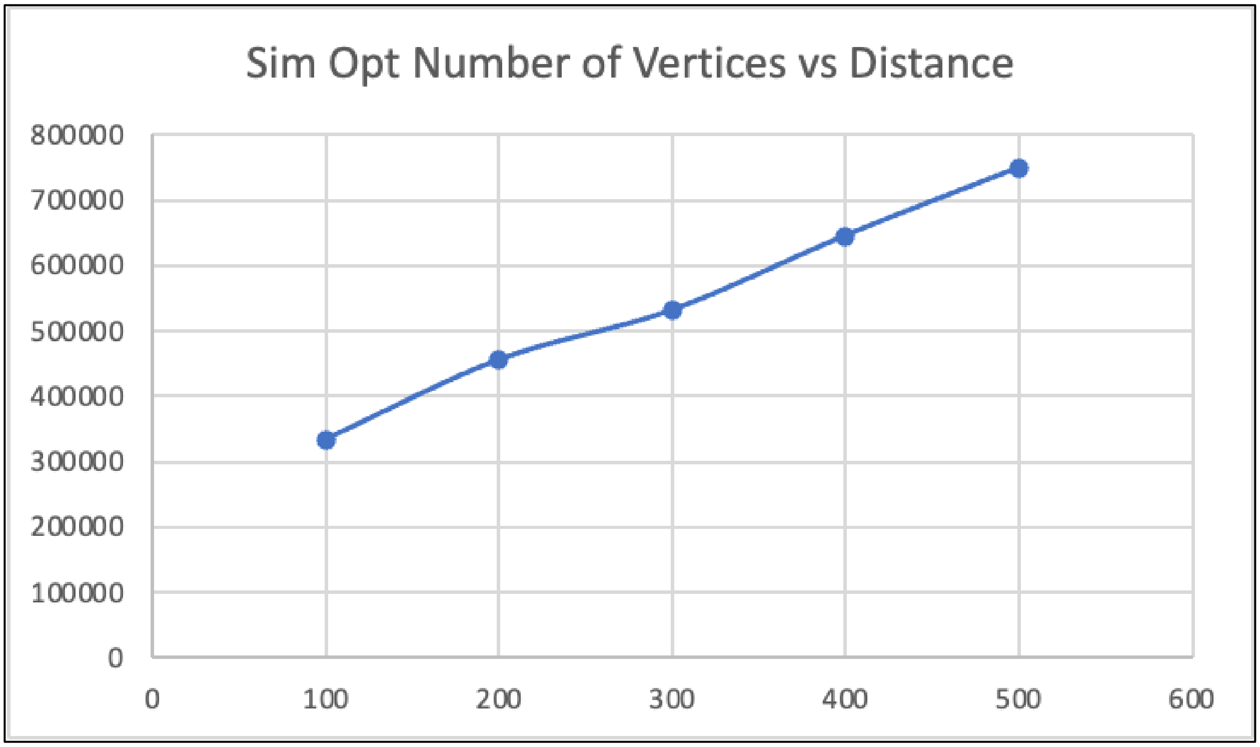
2. **Two Opt Optimization**



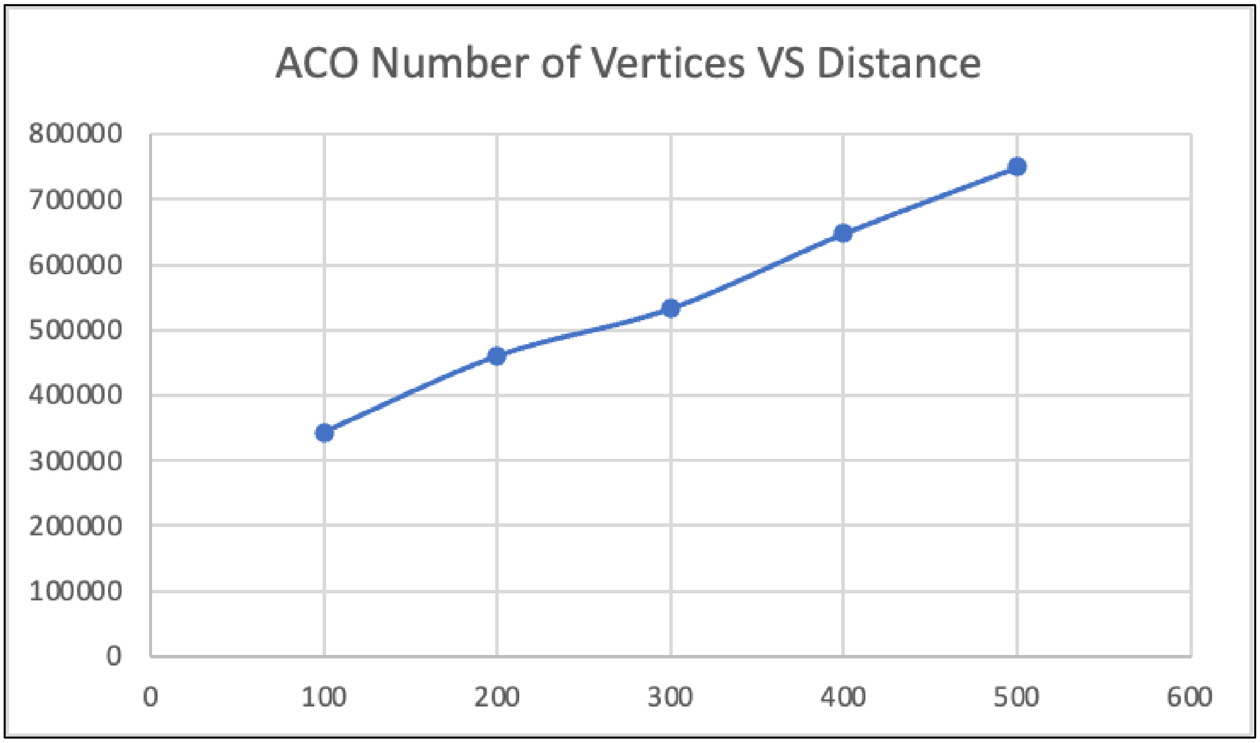
3. Three Opt Optimization



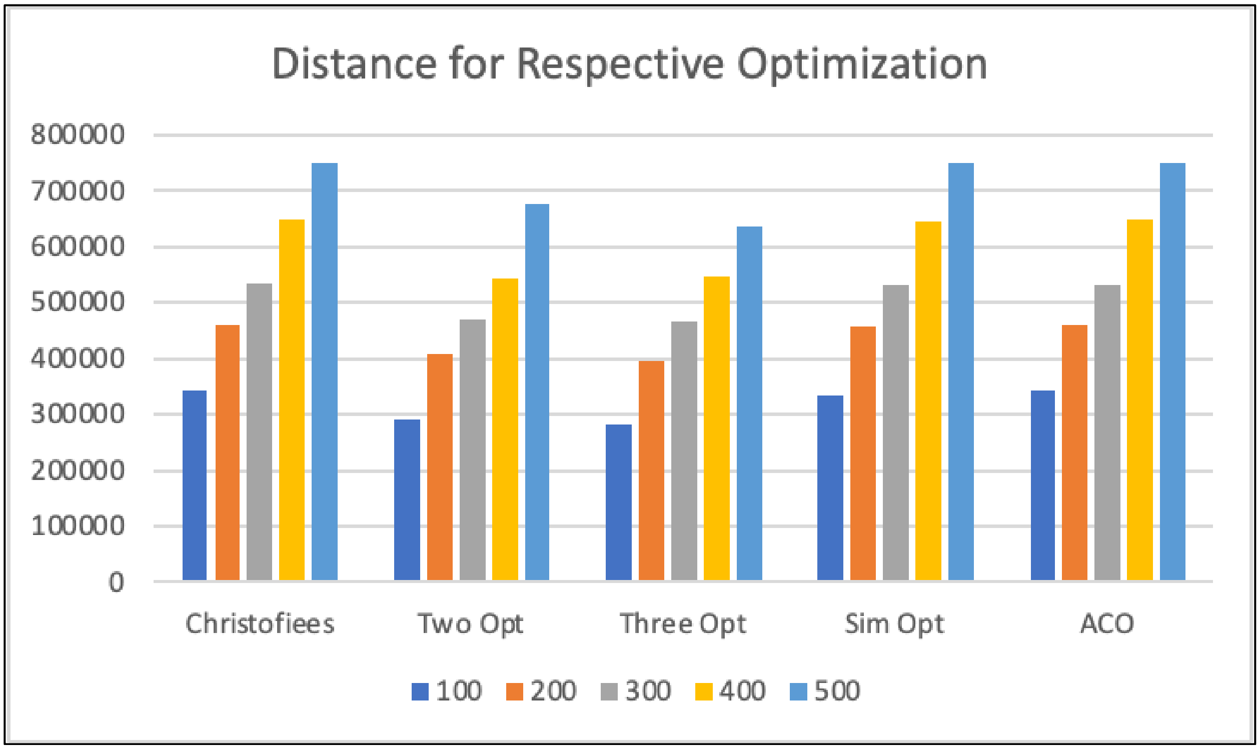
4. Simulated Annealing



5. Ant Colony Optimization



The following graph depicts the comparison between the distances generated by the respective optimization algorithms:



Results

Below are the visualizations plotted using Java AWT. The visualizations are plotted with and without Vertex Id for all the algorithms.

