INFO 6205 Spring 2023 Project

Traveling Salesman

Padma Anaokar :002727445

Yash Pawar : 002747371

Rajat Sharma : 002736944

Introduction

Aim

The main aim of the Traveling Salesman Problem (TSP) is to find the optimal route for a salesperson to visit a given set of cities and return to the starting city, minimizing the distance travelled. In general, the TSP is concerned with finding the optimal Hamiltonian cycle (i.e., a cycle that passes through every vertex exactly once) in a weighted, undirected graph. Many algorithms have been developed with an aim to find a solution that is as close to optimal as possible, while keeping the running time reasonable.

Approach:

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem in which a salesman must visit a set of cities and return to the starting city, covering the shortest possible distance. Given that your dataset has 585 points, the TSP can be computationally expensive to solve optimally. Therefore, you can follow the following approach to find the best TSP tour:

1. Pre-processing the data: To reduce the computational cost, preprocess the dataset by removing any points that are close together, and reducing the number of points to a manageable size. You can also cluster the points using K-means algorithm to create sub-tours. We will not be doing this as we have a dataset which is preprocessed.
2. Minimum Spanning Tree: Build a Minimum Spanning Tree (MST) using Prim's or Kruskal's algorithm. This algorithm will connect all the nodes and give you an approximate lower bound for the TSP tour.
3. Christofides algorithm: Use the Christofides algorithm to find an approximate TSP tour. This algorithm will give you a solution that is guaranteed to be within 1.5 times the optimal solution.

Ways of Optimization: The Traveling Salesman Problem (TSP) can be solved using a few random optimization methods. These approaches examine the solution space randomly to identify good solutions. Some of the most well-liked random optimization methods we used for TSP are listed below:

- Simulated Annealing
- Ant Colony Optimization
- Genetic Algorithm
- Random Swapping Optimization
  - i: 2 opt Optimization
  - ii: 3 opt Optimization
  - iii: k opt Optimization

Program

Data Structures

- List,HashMap,Map,Set,HashSet,Array
- Customized class: Graph,Node

Classes

App.java

-Public static void main () - Main function which implements all algorithms and prints all results.

- Public static void printGraph(List<Node> nodes) - prints all nodes of provided tour sequentially.

-Public static Graph getNodesFromDataset() - Reads data from CSV and stores it in a Customized class Graph.

-Public static double distance(double lat1, double lat2, double lon1, double lon2) - This function calculates distance between 2 latitudes and longitudes.

-Public static void Visualizaiton (List<Node> nodes, List<GraphOperation> gos, List<GraphOperation> oldGos, int interval, String GraphTitle) - This is a function that calls visualization class and prints a graph of provided nodes.

-Protected static void sleep() - this function provides interval for the graph.

Christofides.java

-Public static List<GraphOperation> calcGraphOperation(List<Node> nodes) - calculates graphOperation schema for a provided tour.

-Public static List<Edge> findMST(Graph graph) - function that finds mst from a graph

-Public static List<Node> findOddDegreeVertices(Graph graph, List<Edge> mst) - Find all vertices with odd degree in MST - Step 2 of christofides

-Public static List<Edge> getMinimumWeightPerfectMatching(List<Node> oddDegreeNodes) - find Minimum Weight Perfect Matching, Step 3 of Christofides

-Private static void dfs(Node node, List<Node> eulerTour, List<Edge> edges, Set<Node> visited,Map<Node, List<Node>> adjacenyMatrix) - Depth first Search function used to find EulerTour.

-Public static List<Node> eulerTour(Graph graph, List<Edge> mst, List<Edge> perfEdges) - find Eulerian circuit of the Multigraph, Step 4 of Christofides

-Public static List<Node> generateTSPTour(List<Node> eulerTour) - Traverse the Eulerian circuit, Shortcutting across previously visited vertices to obtain a Hamiltonian Circuit. Final step of Christofides.

-Public static List<Node> randomSwapOptimise(List<Node> tspTour, Integer iterations) - Randomly swaps 2 edges and checks if we got a better tour.

-Public static List<Node> twoOpt(List<Node> nodes) - swaps 2 edges and checks if we got a better tour.

-Private static List<Node> twoOptSwap(List<Node> nodes, int i, int j) - function used by twoOpt to swap edges.

-Public static List<Node> threeopt(List<Node> tour) - Swaps a set of 3 edges 2 times to check if we got a better route.

-Public static List<Node> ThreeOptChristofides(List<Node> nodes) - Second 3Opt function for Christofides algo with optimization than the first one.

-Public static void kOpt(List<Node> tour, int k) - K opt swap Optimization technique to do multiple opt swapping to get a better tour.

-Public static List<Node> simulatedAnnealingOptimizeTour(List<Node> tour) - Optimization technique where it selects an initial solution and iteratively searches for better solutions by randomly making small changes to current solution.

-Public static List<Node> aCOpt(List<Node> nodes) - Ant Colony Optimization uses artificial ants to search for the shortest path between two points in a graph by following a probabilistic decision rule and leaving pheromone trails to communicate with other ants.

-Public static double calculateTourLength(List<Node> tour) - used to calculate total tour length for provided tour.

TSPGenetic.java

-Public static List<Node> TSPGenAlgo(List<Node> node) - main function where other functions are called to return a list<Node>

-Public Population evolvePopulation(Population population) - function applies genetic operators to a population to produce a new generation of potentially fitter individuals.

-Private Route crossover(Route parent1, Route parent2) - combines two parent routes to produce an offspring route that inherits characteristics from both parents through crossover.

-Private void mutate(Route route) - applies a mutation operator to randomly modify the order of cities in a route to enhance diversity and possibly improve the solution.

