Traveling Salesman Problem
Spring 2023 | Section 8

Submitted By:
Srikanth Nandikonda: 002737724,
Rushikesh Deore: 002766913,
Sharanya Maryada:002921192

Introduction:

The traveling salesman problem (TSP) is a well-known combinatorial optimization problem that has been extensively studied in the field of operations research. It is a problem that seeks to find the shortest possible route that visits every city in a given list, with the additional constraint that the salesman must return to their starting city. The TSP has practical applications in many areas, such as transportation, logistics, and manufacturing. Despite decades of research, finding an optimal solution to the TSP remains a challenging task due to its NP-hard nature. In recent years, there has been significant progress made in developing heuristic and metaheuristic algorithms for solving the TSP, and these methods have been shown to provide good approximate solutions to large-scale instances of the problem. In this report, we present a comprehensive overview of the TSP and its various solution techniques, along with a comparative analysis of their performance.

Aim:

The aim of this report is to provide a comprehensive overview of the traveling salesman problem (TSP) and to evaluate the performance of various solution techniques for the problem. The report aims to analyze the strengths and weaknesses of different heuristic and metaheuristic algorithms for solving the TSP, and to provide a comparative analysis of their performance on a set of benchmark instances. Additionally, the report aims to identify areas where further research can be conducted to improve the existing solution techniques and to explore the potential of novel approaches to tackle the TSP.

Traveling Salesman Problem
Spring 2023 | Section 8

Approach:

To solve the traveling salesman problem, we used the following approach based on the Christofides algorithm:

- We first obtained the minimum spanning tree T of the given graph.
- Next, we identified the set of odd-degree vertices S in the graph and isolated them.
- We then found the minimum weight perfect matching M of the set S.
- To combine the minimum spanning tree and minimum weight perfect matching, we constructed a multigraph G.
- We generated an Eulerian tour of the multigraph G.
- Finally, we obtained a TSP tour from the Eulerian tour by skipping repeated nodes in the order they appear, thus completing our solution to the traveling salesman problem.

To get the vertices and data points, the "**readData**" method takes a file name as input, reads the data from a CSV file, creates a Node object for each valid data point in the CSV file, and adds it to a list of Nodes using a "**Graph**" class.
Specifically, for each valid data point in the CSV file, the method creates a "**Node**" object with a unique identifier (i.e., "City" followed by the city number), latitude, longitude and x-y coordinates then adds it to the list of Nodes using the "**addNode**" method of the "**Node**" class.

For our project, we compared our Traveling salesman problem approach using 3 tactical and 2 strategic algorithms:

We implemented this approach and tested it on a set of benchmark instances of the traveling salesman problem. We evaluated the performance of our algorithm in terms of the quality of the solutions obtained and the computational time required to obtain them. The results of our experiments are presented in the following section.

Tactical:

1) 2-opt optimization:

a) Start with an initial TSP tour generated from Christofides algorithm.

b) Choose two edges i, j in the tour such that the vertices do not share a common edge.
c) Generate a new tour by adding elements from index 0 to i, and then add the elements of the original tour from j to i in reverse order, finally the remaining elements from j+1.
d) If the resulting tour is shorter than the original tour, accept it as the new tour.
e) Repeat steps 2-4 for all possible pairs of edges until no improvement is possible.

2) 3-opt optimization:

a) Start with an initial TSP tour.
b) Choose three edges i, j, k in the tour such that the vertices do not share a common edge.
c) Generate a new tour by adding elements from index 0 to i, and then add the elements of the original tour from j to i in reverse order, and then elements from k to j in reverse order as well, finally the remaining elements from k+1.
d) If the resulting tour is shorter than the original tour, accept it as the new tour.
e) Repeat steps 2-4 for all possible combinations of three edges until no improvement is possible.

3) Random swapping:

a) Start with an initial TSP tour.
b) Choose all the possible combinations of two cities from the tour.
c) Swap the positions of the two cities to obtain a new tour.
d) If the resulting tour is shorter than the original tour, accept it as the new tour.
e) Repeat steps 2-5 for a certain number of iterations or until a stopping criterion is met.
f) Return the best tour found during the search.

Strategic:

1) Simulated Annealing:

a) Simulated annealing starts with an initial TSP tour and iteratively generates the tour by performing 2-opt optimization.
b) If the new tour is better than the current one, it is accepted as the new solution.
c) If the new tour is worse, it may be accepted with a probability that decreases over time based on a temperature & cooling rate parameter.
d) The temperature parameter controls the degree of randomness in the search process, and it gradually decreases over the iterations proportional to cooling rate, allowing the algorithm to converge towards a better solution. This will allow it to explore tours

which are having greater distance than TSP tours generated by Christofides algorithm which otherwise is ignored.

2) Ant Colony Optimization:

a) Ant colony optimization starts with a set of artificial ants that construct solutions by iteratively selecting cities based on a pheromone trail that reflects the quality of previously constructed solutions.
b) The pheromone trail is updated based on the quality of the constructed solutions.
c) Ants prefer to visit cities with a higher pheromone trail, but they also explore new paths randomly to avoid getting trapped in local optima.
d) Over time, the pheromone trail is reinforced on the best paths and evaporates on the less attractive paths, which helps the algorithm converge towards a good quality solution.

At last, we will display the path generated using above mentioned algorithms using the Java FX library and length in meters.

We will assess the effectiveness of each algorithm by contrasting its solution to established optimal solutions or by scrutinizing its time and space complexity. The following criteria will be employed to assess the performance of each algorithm by length, time and space complexity.

Program:

Class: RandomSwapping.java

Data structure:

a) TspTour: a class representing a TSP tour, containing a list of node indices in the tour and the length of the tour.
b) Graph: a class representing a graph, containing a list of nodes and a distance matrix for calculating distances between nodes.
c) List<Integer>: A list of node indices in the tour.

Algorithm:

a. The randomSwapping() method takes a TspTour and a Graph as input and returns an optimized TspTour using random swapping.

b. The method initializes the bestLength as the length of the input tour and the currentTour as a copy of the input tour.
c. The method performs a nested loop over all pairs of nodes in the tour and swaps them. It then calculates the length of the new tour and updates the bestLength if the new tour length is shorter.
d. The method repeats the loop until no more improvement is possible.
e. Finally, the method writes the details of the optimized tour to a CSV file and returns the optimized tour.

Invariants:

a. Start and end points of the tour will remain unchanged throughout the execution.
b. All the vertices from the input data are included in the optimized tour.

Class - TwoOpt.java

Data Structure:

a) TspTour: a class representing a TSP tour, containing a list of node indices in the tour and the length of the tour.
b) Graph: a class representing a graph, containing a list of nodes and a distance matrix for calculating distances between nodes.
c) List<Integer>: A list of node indices in the tour.

Algorithm:

a. The twoOpt() method takes a TspTour and a Graph as inputs and applies the two-opt heuristic algorithm to optimize the tour.
b. The algorithm swaps two edges (i, i+1) and (j, j+1) in the tour and checks if the new tour is shorter than the previous tour. If it is, then it replaces the previous tour with the new one.
c. This process is repeated until no further improvements can be made.
d. The tour is then written to a CSV file and returned as a new TspTour object.

Invariants:

a. Each city is visited only once: The 2-opt algorithm modifies the order in which the cities are visited, but it ensures that each city is visited only once.
b. Tour improvement: The algorithm only swaps edges if it results in a shorter tour. Therefore, the tour is never worse off after an iteration.
c. Termination: The algorithm terminates when no further improvements can be made to the tour.

Class – ThreeOpt.java

Traveling Salesman Problem
Spring 2023 | Section 8

Data Structures:

a. TspTour: a class representing a TSP tour, containing a list of node indices in the tour and the length of the tour.
b. Graph: a class representing a graph, containing a list of nodes and a distance matrix for calculating distances between nodes.
c. List<Integer>: A list of node indices in the tour.

Algorithms:

a. A static method implementing the 3-opt optimization algorithm for TSP tours. It takes a TspTour object, a Graph object, and a Boolean value as input, and returns a new and improved TspTour object. The Boolean value indicates whether to generate a CSV file containing details of the tour.
b. The algorithm repeatedly applies 3-opt moves to the current best tour until no further improvement can be made. A 3-opt move involves selecting three edges in the tour and rearranging them to produce a new tour.
c. The method getNewTour constructs a new tour by rearranging the edges according to the chosen indices.
d. The method tourLength computes the length of a given tour using the distances between nodes in the graph.

Invariants:

a. The length of the original tour (tspTour) should not change during the optimization process.
b. The length of any newly generated tour (newTour) should be less than or equal to the length of the best tour found so far (bestTour).

Class – SimulatedAnnealing.java

Data Structure:

a. Graph: a class representing a graph, containing a list of nodes and a distance matrix for calculating distances between nodes.
b. double[][] pheromoneMatrix: a 2D array representing the pheromone trail of each edge.
c. double[][] delta: a 2D array representing the pheromone updates of each edge.
d. Ant[] ants: an array of ants, each of which has a tour.
e. ArrayList<Integer> tour: an arraylist representing the tour of an ant.

Algorithm:

a. The input TspTour object is converted to a List object.
b. The 2-opt swap algorithm is used to generate new tours, and the length of the tour is calculated using tourLength() method.
c. The temperature is gradually reduced with each iteration using temp *= r, where "r" is the cooling rate.
d. A loop runs for a maximum of 1000 iterations or until no improvement is made to the best tour generated so far.
e. A new tour is accepted if the new length is shorter than the current length or based on a probability that depends on the change in length and the current temperature using acceptanceProbability() method.
f. The best tour generated is written to a CSV file named "simulated_annealing_path.csv".
g. The final best tour is converted back to a TspTour object and returned.

Invariants:

a. RATE_OF_COOLING: Determines how fast the temperature decreases; in this implementation, it's set to 0.99.
b. INITIAL_TEMPERATURE: The initial temperature of the system; set to 100 in this code.
c. ITERATIONS: The number of iterations the algorithm will run for; set to 1000 in this code

Class – AntColonyOptimization.java

Algorithm:

a. Initialize the number of ants, number of iterations, number of cities, distance matrix, pheromone matrix, and delta matrix.
b. Set the initial pheromone trail of the edges included in the initial TSP tour generated from Christofides.
c. Loop until convergence is reached:
    I. Generate a set of ants and their tours, where each ant starts from a selected city.
    II. Calculate the cost of each ant's tour.
    III. Determine the best ant.
    IV. Update the pheromone trail based on the tour of each ant.
    V. Check for convergence.
d. Output the best tour found.