1. Christofides Algorithm

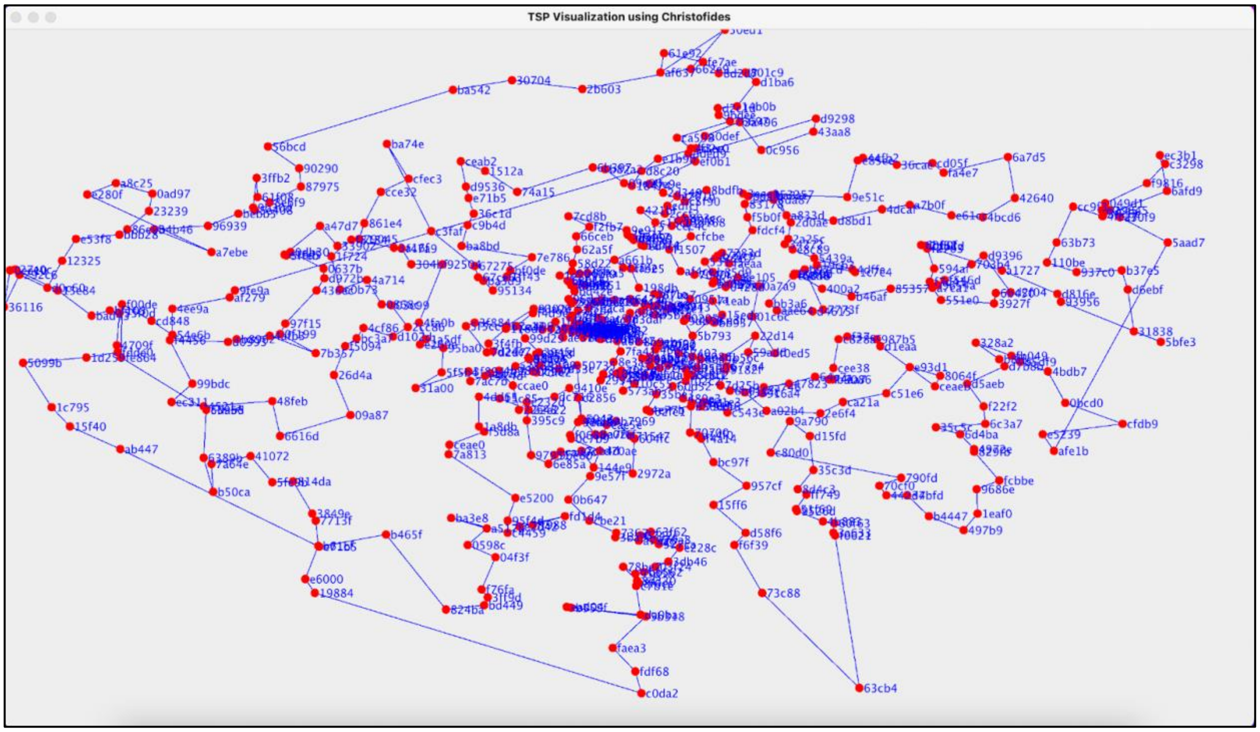


Fig: Christofides Algorithm with Vertex Id

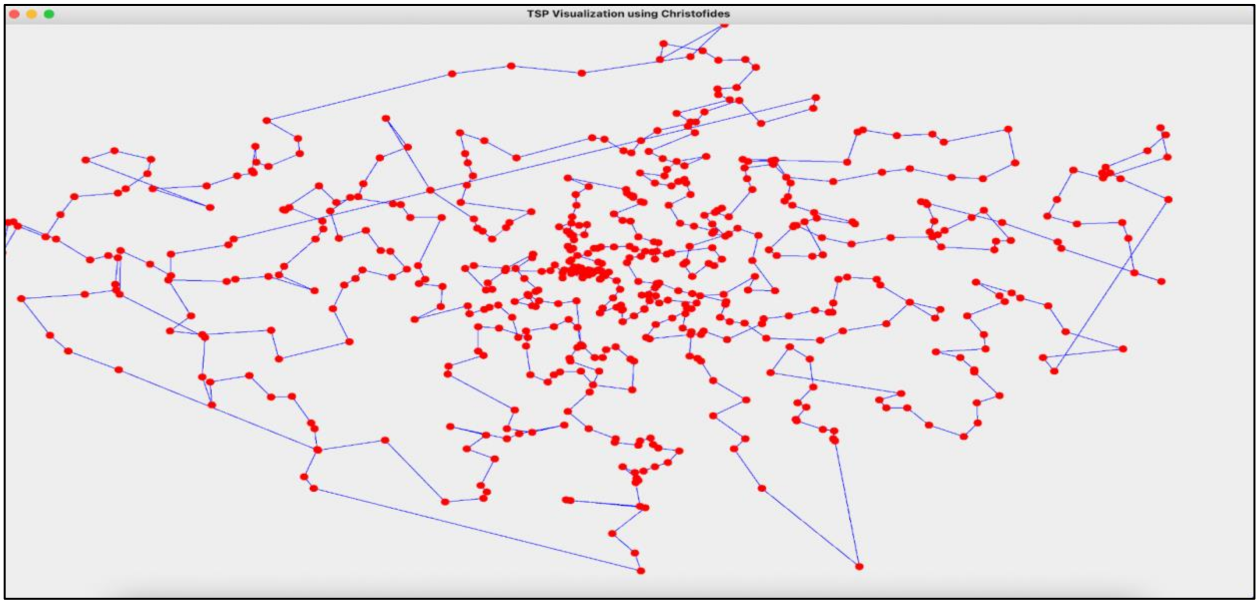


Fig: Christofides Algorithm without Vertex Id