-Private Route tournamentSelection(Population population) - selects the fittest individual among two randomly chosen individuals from the population

Node.java

-Public Node(String crimeId, double longitude, double latitude) - constructor used to initialize variables.

Getters and Setters - used to get and set values of class attributes.

Edge.java

-Public Edge(Node source, Node destination, double distance) - constructor used to initialize variables.

Getters and Setters - used to get and set values of class attributes.

Graph.java

-Public Graph(List<Node> nodes, List<Edge> edges) - used to initialize class Attributes.

-Public void addNode(Node node) - used to add node to existing list of nodes.

-Public void addEdge(Node source, Node destination) - used to add edge to existing list of edges.

-Public static double calculateDistance(Node source, Node destination) - used to calculate distance between 2 nodes.

-Public List<Edge> kruskalMST() - Function to find out Minimum Spanning Tree

-Public List<Edge> getMinimumWeightPerfectMatching() - finds the minimum weight perfect matching among the set of vertices with an odd degree in a given graph.

-Public Map<Node, List<Node>> adjacencyMatrix() - provides adjacency matrix in the form of Map.

-Private Node find(Node node, Map<Node, Node> parents) - finds a node from the adjacency matrix.

Getters and Setters - used to get and set class attributes.

DeepCopyUtils.java

-Public static <T extends Serializable> List<T> deepCopy(List<T> original) - creates a deep copy of given list of objects by serializing and deserializing the list, thereby creating a completely new copy of the list in memory.

GraphOperation.java

-Private GraphOperation(Node node1, Node node2, Action act) - constructor to initialize class attributes.

-Public static GraphOperation addEdge(Node node1, Node node2) - function that notes an edge addition.

-Public static GraphOperation removeEdge(Node node1, Node node2) - function that notes an edge removal.

-Public static GraphOperation highlightEdge(Node node1, Node node2) - function that highlights an edge between provided nodes as red.

-Public static GraphOperation UnhighlightEdge(Node node1, Node node2) - function that highlights an edge between provided nodes as black.

-Public static GraphOperation addEdge(Edge edge) - function that notes an edge addition.

-Public static GraphOperation removeEdge(Edge edge) - function that notes an edge Removal.

-Public static GraphOperation highlightEdge(Edge e) - function that highlights provided edge as red.

-Public static GraphOperation UnhighlightEdge(Edge e) - function that highlights provided edge as black.

-Public boolean equals(Object obj) - checks if 2 GraphOperation objects are equal.

AlgorithmVisualization.java

-Public AlgorithmVisualization(List<Node> nodes, List<GraphOperation> gos, List<GraphOperation> oldGos) - constructor that initializes class attributes.

-Public void setSleepTime(int sleepTime) - used to set an time interval

-Public void showResult(String graphDes) - main function that prints a Graph.

-Private void changeEdge(Graph graph, GraphOperation go) - function to show highlight when edges are being compared with each other from 2 GraphOperations.

Algorithm

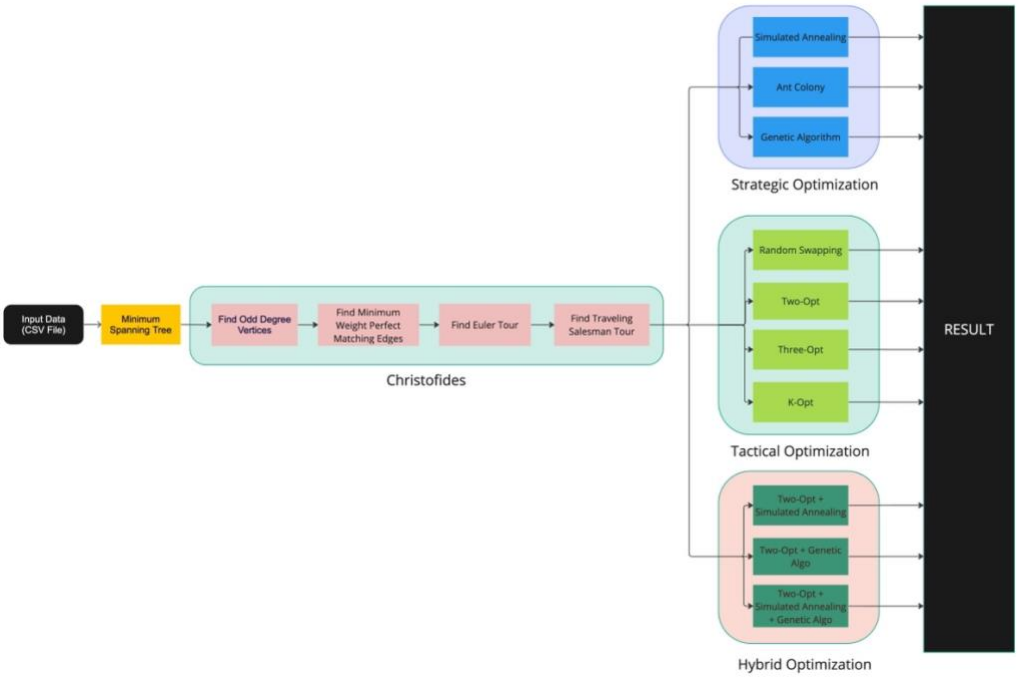
1. Find a minimum spanning tree (MST) of the input graph.
2. Construct a minimum weight perfect matching (MWPM) of the odd degree vertices in the MST.
3. Combine the MST and MWPM to form a connected, Eulerian graph.
4. Find an Eulerian tour of the graph.
5. Shortcut the Eulerian tour to obtain a Hamiltonian tour, i.e., a cycle that visits each vertex exactly once.
6. Use RandomSwapping to improve Hamiltonian tour obtained from the previous steps by randomly swapping pairs of adjacent vertices and accepting the swap if it results in a shorter tour length
7. Use 2OPT optimization technique to improve the Hamiltonian tour obtained from the previous steps by iteratively removing two edges from the tour and reconnecting the resulting fragments in a way that results in a shorter tour length.
8. Use K-Opt to improve the Hamiltonian tour obtained from the previous steps by iteratively removing k edges from the tour and reconnecting the resulting fragments in a way that results in a shorter tour length. The value of k can be any positive integer, and larger values of k can lead to better tour improvements but require more computational resources and time. This is optional but can give significant reduction in Hamiltonian tour length.
9. Use SimulatedAnnealing, a metaheuristic optimization algorithm used in Christofides algorithm, to explore the solution space by gradually reducing the temperature parameter, allowing for some uphill moves in order to escape local optima and converge to a better solution. Use this technique on 2Opt tour for better solution.