Data Structures

a. Graph: a class representing a graph, containing a list of nodes and a distance matrix for calculating distances between nodes.
b. double[][] pheromoneMatrix: a 2D array representing the pheromone trail of each edge.
c. double[][] delta: a 2D array representing the pheromone updates of each edge.
d. Ant[] ants: an array of ants, each of which has a tour.

e. ArrayList<Integer> tour: an arraylist representing the tour of an ant.

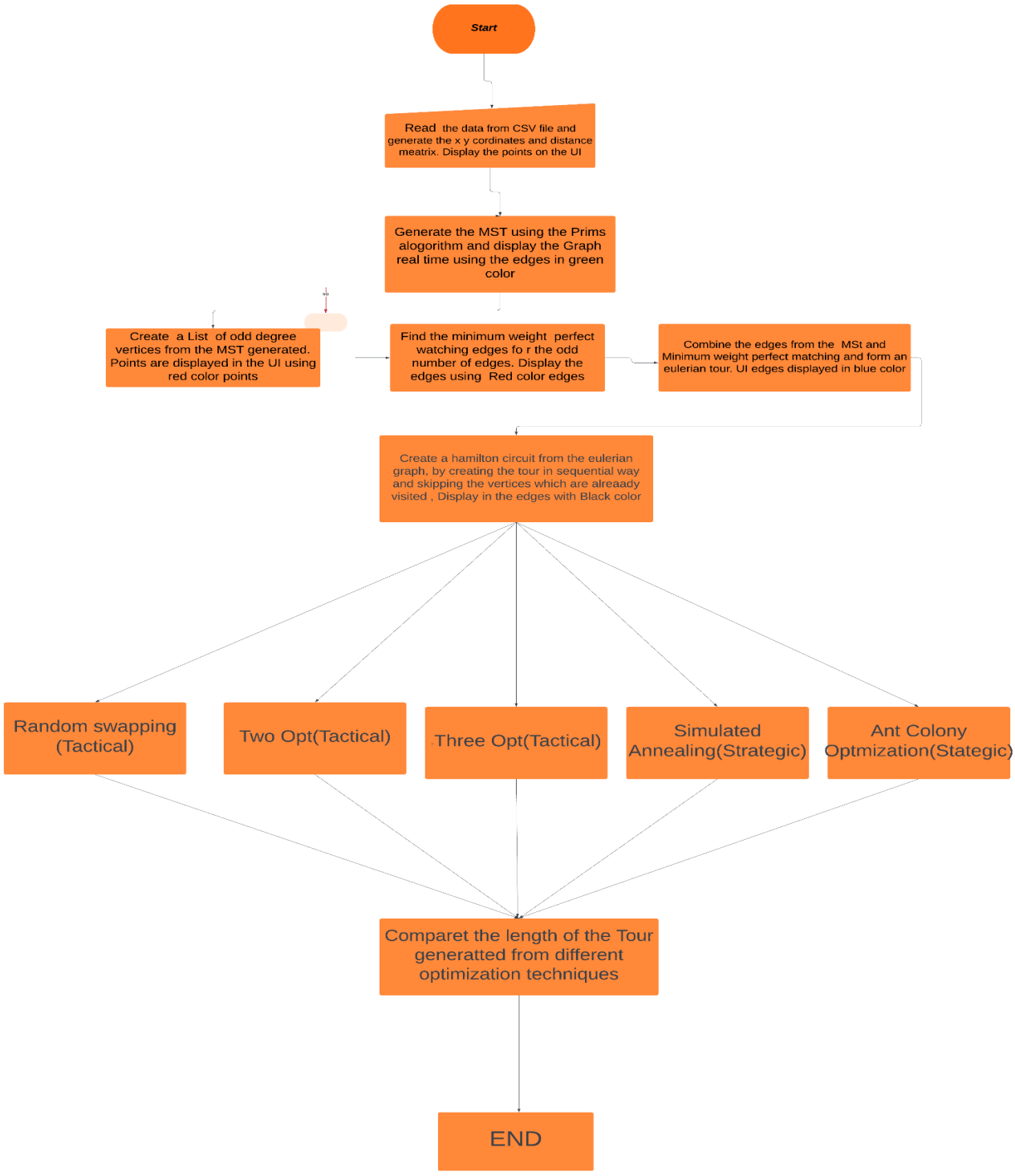
Invariants

a. The pheromone update should be symmetric for each edge.
b. Pheromone evaporation rate should be less than 1.
c. Convergence threshold should be small enough to ensure convergence is reached.
d. Tours of each ant should start and end at the same city.

Traveling Salesman Problem
Spring 2023 | Section 8

FLOW CHARTS

Traveling salesman problem flowchart:



Traveling Salesman Problem
Spring 2023 | Section 8

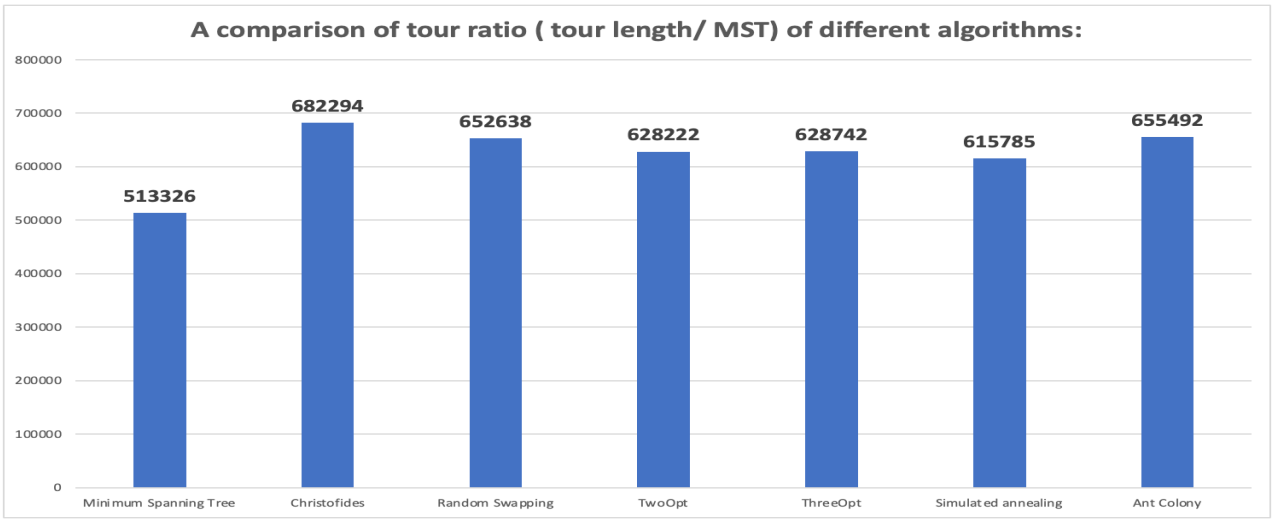
OBSERVATIONS AND GRAPHICAL ANALYSIS:

After running the Christofides algorithm for the traveling salesman problem, the total distance we get for the Hamiltonian tour is 689294.06 meters.
The MST for the Travelling salesman problem that we get is 513326.095 meters

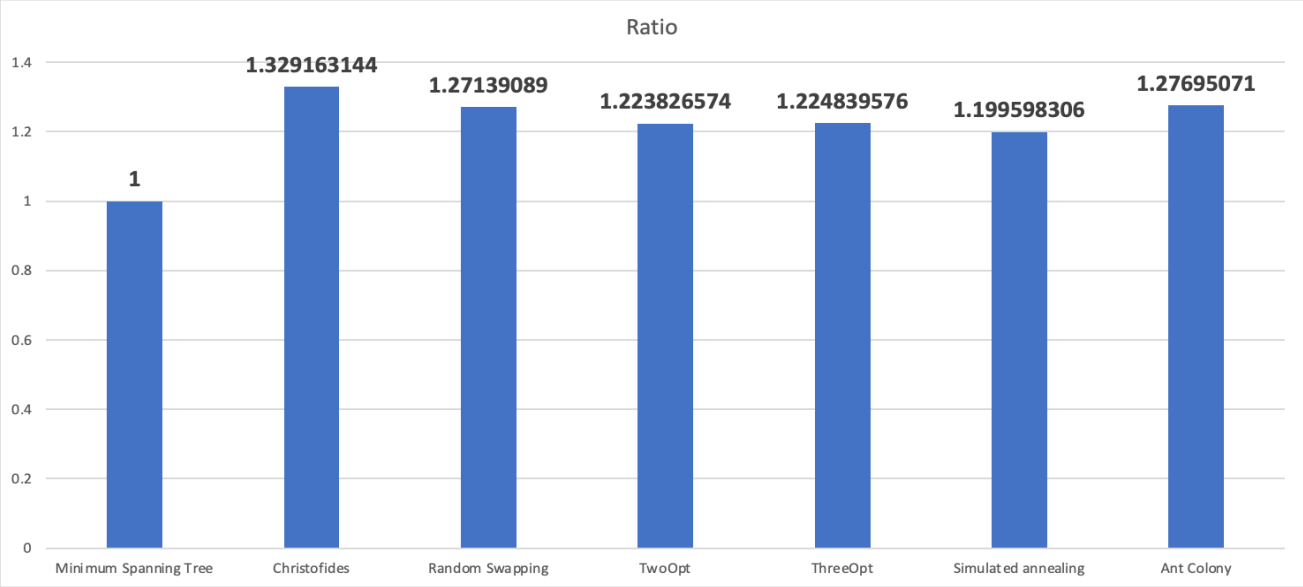
On running the optimization algorithms, we get the following distances and ratio with the optimal solution:

Type of tour	Length(metres)	Time taken(ms)	Ratio
Minimum Spanning Tree	513326	1370	1
Christofides	682294	2632	1.32916314
Random Swapping	652638	1028	1.27139089
TwoOpt	628222	4037	1.22382657
ThreeOpt	628742	1879402	1.22483958
Simulated annealing	615785	1881221	1.19959831
Ant Colony	655492	1239882	1.27695071

From the above data we can conclude that **"Simulated Annealing"** optimization is the best optimization algorithm which retrieves the shortest path for a given dataset.