2. Two Opt Algorithm

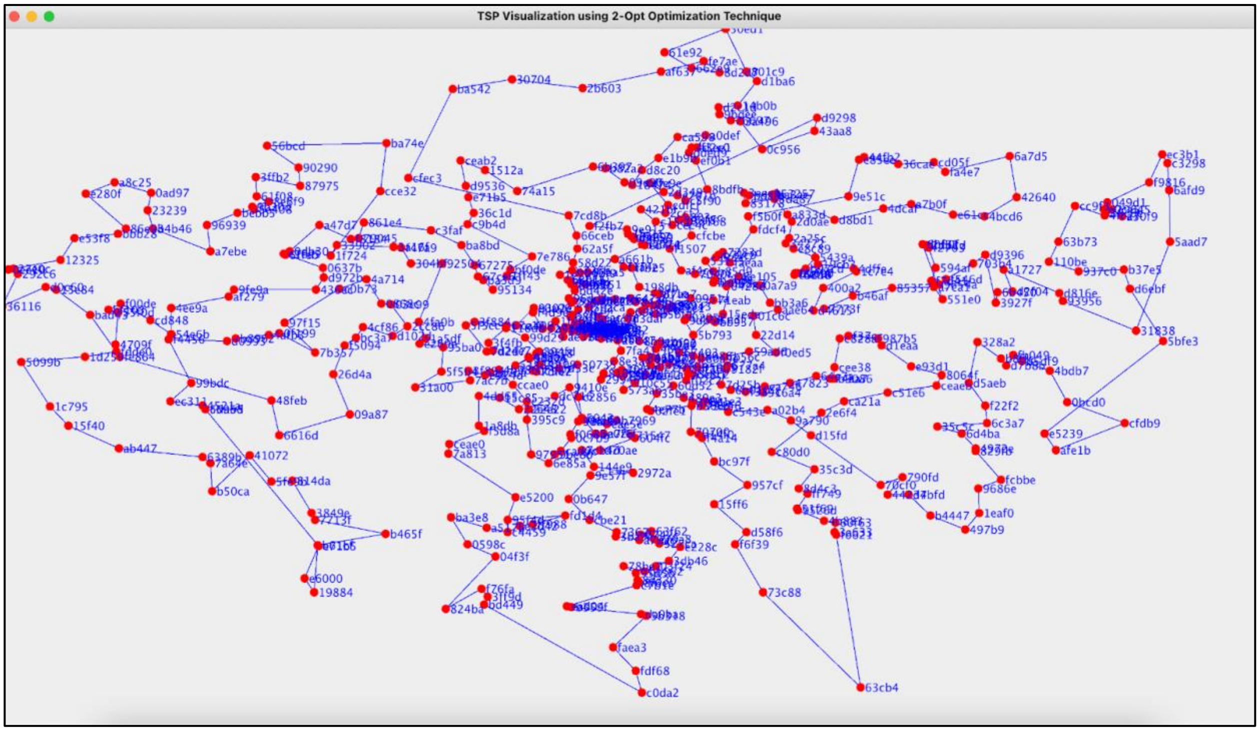


Fig: 2-Opt Algorithm with Vertex Id

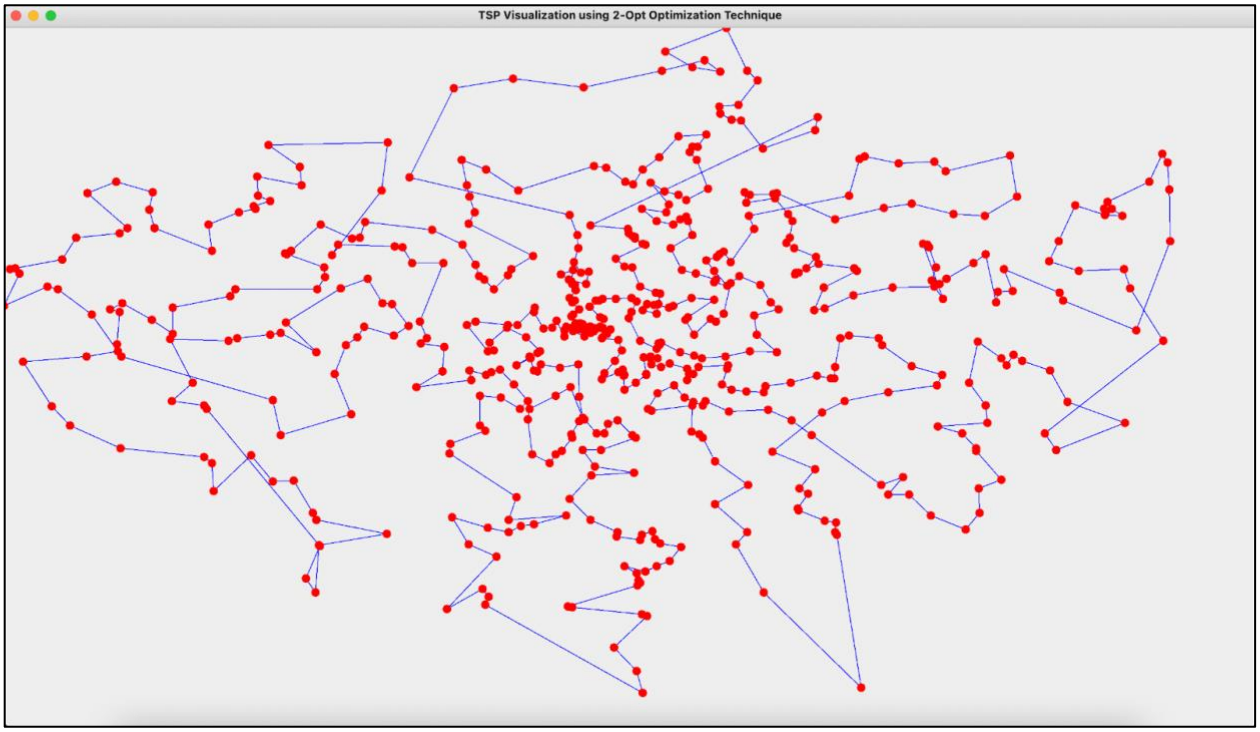


Fig: 2-Opt Algorithm without Vertex Id

3. Three Opt Algorithm