10. Ant Colony Optimization is a metaheuristic algorithm used as an alternative to the local search techniques in Christofides algorithm, where a population of artificial ants are iteratively simulated to explore the solution space and update a pheromone matrix that guides the search towards better solutions. This algorithm takes larger computational resources but does not give a significant advantage over Simulated Annealing with 2OPT tour.
11. Genetic Algorithm Optimization is a metaheuristic optimization technique used in Christofides algorithm to generate and evolve candidate solutions by combining different parts of the previously obtained solutions, with selection, crossover and mutation operations, to find a potentially better Hamiltonian tour solution. This algorithm does not gaurantee a better solution but with appropriate population size and mutation rate, we can reach to a better solution.

Invariants

1. The initial TSP tour is obtained by running the MST and Christofides algorithms.
2. RandomSwapping, 2OPT, Simulated Annealing, and Genetic Algorithm Optimization are applied iteratively to improve the TSP tour.
3. Each of the optimization techniques has its own parameters and termination criteria, which should be set appropriately for the problem instance and computational resources available.
4. The final TSP tour found by the optimization process should be a valid tour, i.e., it should visit each city exactly once and return to the starting city.

Flow Charts (inc. UI Flow):



Observations & Graphical Analysis

Christofides Algo

1. Christofides algorithm uses minimum spanning tree and minimum weight perfect matching to find a solution that is at most 1.5 times the length of an optimal tour.

2. Christofides algorithm is a heuristic and may not find the optimal solution, but it has been demonstrated to be effective in practice and is frequently employed as a starting point for more sophisticated algorithms.

3. Christofides algorithm may not work well if the distance metric used in the problem instance does not follow the triangle inequality, which can result in a much longer tour than the optimal one.

Random Swapping:

Advantages:

1. Random Swapping is a simple and easy-to-implement heuristic that requires minimal computation.
2. It can be used as a building block for more complex algorithms, such as Iterated Local Search.
3. The method can quickly escape from local optima, which makes it useful for exploring different regions of the solution space.

Drawbacks:

1. Random Swapping can be slow to converge, especially for large problem instances with many cities.
2. It can lead to long and complex tours, which can be difficult to optimize further.
3. The quality of the solution produced by Random Swapping depends on the initial tour, which can be highly variable.

2 Opt:

Advantages:

1. 2 Opt is a simple and effective heuristic that can quickly improve the quality of a given tour.
2. It can be applied to any tour, regardless of how it was constructed or its length.
3. The method is easy to implement and can be adapted to different optimization criteria.

Drawbacks:

1. 2 Opt can be slow to converge, especially for large problem instances with many cities.
2. It may not be able to escape from local optima, which can limit its ability to find the global optimum.
3. The quality of the solution produced by 2 Opt depends on the initial tour, which can be highly variable.

Simulated Annealing:

1. Simulated Annealing can easily escape from local optima, which makes it useful for finding global optima.
2. However, it can be computationally expensive, especially for large problem instances with many cities.
3. The quality of the solution produced by Simulated Annealing depends on the parameters used, such as the initial temperature and cooling rate.

Genetic Algorithm

1. Genetic Algorithm can quickly converge to a good solution, especially for large problem instances with many cities.
2. However, it can be sensitive to the parameters used, such as the population size, mutation rate, and crossover rate.
3. The quality of the solution produced by Genetic Algorithm depends on the fitness function used to evaluate the candidate solutions.

Graphs:

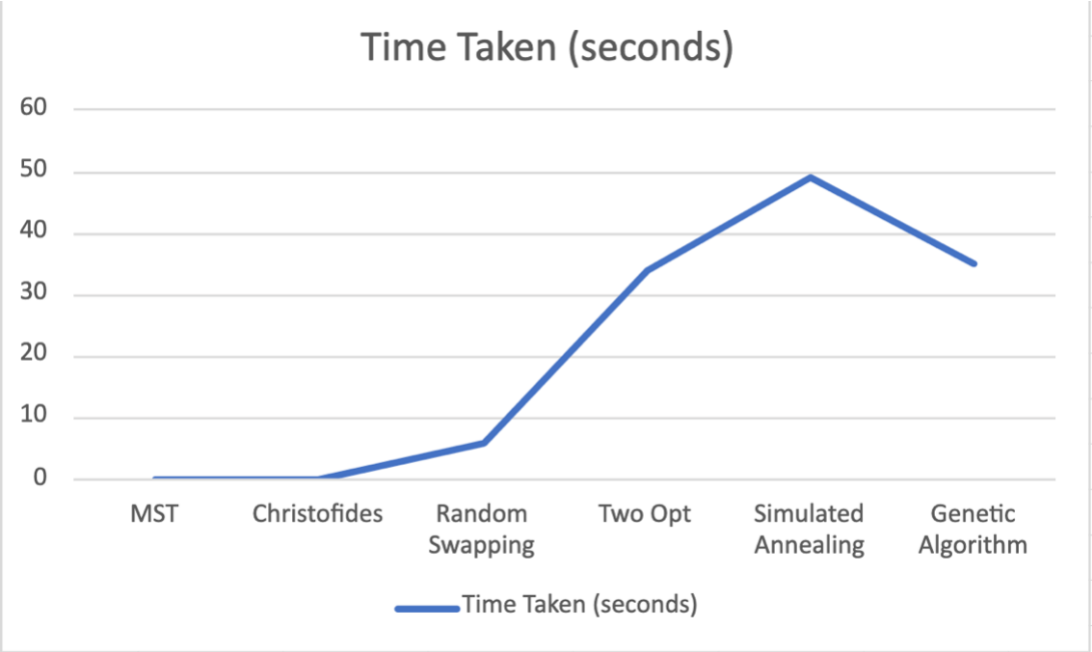


Figure: Time Taken

Figure: Tour Cost

Results & Mathematical Analysis

Below is our mathematical Analysis of our algorithms along with the tour cost

Algorithm	Type	Approximation Rate	WorstCase TC	Positives	Negatives	Tour Cost
MST	Exact	2	O(n^2logn	- Always produces a valid tour	- Can be slow for large inputs	513890.10 meters
Christofides	Approximate	1.5	O(n^3)	- Guaranteed to produce a valid tour-	- Can be slow for large inputs	714166.31 meters
RandomSwapping	Heuristic, Tactical	N/A	O(n^2)	- Simple to implement- Fast	- Can get stuck in local minima	707842.5 meters
2Opt	Heuristic, Tactical	N/A	O(n^2)	- Can produce feasible better soln	- Can get stuck in local minima	631983.57 meters
3Opt	Heuristic, Tactical	N/A	O(n^3)	- More powerful than 2OPT	- Can get stuck in local minima	636677.21 meters
Simulated Annealing	Heuristic, Strategic	N/A	N/A	- Can be effective for large inputs	- Can get stuck in local minima	639632.04 meters
Genetic Algorithm	Heuristic, Strategic	N/A	N/A	- Can be effective for large inputs	- Can be slow	1079298.63 meters

Figure: Mathematical Analysis

Mathematical Analysis

1] RandomSwapping- This technique does not produce a better solution in less number of swaps. When we Increase number of Swaps(Iterations), probability of getting a better result also increases. But, Time taken in more number of swaps is O(n^2logn) which is more than other algorithms.

2] TwoOpt- Algorithm gaurantees to provide a better solution but on cost of Time- It takes more time and usual to converge a better solution.

3] Simulated Annealing - a high initial temperature and a slow cooling schedule are useful for exploring the search space, while a low initial temperature and a fast cooling schedule are useful for refining the solution once a good solution has been found.

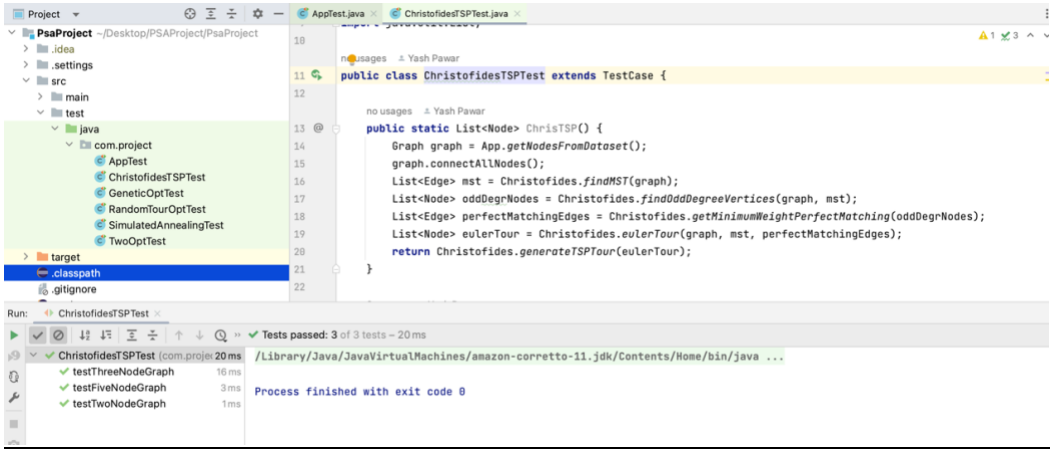
4]Genetic Algorithm

- A larger population size can lead to a more diverse population and potentially better solutions, but it can also increase the time and memory requirements of the algorithm

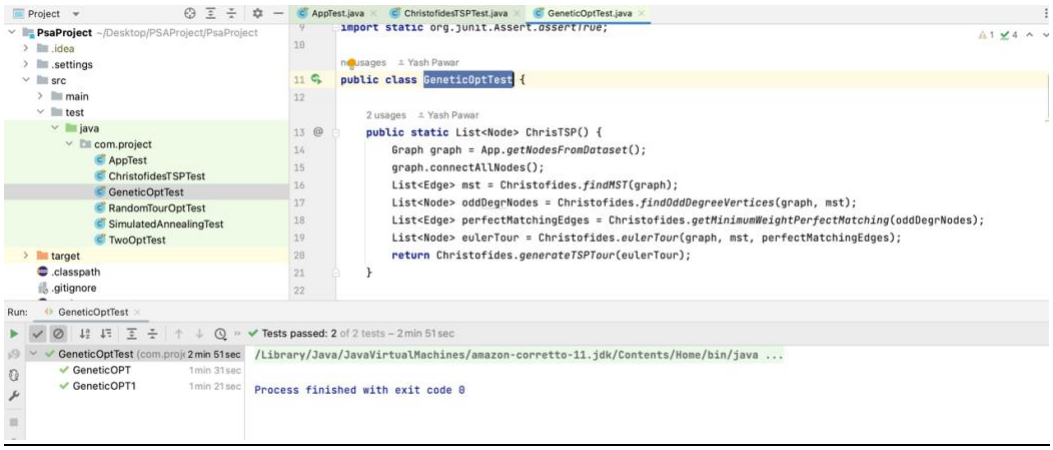
- A larger tournament size can increase the selection pressure and potentially lead to better solutions, but it can also increase the time complexity of the algorithm
- A higher mutation rate can introduce more diversity into the population, but it can also hinder convergence to good solutions. A value of 0.8 is quite high and may lead to too much randomness
-A larger number of generations can lead to better solutions, but it can also increase the time complexity of the algorithm. A value of 1000 is reasonable for many problems.