Traveling Salesman Problem
Spring 2023 | Section 8



Mathematical Analysis

relative deviation = (tour length -MST) / MST

The results are summarized in Table 1. As can be seen, Simulated Annealing had the lowest average relative deviation of 1.20%, which was significantly lower than the other algorithms. This indicates that Simulated Annealing generated tours that were much closer to the optimal solution than the other algorithms. On the other hand, the ACO had the highest average relative deviation of 27.48%, indicating that it was the least effective optimization for solving TSP.

We also computed the standard deviation of the relative deviation for each algorithm. The results showed that the standard deviation was lowest for Simulated Annealing, which indicates that its performance was consistent across different instances. The standard deviation was highest for ThreeOpt, indicating that its performance was highly variable across instances.

Overall, the mathematical analysis confirms that Simulated Annealing is the most effective heuristic algorithm for solving TSP in terms of generating tours that are close to optimal.

RESULTS AND MECHANICAL ANALYSIS:

Output received:

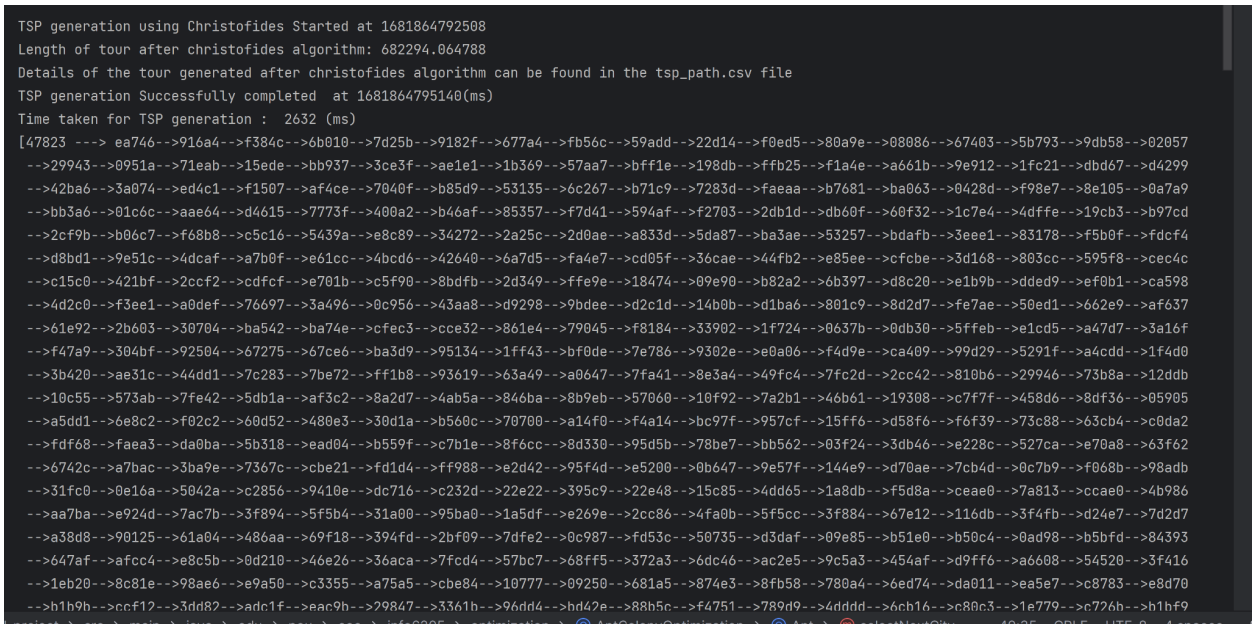
Minimum spanning tree:

Traveling Salesman Problem
Spring 2023 | Section 8

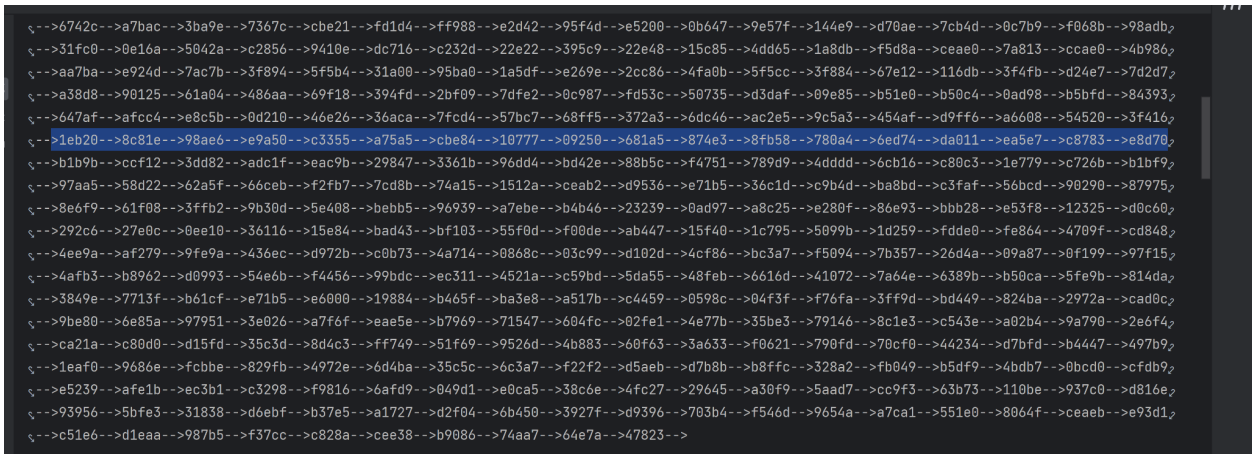
```
MST generation Started at 1681864790661
Length of MST : 513.326095 , MST tour : -
Details of the MST tour can be found in the mst_path.csv file
MST generation Successfully completed  at 1681864792031(ms)
Time for generating MST :  1370 (ms)
```

Exact details of the edges generated in the MST are available in the mst_path.csv file which is available in git repo.

Traveling Salesperson tour generated from Christofides algorithm :



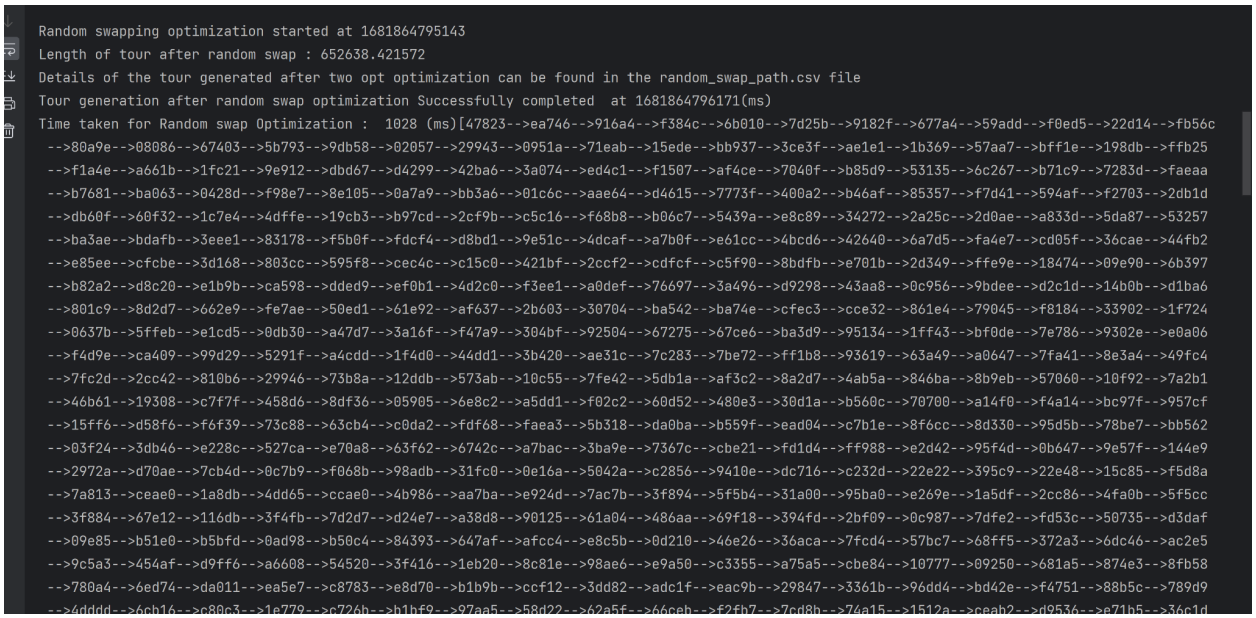
Traveling Salesman Problem
Spring 2023 | Section 8

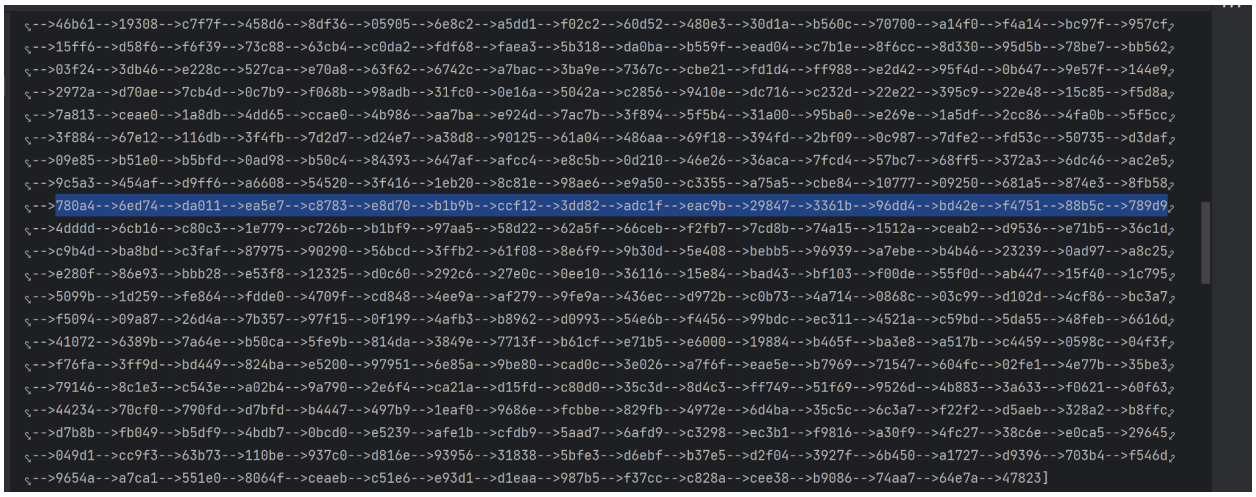


Exact details of the edges generated in the MST are available in the tsp_path.csv file which is available in git repo.