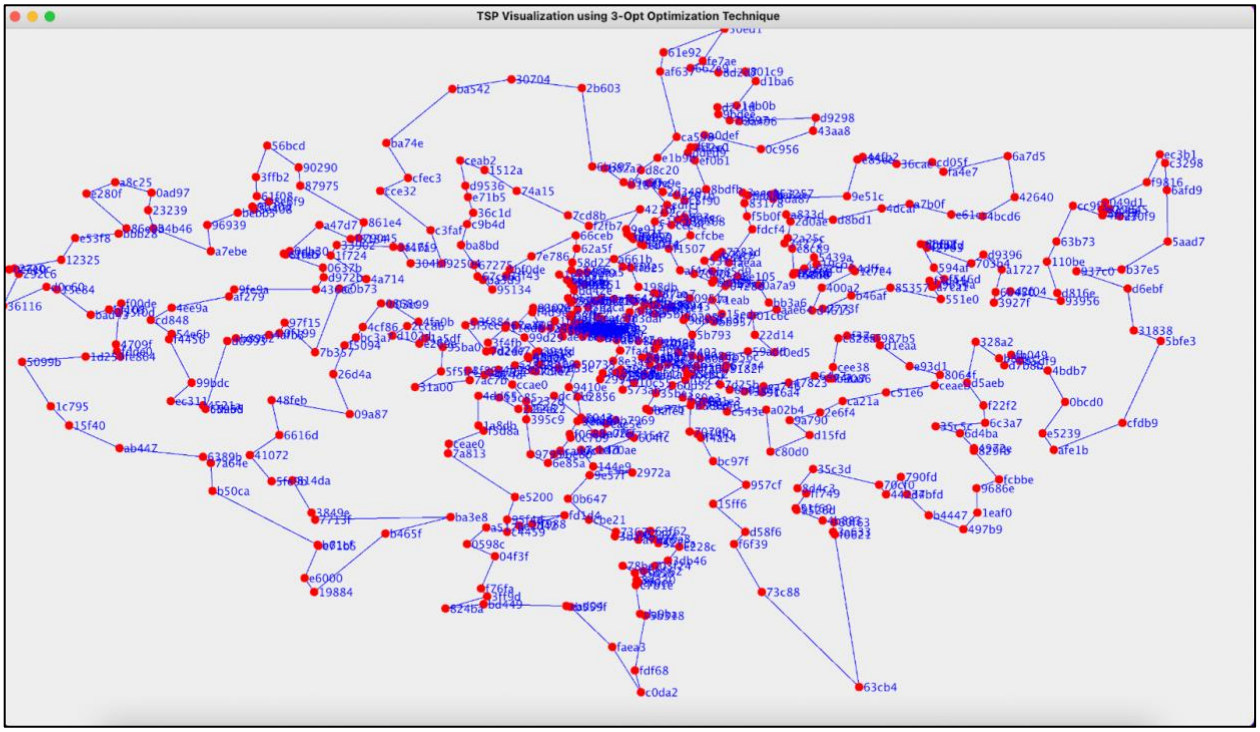


Fig: 3-Opt Algorithm with Vertex Id

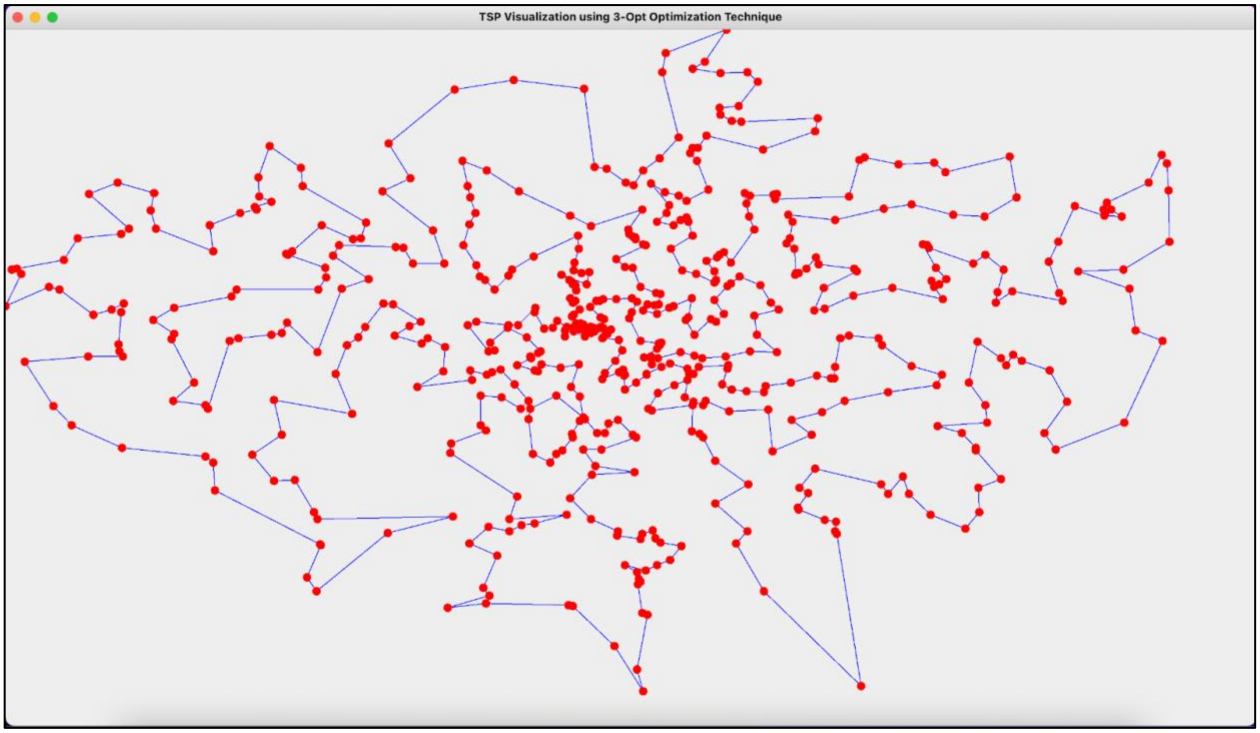


Fig: 3-Opt Algorithm without Vertex Id

4. Simulated Annealing Algorithm

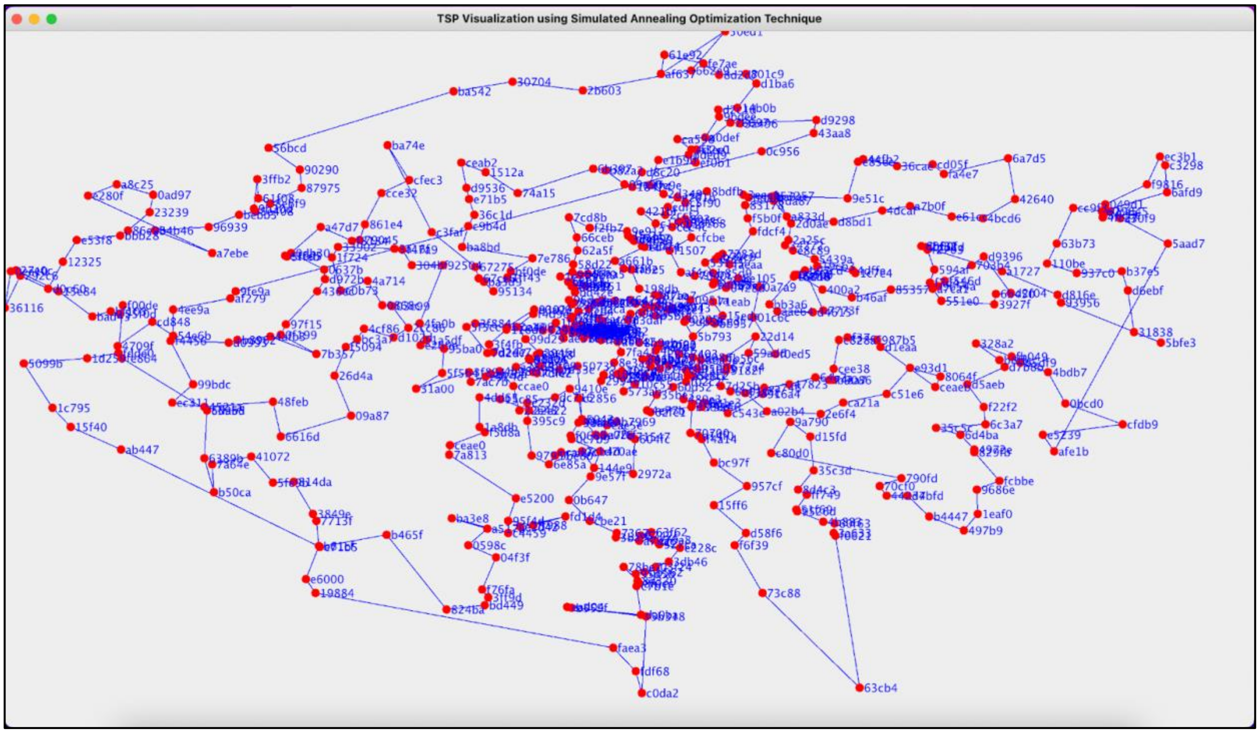