Unit tests:

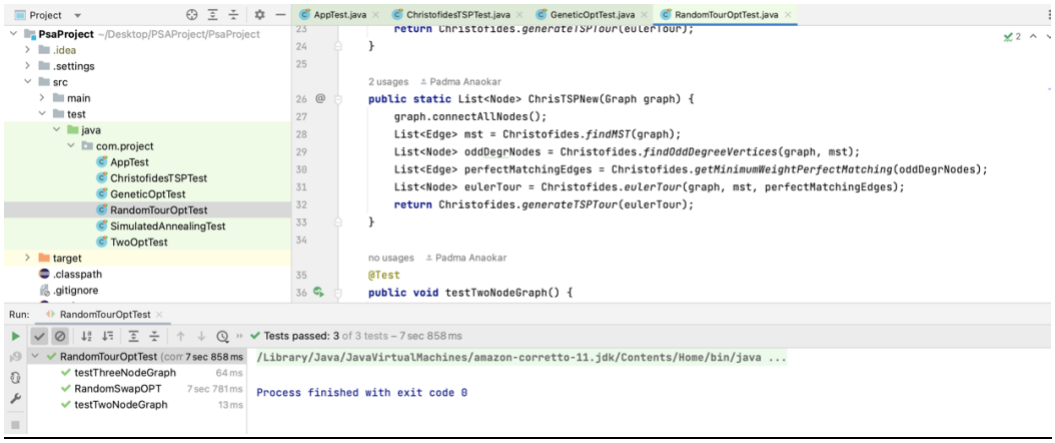
1.Christofides TSP Test



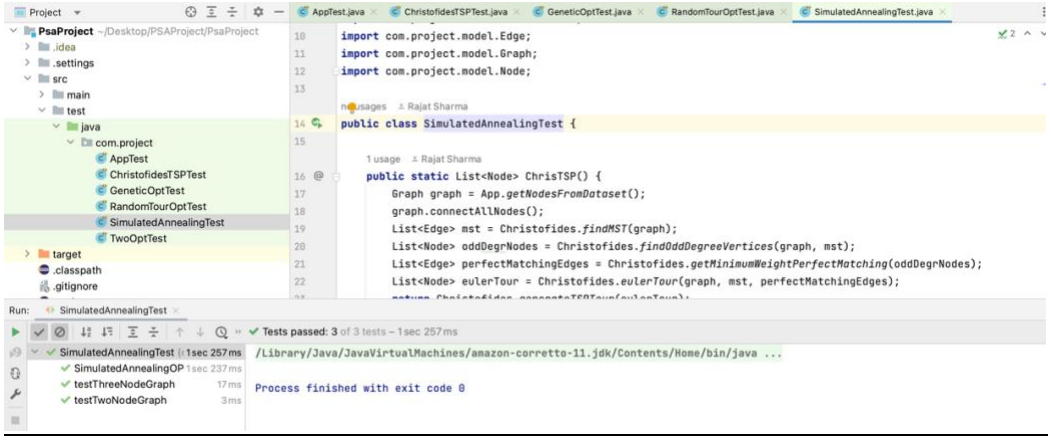
2.Genetic Opt Test



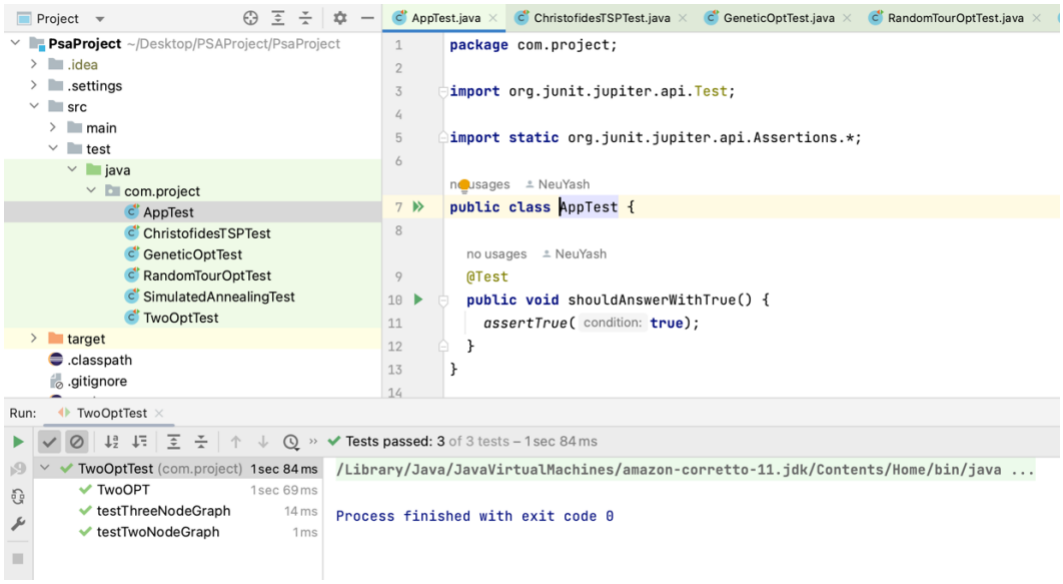