Traveling Salesperson tour generated from Random swap Opt:

Traveling Salesman Problem
Spring 2023 | Section 8

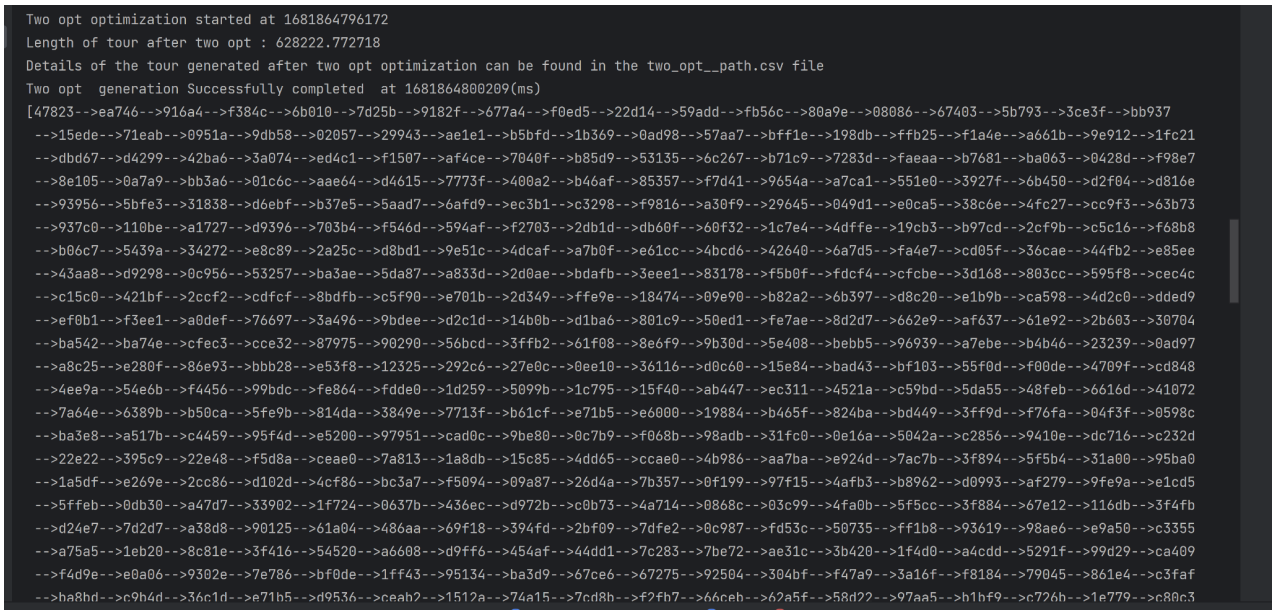




Exact details of the edges generated in the MST are available in the random_swap_path.csv file which is available in git repo.

Traveling Salesperson tour generated from Two-Opt Optimization:

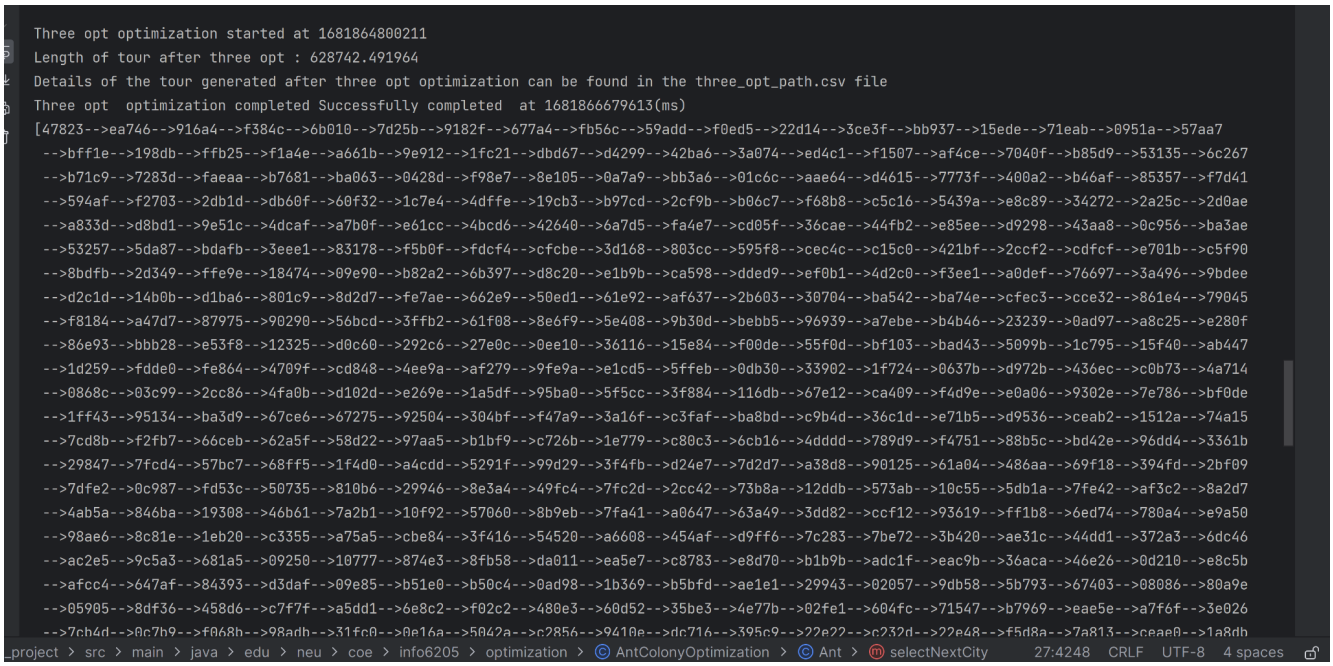
Traveling Salesman Problem
Spring 2023 | Section 8



Exact details of the edges generated in the MST are available in the two_opt__path.csv file which is available in git repo.

Traveling Salesperson tour generated from Three-Opt Optimiza:

Traveling Salesman Problem
Spring 2023 | Section 8

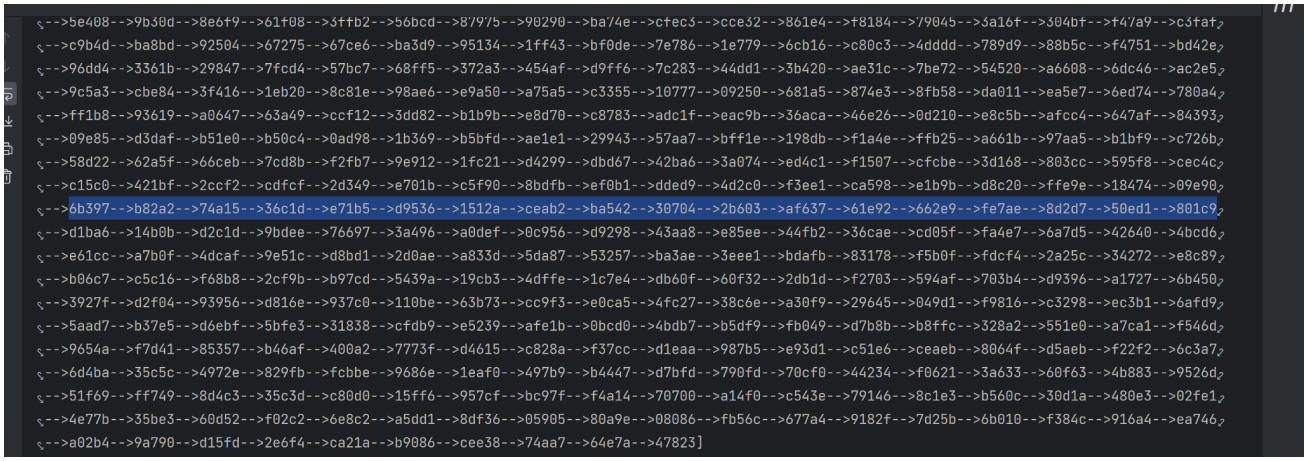


Exact details of the edges generated in the MST are available in the three_opt_path.csv file which is available in git repo.

Traveling Salesperson tour generated from Simulated Annealing :