TSP Visualization using Simulated Annealing Optimization Technique

Fig: Simulated Annealing Algorithm with Vertex Id

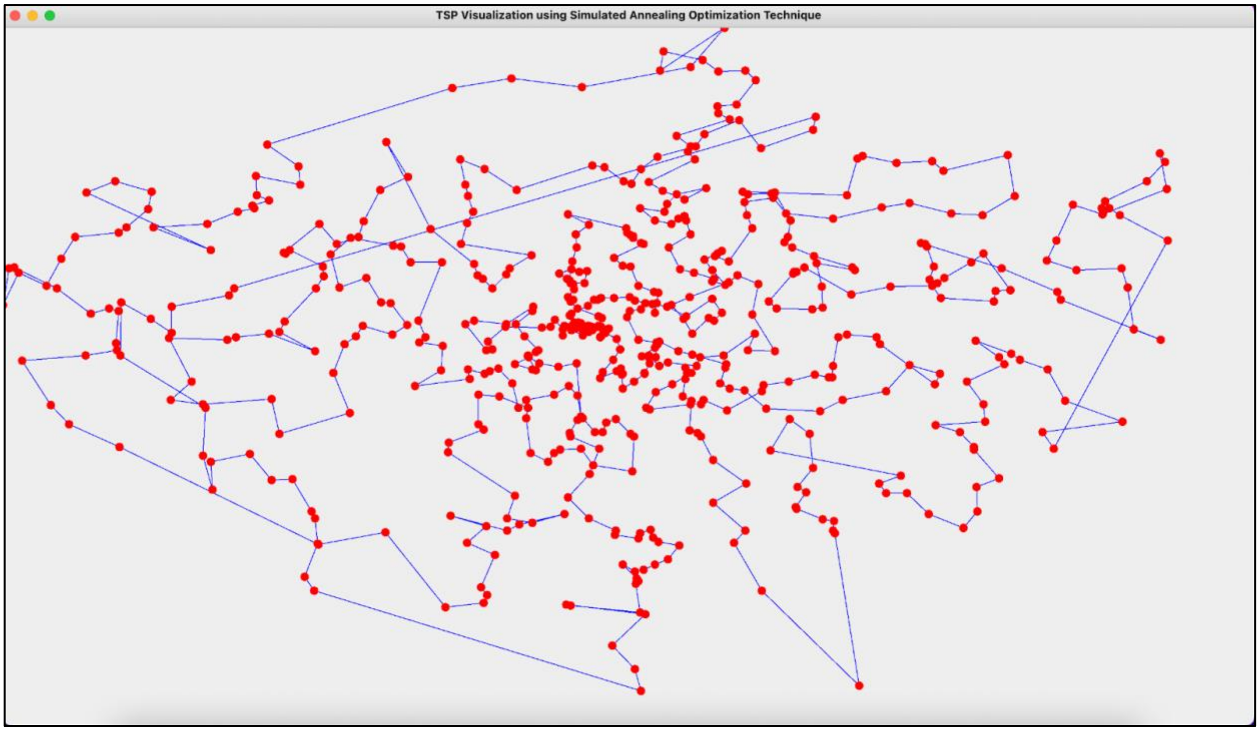
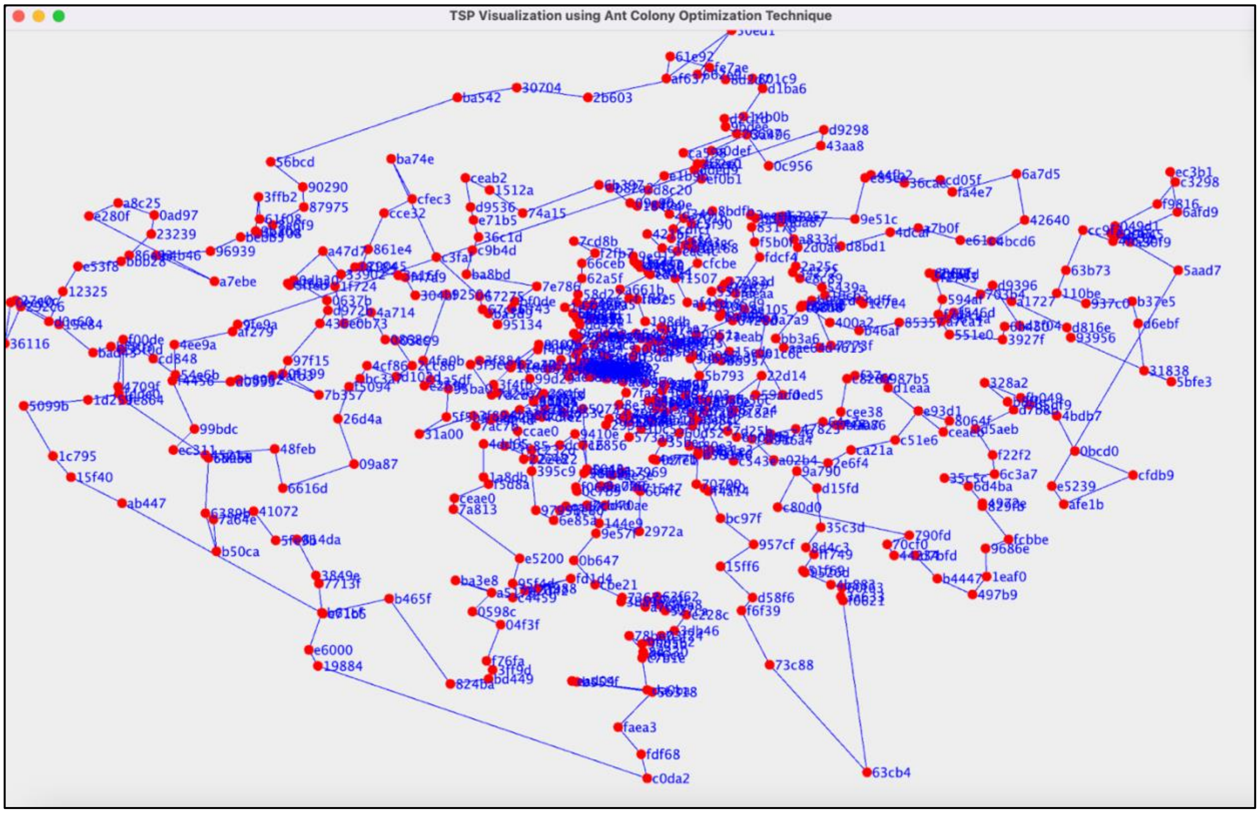