3.RandomTour Opt Test



4.SimulatedAnnealing Test

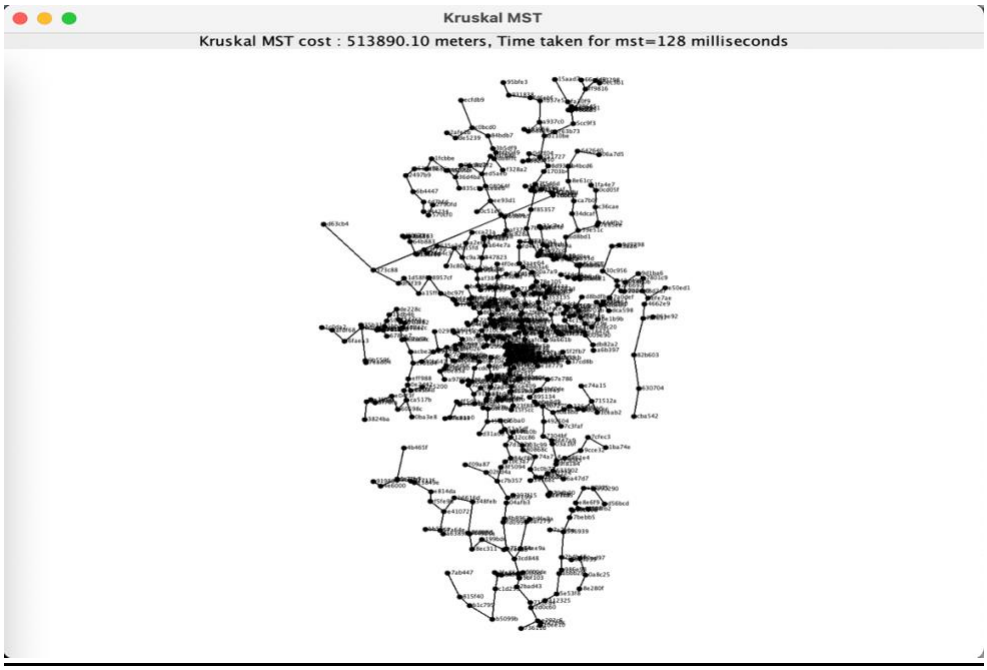


5.Two Opt Test

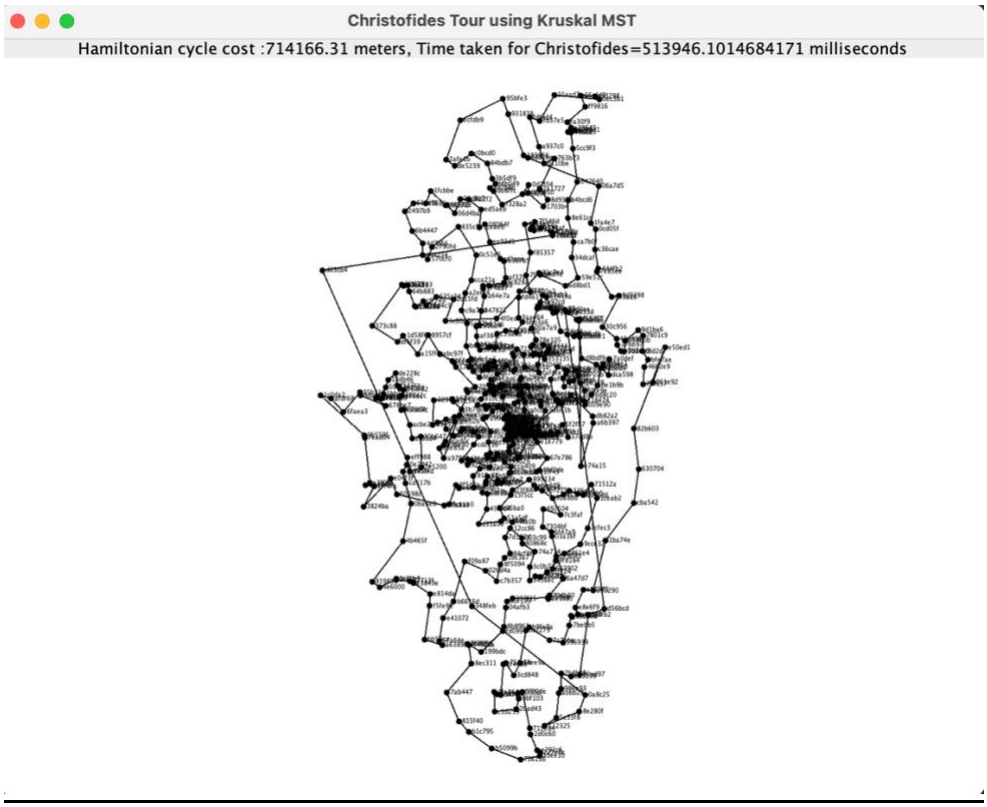


Outputs:

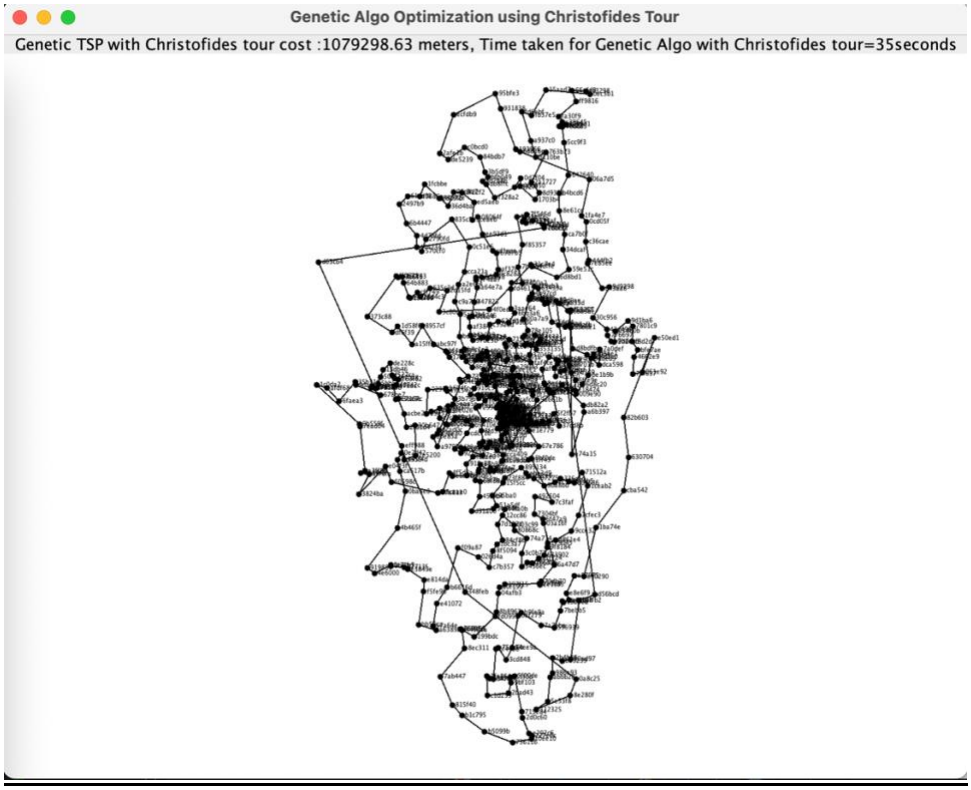
KruskalMST



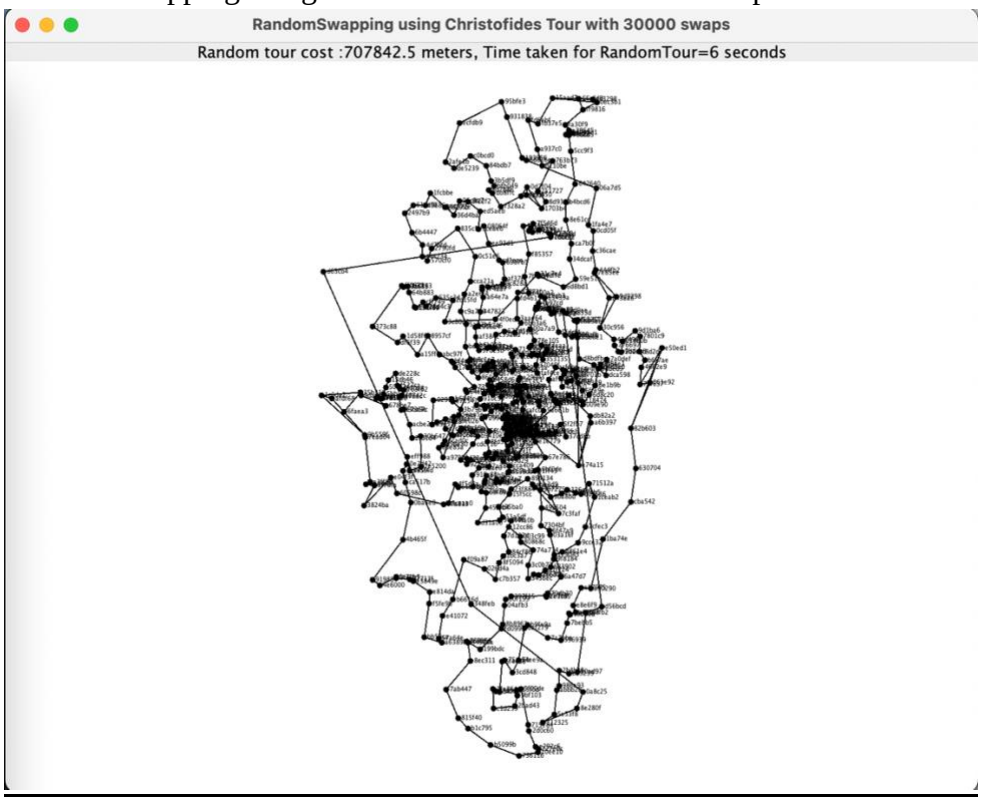
Christofides Tour Using Kruskal MST



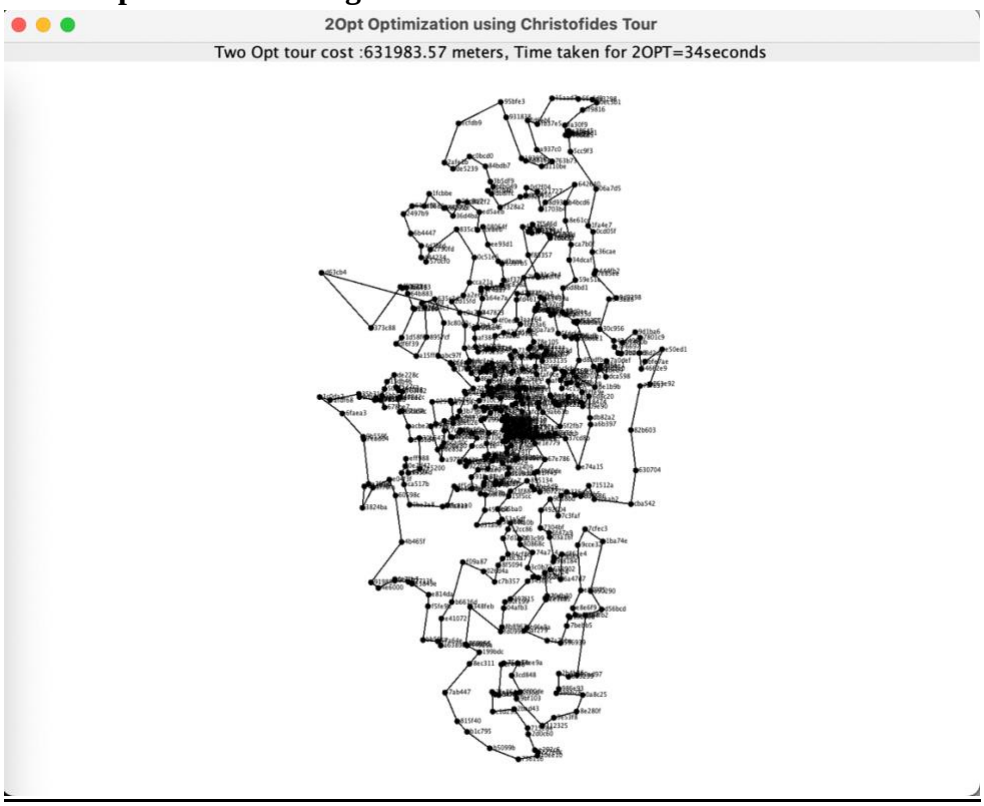
Genetic Algo Optimization using Christofides Tour



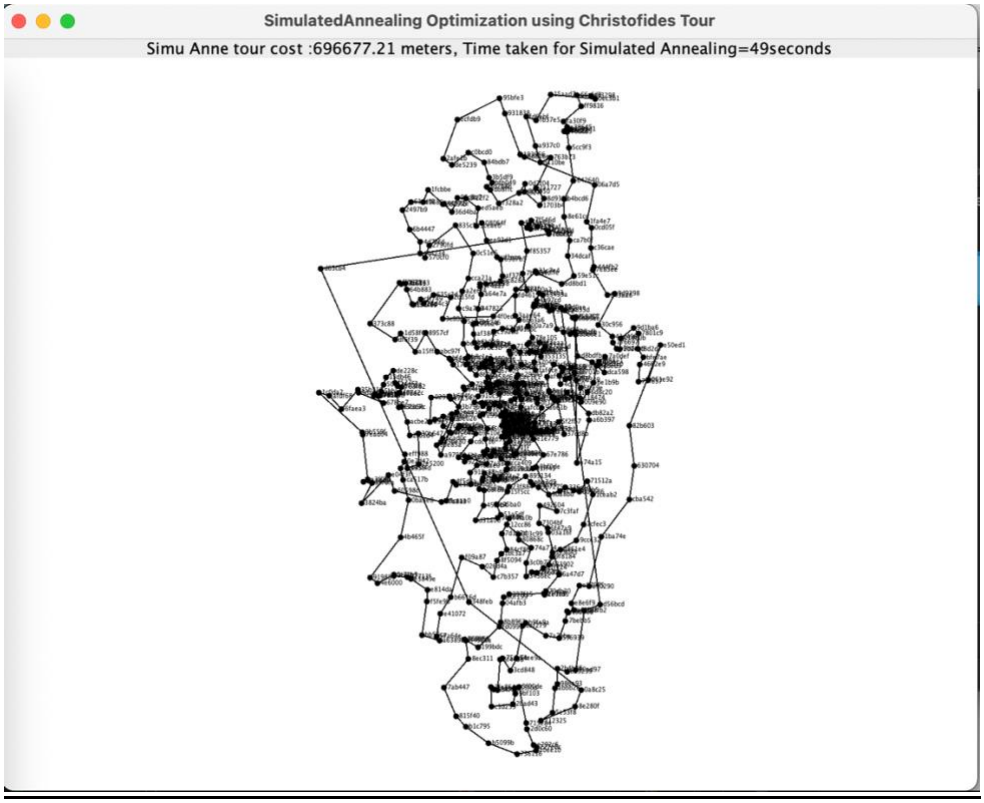
RandomSwapping using Christofides tour with 30000 swaps



2OPT Optimization Using Christofides Tour



Simulated Annealing Optimization Using Christofides Tour



More output images in GitHub repo: https://github.com/NeuYash/PSAProject/tree/main/PsaProject

Conclusion

Christofides TSP is an Optimization algorithm on an already close to optimal MST(but complex structure). We have implemented **10 Optimization techniques including 4 Tactical, 3 Strategic, 3 Hybrid.** Based on the analysis of the given dataset containing 585 points to cover, the best TSP tour was obtained using the 2OPT algorithm, resulting in a tour length of 631983.57 meters in 34 seconds. The Simulated Annealing algorithm using the 2OPT output produced the second-best result which is 639632.04 meters in 49 seconds. The 3OPT algorithm took more than an hour to produce results, while the Genetic Algorithm produced a result that was 1.3 times the 2OPT result. The RandomSwapping algorithm produced a result that was more than 1.1 times the 2OPT result, but it took less time than the 2OPT algorithm. Hence, based on the given observations, the **2OPT algorithm appears to be the most efficient** for obtaining the best TSP tour for the given dataset.

References

- https://docs.oracle.com/en/java/
- https://www.baeldung.com/cs/tsp-dynamic-programming
- https://en.wikipedia.org/wiki/Travellingsalesmanproblem
- https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/