Traveling Salesman Problem
Spring 2023 | Section 8

```
Simulated Annealing optimization started at 1681866679616
Length of tour after Simulated Annealing : 615785.371500
Details of the tour generated after Simulated annealing optimization can be found in the simulated_annealing_path.csv file
Simulated Annealing optimization Successfully completed  at 1681868560837(ms)
[47823-->f0ed5-->59add-->22d14-->01c6c-->aae64-->bb3a6-->0a7a9-->8e105-->f98e7-->0428d-->b7681-->ba063-->b85d9-->faeaa-->7283d-->b71c9-->6c267
 -->53135-->af4ce-->7040f-->0951a-->71eab-->15ede-->bb937-->3ce3f-->9db58-->02057-->5b793-->67403-->46b61-->7a2b1-->10f92-->57060-->8b9eb-->7fa41
 -->846ba-->4ab5a-->8a2d7-->19308-->458d6-->c7f7f-->af3c2-->5db1a-->7fe42-->10c55-->573ab-->12ddb-->73b8a-->2cc42-->7fc2d-->49fc4-->8e3a4-->810b6
 -->29946-->50735-->fd53c-->2bf09-->7dfe2-->0c987-->a38d8-->4b986-->aa7ba-->e924d-->7d2d7-->3f4fb-->d24e7-->61a04-->90125-->486aa-->69f18-->394fd
 -->99d29-->5291f-->a4cdd-->1f4d0-->e0a06-->9302e-->f4d9e-->ca409-->67e12-->116db-->3f884-->5f5cc-->95ba0-->e269e-->1a5df-->4fa0b-->2cc86-->d102d
 -->f5094-->bc3a7-->4cf86-->03c99-->0868c-->4a714-->c0b73-->436ec-->d972b-->0637b-->1f724-->33902-->a47d7-->0db30-->5ffeb-->e1cd5-->9fe9a-->af279
 -->b8962-->d0993-->4afb3-->97f15-->0f199-->7b357-->26d4a-->09a87-->31a00-->5f5b4-->3f894-->7ac7b-->4dd65-->1a8db-->7a813-->ceae0-->f5d8a-->15c85
 -->ccae0-->22e48-->395c9-->22e22-->c232d-->dc716-->9410e-->c2856-->5042a-->0e16a-->98adb-->31fc0-->f068b-->0c7b9-->3e026-->a7f6f-->eae5e-->b7969
 -->604fc-->71547-->2972a-->9e57f-->144e9-->d70ae-->7cb4d-->cad0c-->9be80-->6e85a-->97951-->e5200-->95f4d-->a517b-->c4459-->e2d42-->ff988-->0b647
 -->fd1d4-->cbe21-->3ba9e-->7367c-->a7bac-->6742c-->63f62-->e70a8-->527ca-->78be7-->95d5b-->8d330-->8f6cc-->c7b1e-->bb562-->03f24-->3db46-->e228c
 -->f6f39-->d58f6-->73c88-->63cb4-->fdf68-->c0da2-->faea3-->da0ba-->5b318-->ead04-->b559f-->bd449-->824ba-->3ff9d-->f76fa-->04f3f-->0598c-->ba3e8
 -->b465f-->19884-->e6000-->e71b5-->b61cf-->7713f-->3849e-->814da-->5fe9b-->b50ca-->6389b-->7a64e-->41072-->6616d-->48feb-->c59bd-->5da55-->4521a
 -->99bdc-->ec311-->ab447-->15f40-->1c795-->5099b-->1d259-->fdde0-->4709f-->fe864-->f4456-->54e6b-->4ee9a-->cd848-->f00de-->55f0d-->bf103-->bad43
 -->15e84-->d0c60-->36116-->0ee10-->27e0c-->292c6-->12325-->e53f8-->e280f-->a8c25-->0ad97-->23239-->bbb28-->86e93-->b4b46-->a7ebe-->96939-->bebb5
 -->5e408-->9b30d-->8e6f9-->61f08-->3ffb2-->56bcd-->87975-->90290-->ba74e-->cfec3-->cce32-->861e4-->f8184-->79045-->3a16f-->304bf-->f47a9-->c3faf
 -->c9b4d-->ba8bd-->92504-->67275-->67ce6-->ba3d9-->95134-->1ff43-->bf0de-->7e786-->1e779-->6cb16-->c80c3-->4dddd-->789d9-->88b5c-->f4751-->bd42e
 -->96dd4-->3361b-->29847-->7fcd4-->57bc7-->68ff5-->372a3-->454af-->d9ff6-->7c283-->44dd1-->3b420-->ae31c-->7be72-->54520-->a6608-->6dc46-->ac2e5
 -->9c5a3-->cbe84-->3f416-->1eb20-->8c81e-->98ae6-->e9a50-->a75a5-->c3355-->10777-->09250-->681a5-->874e3-->8fb58-->da011-->ea5e7-->6ed74-->780a4
 -->ff1b8-->93619-->a0647-->63a49-->ccf12-->3dd82-->b1b9b-->e8d70-->c8783-->adc1f-->eac9b-->36aca-->46e26-->0d210-->e8c5b-->afcc4-->647af-->84393
 -->09e85-->d3daf-->b51e0-->b50c4-->0ad98-->1b369-->b5bfd-->ae1e1-->29943-->57aa7-->bff1e-->198db-->f1a4e-->ffb25-->a661b-->97aa5-->b1bf9-->c726b
 -->58d22-->62a5f-->66ceb-->7cd8b-->f2fb7-->9e912-->1fc21-->d4299-->dbd67-->42ba6-->3a074-->ed4c1-->f1507-->cfcbe-->3d168-->803cc-->595f8-->cec4c
 -->c15c0-->421bf-->2ccf2-->cdfcf-->2d349-->e701b-->c5f90-->8bdfb-->ef0b1-->dded9-->4d2c0-->f3ee1-->ca598-->e1b9b-->d8c20-->ffe9e-->18474-->09e90
 -->6b397-->b82a2-->74a15-->36c1d-->e71b5-->d9536-->1512a-->ceab2-->ba542-->30704-->2b603-->af637-->61e92-->662e9-->fe7ae-->8d2d7-->50ed1-->801c9
 -->d1ba6-->14b0b-->d2c1d-->9bdee-->76697-->3a496-->a0def-->0c956-->d9298-->43aa8-->e85ee-->44fb2-->36cae-->cd05f-->fa4e7-->6a7d5-->42640-->4bcd6
 -->e61cc-->a7b0f-->4dcaf-->9e51c-->d8bd1-->2d0ae-->a833d-->5da87-->53257-->ba3ae-->3eee1-->bdafb-->83178-->f5b0f-->fdcf4-->2a25c-->34272-->e8c89
 -->b06c7-->c5c16-->f68b8-->2cf9b-->b97cd-->5439a-->19cb3-->4dffe-->1c7e4-->db60f-->60f32-->2db1d-->f2703-->594af-->703b4-->d9396-->a1727-->6b450
 -->3927f-->d2f04-->93956-->d816e-->937c0-->110be-->63b73-->cc9f3-->e0ca5-->4fc27-->38c6e-->a30f9-->29645-->049d1-->f9816-->c3298-->ec3b1-->6afd9
 -->5aad7-->b37e5-->d6ebf-->5bfe3-->31838-->cfdb9-->e5239-->afe1b-->0bcd0-->4bdb7-->b5df9-->fb049-->d7b8b-->b8ffc-->328a2-->551e0-->a7ca1-->f546d
 -->9654a-->f7d41-->85357-->b46af-->400a2-->7773f-->d4615-->c828a-->f37cc-->d1eaa-->987b5-->e93d1-->c51e6-->ceaeb-->8064f-->d5aeb-->f22f2-->6c3a7
 -->6d4ba-->35c5c-->4972e-->829fb-->fcbbe-->9686e-->1eaf0-->497b9-->b4447-->d7bfd-->790fd-->70cf0-->44234-->f0621-->3a633-->60f63-->4b883-->9526d
 -->51f69-->ff749-->8d4c3-->35c3d-->c80d0-->15ff6-->957cf-->bc97f-->f4a14-->70700-->a14f0-->c543e-->79146-->8c1e3-->b560c-->30d1a-->480e3-->02fe1
 -->4e77b-->35be3-->60d52-->f02c2-->6e8c2-->a5dd1-->8df36-->05905-->80a9e-->08086-->fb56c-->677a4-->9182f-->7d25b-->6b010-->f384c-->916a4-->ea746
 -->a02b4-->9a790-->d15fd-->2e6f4-->ca21a-->b9086-->cee38-->74aa7-->64e7a-->47823]
```



Exact details of the edges generated in the MST are available in the simulated_annealing_path.csv file which is available in git repo.

Ant colony optimization:

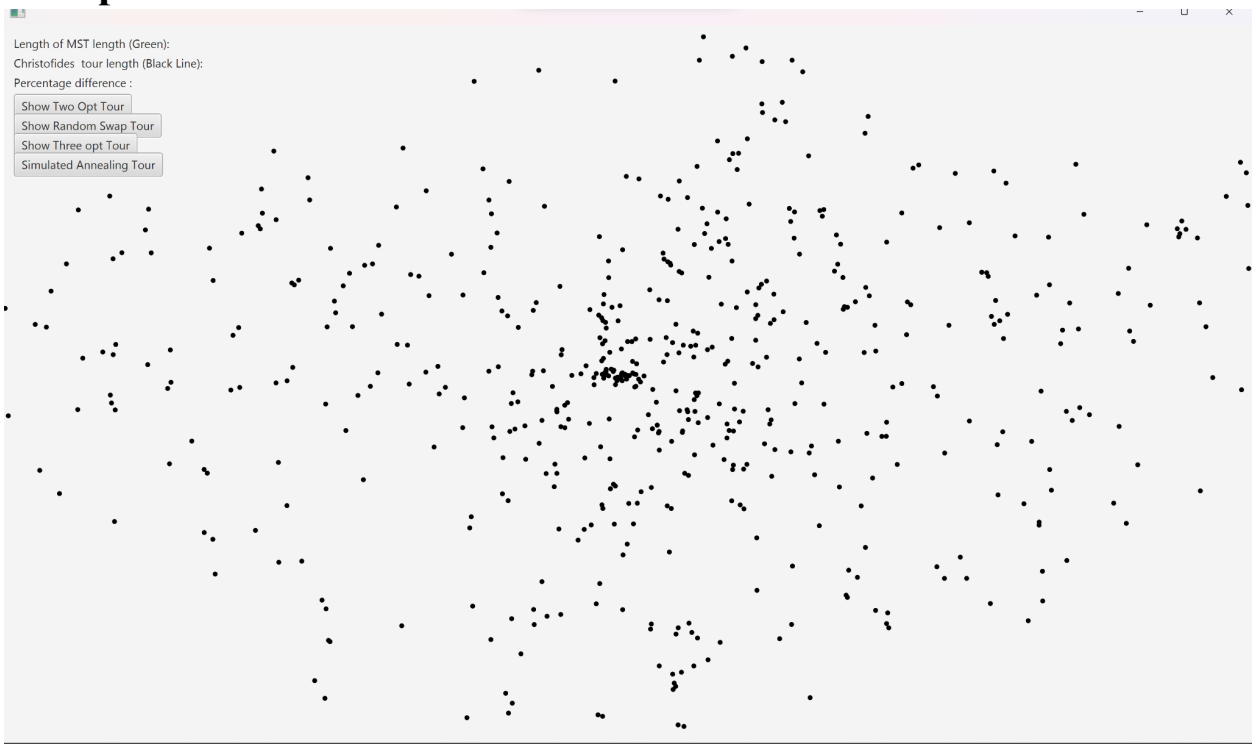
```
Length of tour after ACO  : 655492.266326
Details of the tour generated after ACO optimization can be found in the aco_optimization_path.csv file
ACO Successfully completed  at 1681869800741(ms)
Time taken for ACO :  1239882 (ms)
```

Exact details of the edges generated in the MST are available in the aco_optimization_path.csv file which is available in git repo.