TSP Visualization using Simulated Annealing Optimization Technique

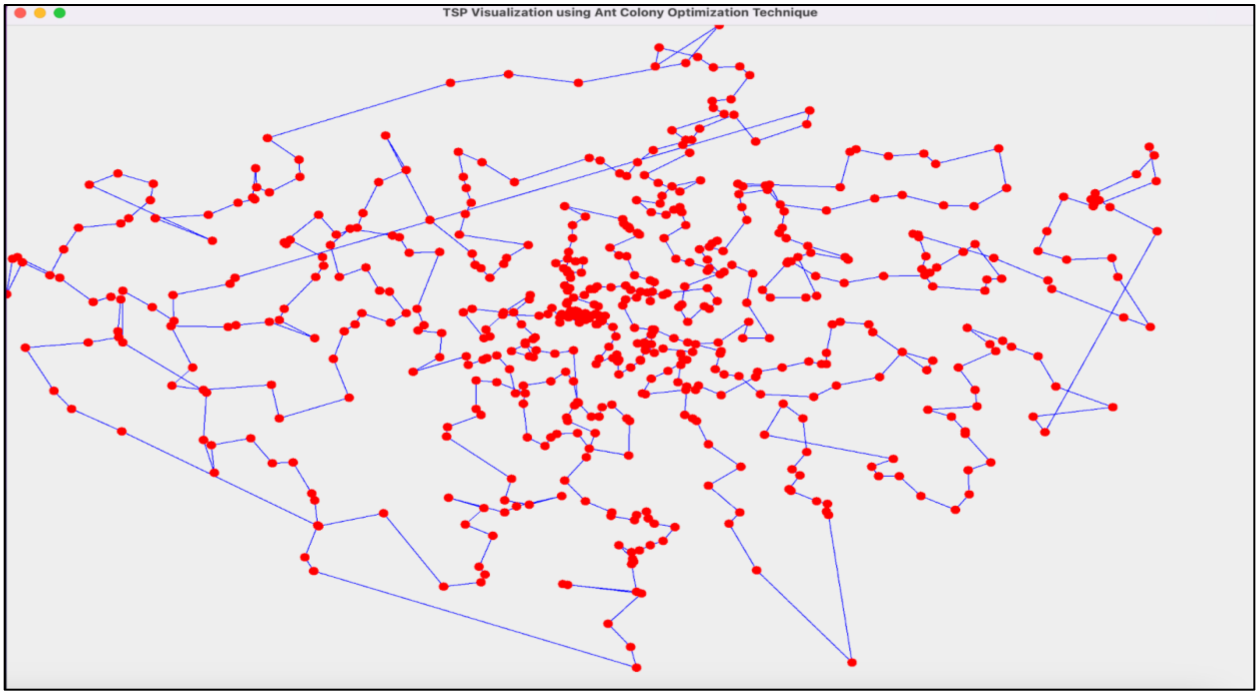
Fig: Simulated Annealing Algorithm without Vertex Id

5. Simulated Annealing Algorithm



TSP Visualization using Ant Colony Optimization Technique

Fig: Any Colony Algorithm with Vertex Id



TSP Visualization using Ant Colony Optimization Technique

Fig: Any Colony Algorithm without Vertex Id

Mathematical Analysis

1. Mathematical Analysis for Christofides Algorithm

a. Prim's Algorithm

It is used to determine the given graph's smallest spanning tree. When E is the number of edges and V is the number of vertices, the time complexity is O(ElogV).

b. Duplicate Removal

Since the minimum spanning tree may have redundant edges, they are taken out to produce a straightforward path. This method takes O(E) time complexity.

c. Hierholzer's Algorithm

On the path that results from step 2, it is utilized to locate an Eulerian circuit. It takes O(E) amount of time to complete, where E is the number of edges.

d. Shortcut Edges

By excluding the repeated vertices from the Eulerian circuit formed in step 3, a Hamiltonian circuit is produced. Ideally, it can be completed in O(V) time.

Overall, Christofides' approach is O (ElogV + E + V) in terms of time complexity when employing Prim's algorithm and Hierholzer's algorithm.

2. Mathematical Analysis for Two Opt Algorithm

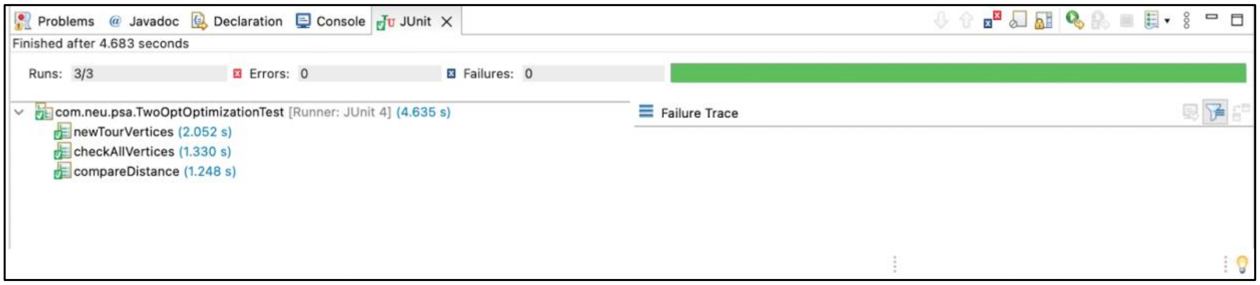
The time complexity of the Two Opt algorithm is O(n^3), where n is the number of vertices. This is due to the fact that, for each iteration of the method, all pairs of edges (i,k) such that i k must be evaluated, which requires O(n^2) pairings, and that, for each pair of edges, the order of the edges between vertices i and k must be reversed, which requires an O(n) operation. As a result, the algorithm's overall time complexity is O(n*n^2) = O(n^3).

3. Mathematical Analysis for Three Opt Algorithm

The algorithm's time complexity is O(n^3). This is because we execute the three-opt swap operation, which requires O(n) time and results in a total O(n^3) time complexity, for every pair of edges (i, j). The algorithm's spatial complexity is O(n). This is so that the algorithm can retain a single list of N vertices that represents the current tour.