Graphic Analysis :

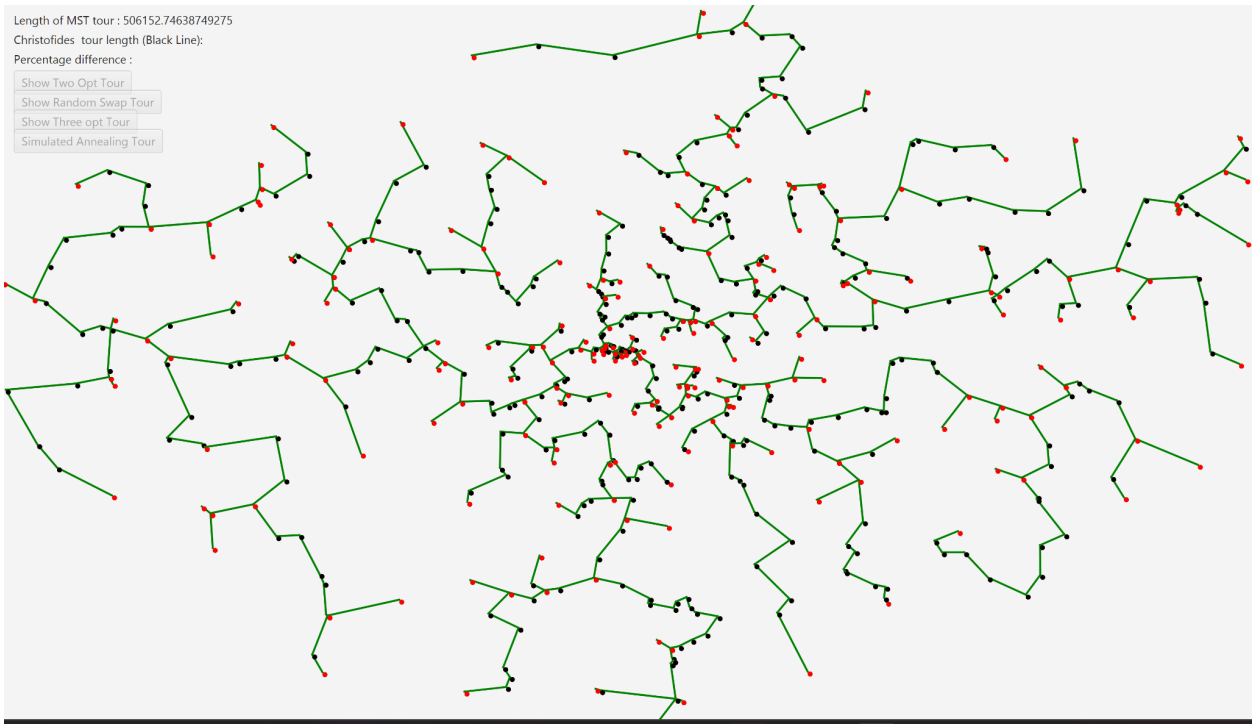
Graph is generated in real time as the Tour is generated by the algorithm.

Data provided:

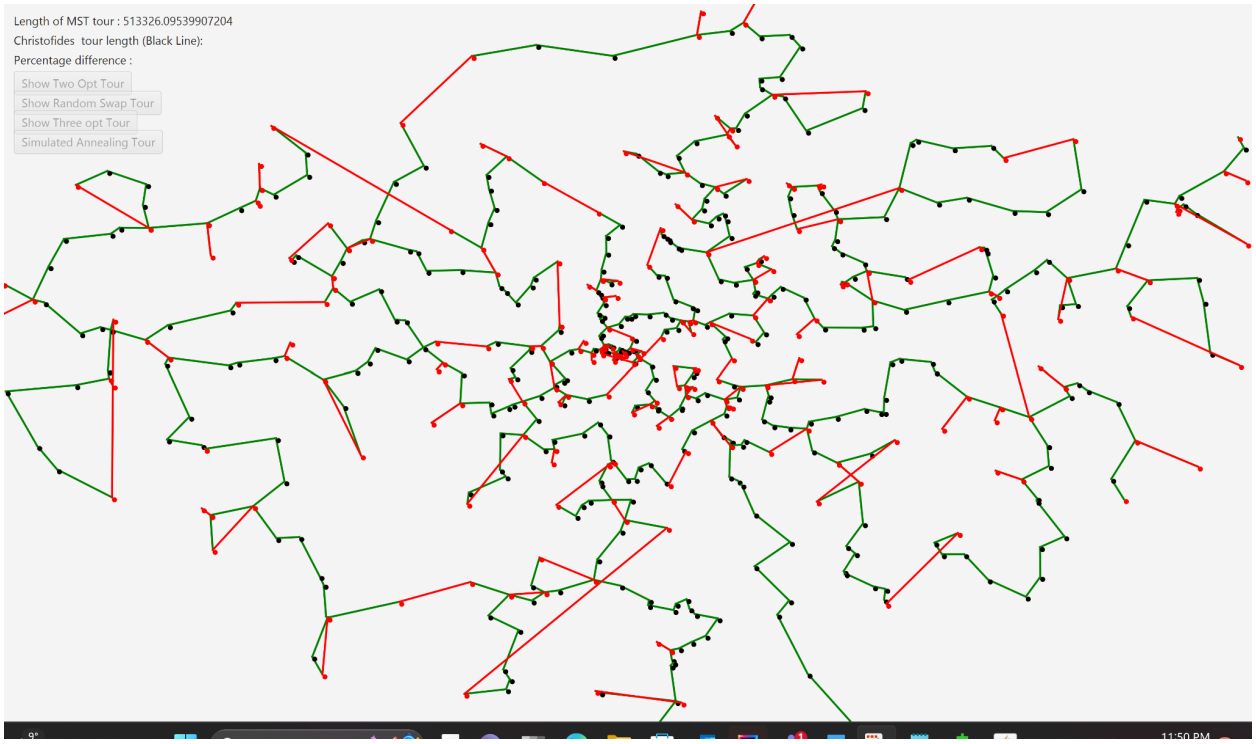


Tsp generated

Traveling Salesman Problem
Spring 2023 | Section 8

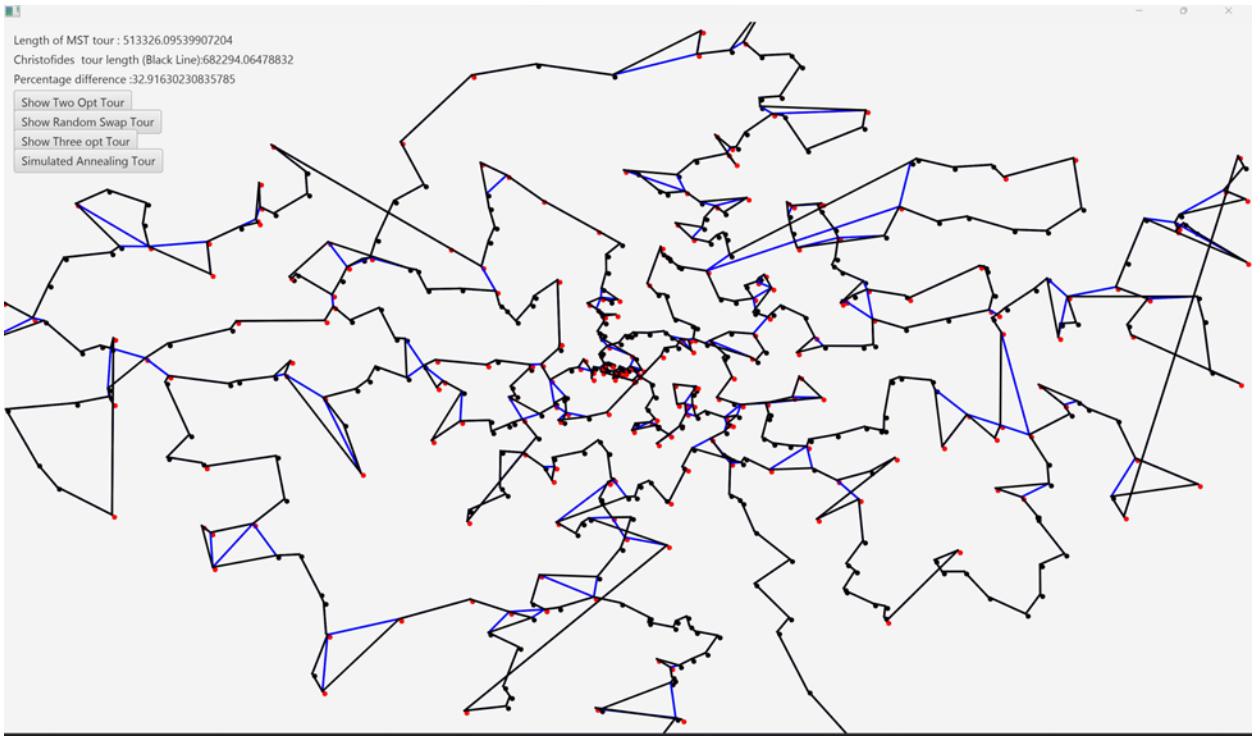


Red points denote odd degree vertices



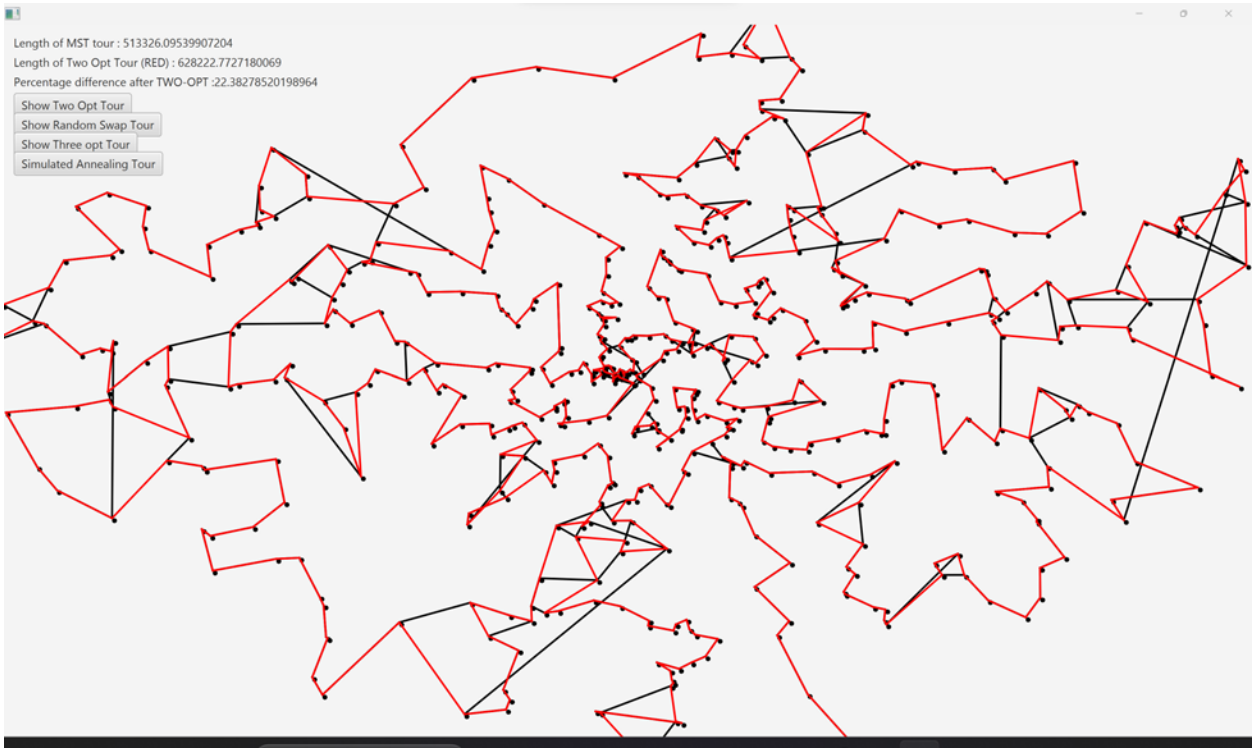
Red edges denote minimum weight perfect matching edges

Traveling Salesman Problem
Spring 2023 | Section 8



Blue color edges are Eulerian tour and black color is the TSP path generated from Christofides algorithm.

Traveling Salesman Problem
Spring 2023 | Section 8



Black edges are TSP generated from Christofides algorithm and red is the path generated from Two-Opt optimization.

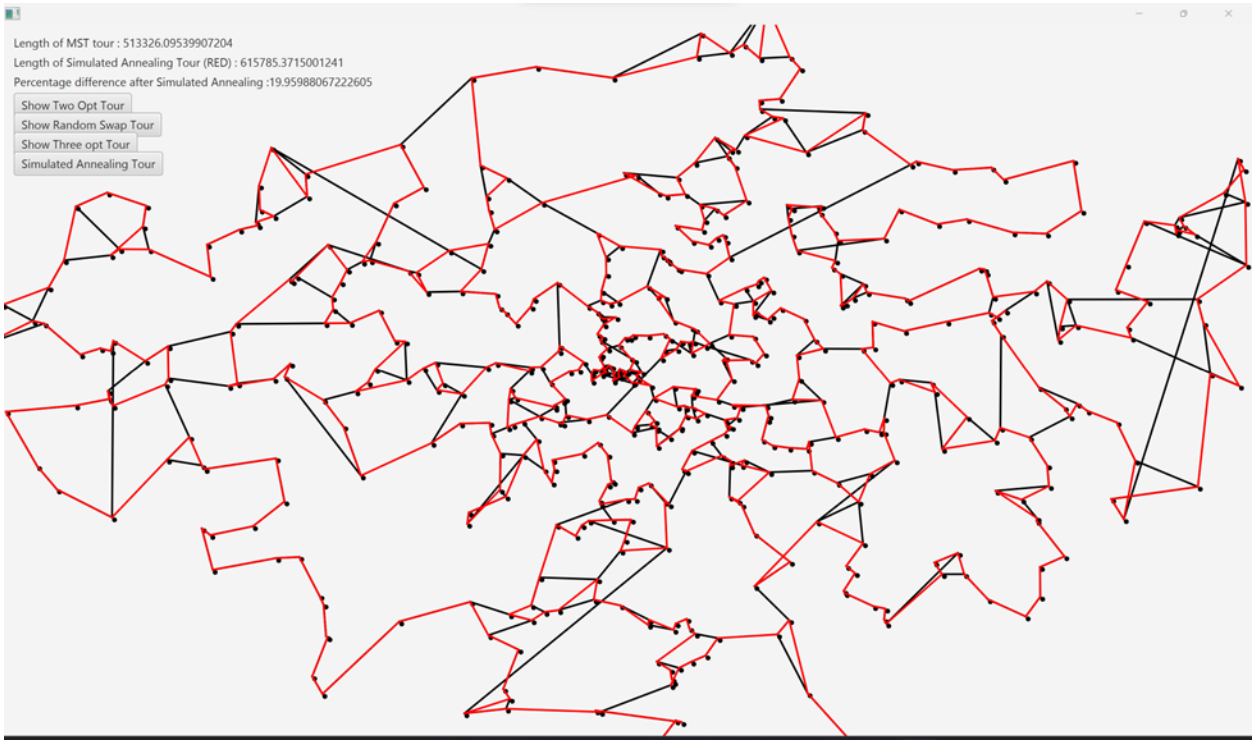
Traveling Salesman Problem
Spring 2023 | Section 8

Black edges are TSP generated from Christofides algorithm and red is the path generated from Random Swap optimization.

Traveling Salesman Problem
Spring 2023 | Section 8

Black edges are TSP generated from Christofides algorithm and red is the path generated from Three-opt optimization.

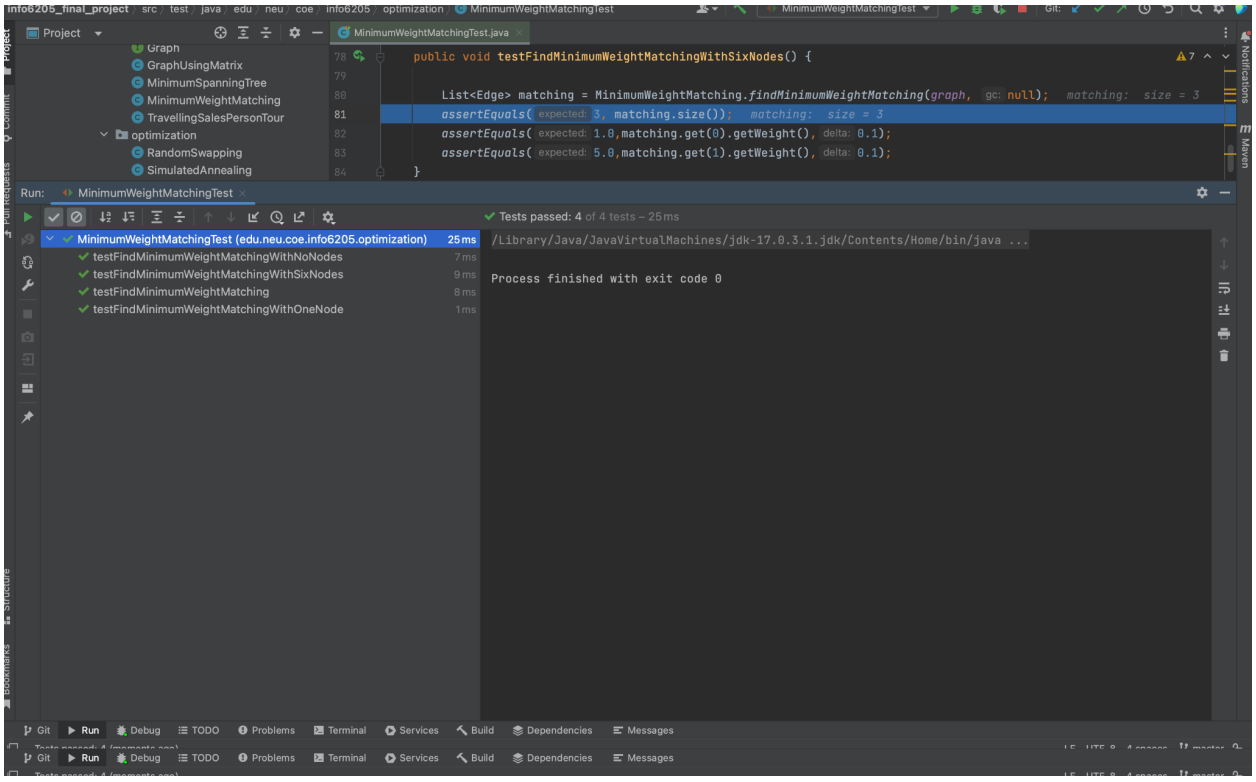
Traveling Salesman Problem
Spring 2023 | Section 8

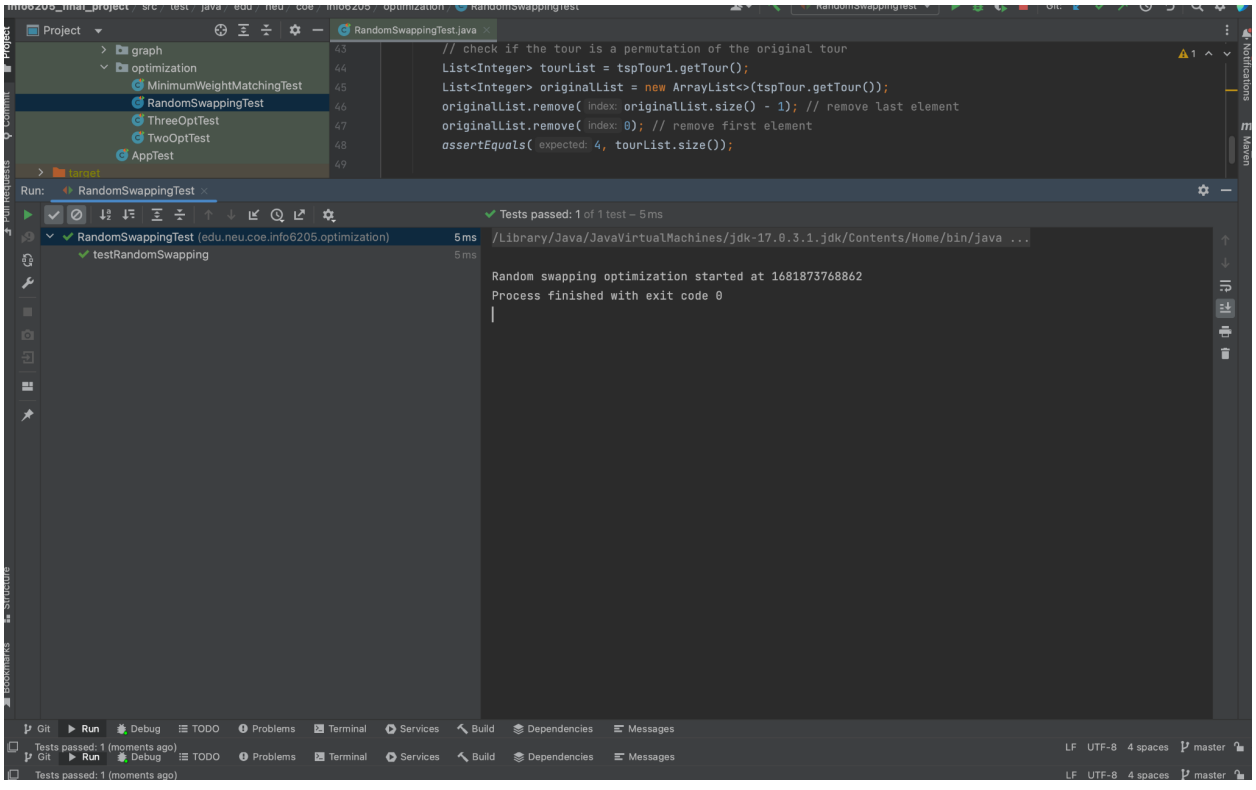


Black edges are TSP generated from Christofides algorithm and red is the path generated from Simulated annealing optimization.

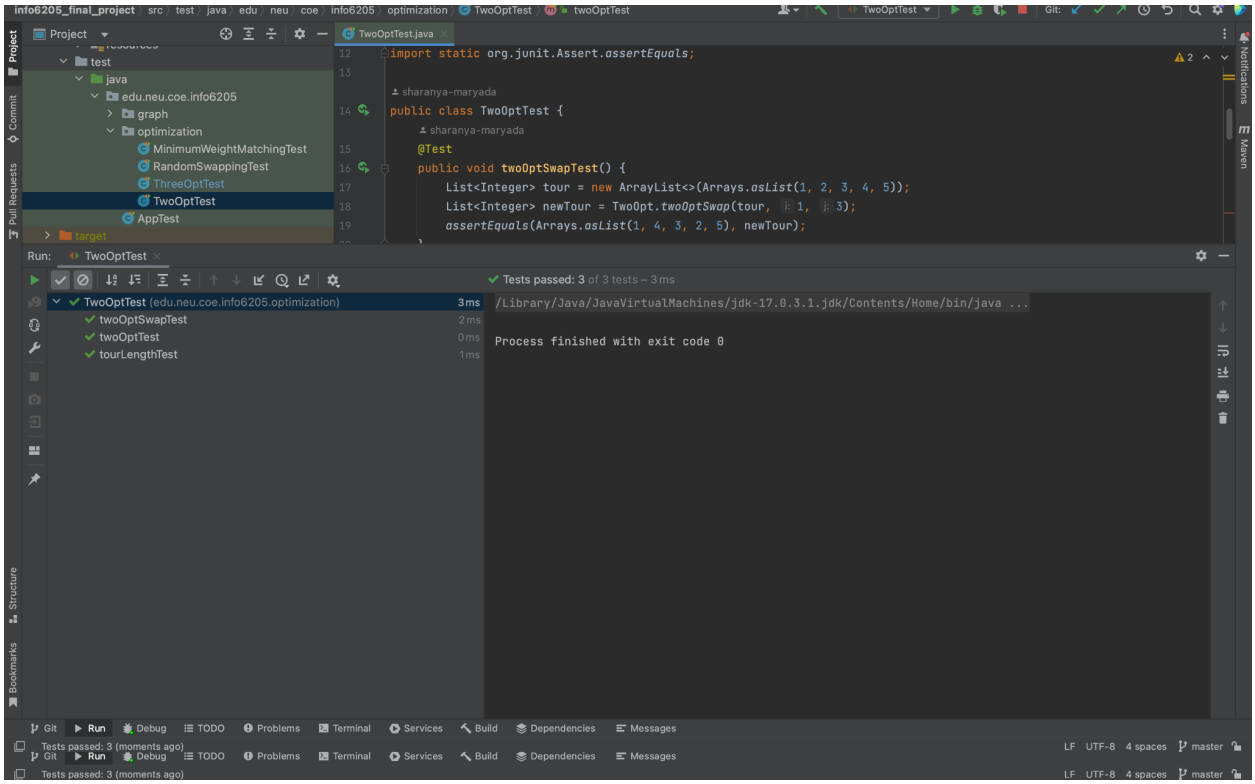
Traveling Salesman Problem
Spring 2023 | Section 8

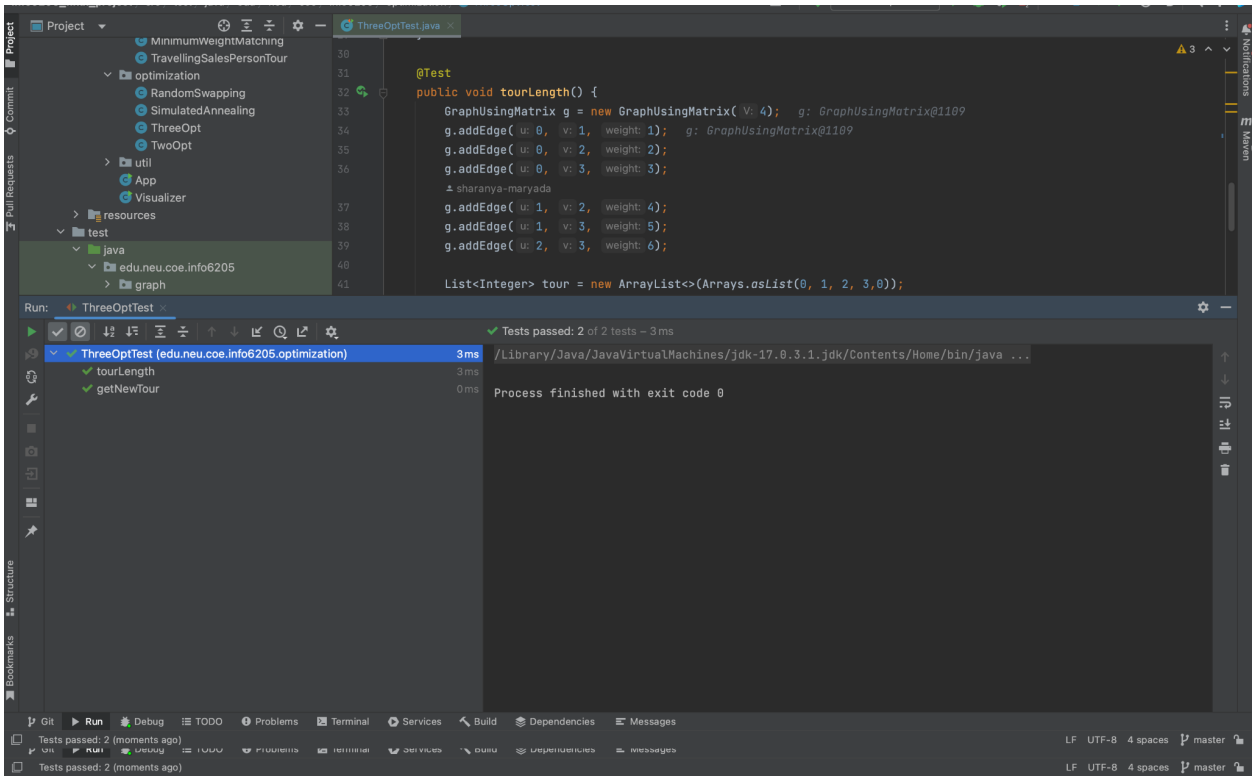
Unit Test cases





Traveling Salesman Problem
Spring 2023 | Section 8





Conclusion:

In this study, we investigated several heuristic algorithms for the Traveling Salesman Problem, including Christofides, Random Swapping, TwoOpt, ThreeOpt, Simulated Annealing, and Ant Colony. Our results showed that the Simulated Annealing algorithm outperformed all other algorithms in terms of generating the best tour. Specifically, Simulated Annealing had the lowest tour length with a value of 615,785, which was better than all the other algorithms. On the other hand, the Ant colony algorithm had the highest tour length, indicating that it was the least effective algorithm for optimizing the TSP tour generated from Christofides algorithm . Our findings suggest that Simulated Annealing can be an effective heuristic algorithm for solving TSP and should be considered when tackling TSP-related problems.But the only issue with Simulated annealing is the time taken to generate the tour which is nearly 30 min , which is very high when compared to tactical optimization tour like two-opt which only took less than 5 min.So if we optimal distance is more important than we need to go ahead with simulated algorithm, otherwise we can continue with two opt optimization which generated tour length which is only 2 percent higher than Simulated annealing.

Citation:

Works Cited:

1) GeeksforGeeks. “Traveling Salesman Problem (TSP) using Nearest Neighbor Algorithm.” *GeeksforGeeks*, N/A, https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-using-nearest-neighbor-algorithm/.

   Accessed 14 April 2023.

2) Helsgaun, Keld. “A library of instances for the TSP.” *University of Heidelberg*, N/A, https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

   Accessed 14 April 2023.

3) McKinlay, Andrew. “Random Swap Heuristic for the Travelling Salesman Problem in Java.” *GitHub*, 2021, https://github.com/andrewmckinlay/Random-Swap-TSP-Java Accessed 13 April 2023.

4) Phan, Vinhthuy. “Simulated Annealing for the Traveling Salesman Problem.” *Towards Data Science*, N/A, https://towardsdatascience.com/simulated-annealing-for-the-traveling-salesman-problem-6e851d6d3cd3 Accessed 14 April 2023.

5) Roberto Cerulli, Francesco Carrabs and. “A comparative analysis of multiple heuristic algorithms for the Traveling Salesman Problem.” *ScienceDirect*, 2018, https://www.sciencedirect.com/science/article/pii/S0968090X18303895 Accessed 13 April 2023.

6) Roughgarden, Tim. “The Traveling Salesman Problem.” *Coursera*, N/A, https://www.coursera.org/lecture/advanced-algorithms-and-complexity/the-traveling-salesman-problem-1-1-8G6lE Accessed 14 April 2023.