Unit Tests

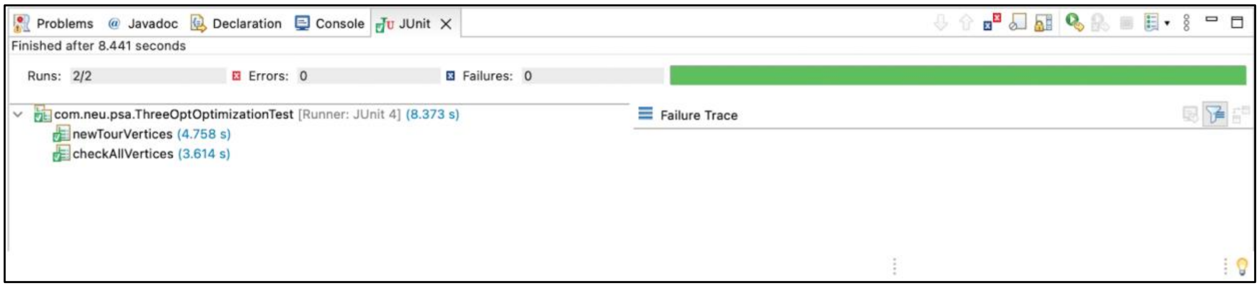
1. Two Opt Unit Test Result



```
Finished after 4.683 seconds
Runs: 3/3    Errors: 0    Failures: 0
com.neu.psa.TwoOptOptimizationTest [Runner: JUnit 4] (4.635 s)
    newTourVertices (2.052 s)
    checkAllVertices (1.330 s)
    compareDistance (1.248 s)
```

Fig: 2-Opt Unit Test Results

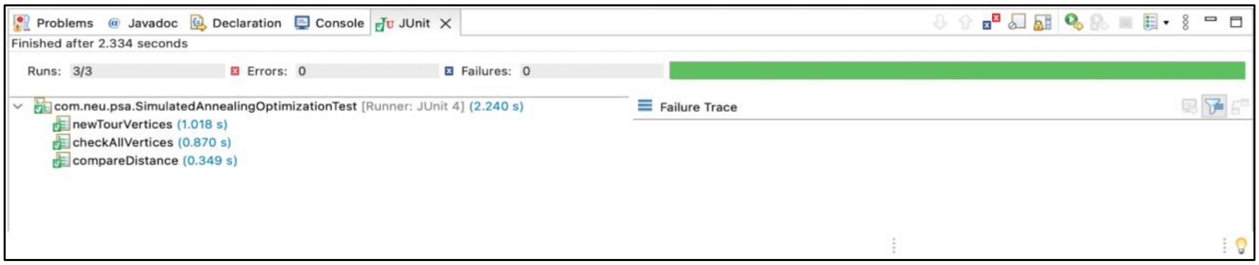
2. Three Opt Unit Test Result



```
Finished after 8.441 seconds
Runs: 2/2    Errors: 0    Failures: 0
com.neu.psa.ThreeOptOptimizationTest [Runner: JUnit 4] (8.373 s)
    newTourVertices (4.758 s)
    checkAllVertices (3.614 s)
```

Fig: 3-Opt Unit Test Results

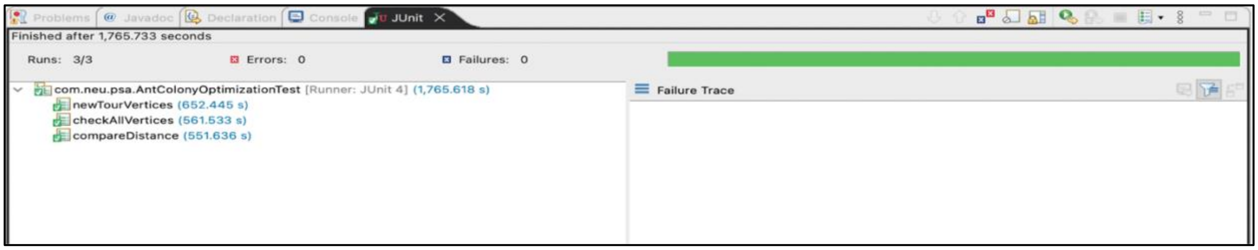
3. Simulated Annealing Unit Test Result



```
Finished after 2.334 seconds
Runs: 3/3    Errors: 0    Failures: 0
com.neu.psa.SimulatedAnnealingOptimizationTest [Runner: JUnit 4] (2.240 s)
    newTourVertices (1.018 s)
    checkAllVertices (0.870 s)
    compareDistance (0.349 s)
```

Fig: Simulated Annealing Unit Test Results

4. Ant Colonization Unit Test Results



```
Finished after 1,765.733 seconds
Runs: 3/3    Errors: 0    Failures: 0
com.neu.psa.AntColonyOptimizationTest [Runner: JUnit 4] (1,765.618 s)
    newTourVertices (652.445 s)
    checkAllVertices (561.533 s)
    compareDistance (551.636 s)
```

Fig: Simulated Annealing Unit Test Results

Conclusion

Based on the results obtained, it can be concluded that the Three Opt Optimization technique is the most effective method for optimizing the distance for the given data set. However, when considering the time taken for the optimization algorithm to run, the Simulated Optimization technique proves to be the fastest.

Furthermore, in terms of the algorithm's stability, it is observed that Strategic optimization technique algorithms are more stable compared to Tactical optimization techniques. However, for smaller data sets, Tactical Optimization techniques work better in terms of computational efficiency.

Overall, it is important to choose the optimization technique based on the specific problem and data set being analyzed, considering factors such as the size of the data set, the desired level of optimization, and the available computational resources.

References

1. https://www.math.cmu.edu/~af1p/Teaching/OR2/Projects/P58/OR2_Paper.pdf

2. https://www.baeldung.com/java-simulated-annealing-for-traveling-salesman

3. https://www.technical-recipes.com/2017/applying-the-2-opt-algorithm-to-travelling-salesman-problems-in-c-wpf/

4. https://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5

5. https://stackabuse.com/simulated-annealing-optimization-algorithm-in-java/