

**CST3170 Artificial Intelligence**

**Coursework 1: Travelling Salesman**

**Dereck Lam Hon Wah**

**M00826933**

**Date of Submission: 07.12.23**

**Lab Tutor: Aditya Santokhee**

Table of Contents

[Travelling Salesman Problem 2](#_Toc1223442109)

[Self-Marking Sheet 3](#_Toc2128217076)

[Algorithms Used 3](#_Toc1648573513)

[Nearest Neighbor 3](#_Toc269255842)

[Description 3](#_Toc1743166820)

[Implementation 4](#_Toc933179359)

[Justification 4](#_Toc1304740796)

[Dijkstra Algorithm 4](#_Toc1628497612)

[Description 4](#_Toc1269115043)

[Implementation 4](#_Toc260783762)

[Justification 5](#_Toc2060545978)

[Prim’s Algorithm (Minimum Spanning Tree) 5](#_Toc624030091)

[Description 5](#_Toc1570676561)

[Implementation 5](#_Toc1461133775)

[Justification 5](#_Toc141842195)

[Test Result 6](#_Toc855437313)

[Train 1 6](#_Toc1384874281)

[Train 2 6](#_Toc1319405363)

[Train 3 6](#_Toc2004065985)

[Test 1 6](#_Toc16724125)

[Test 2 6](#_Toc1093996445)

[Test 3 7](#_Toc1853960517)

[Test 4 7](#_Toc1808269138)

[Conclusion 7](#_Toc1407967474)

[Reference List 8](#_Toc1054413911)

# Travelling Salesman Problem

The Travelling Salesman Problem (TSP) requires that a salesman traverse several cities exactly once each and return to its starting point at the end by using an optimal route. A graph often represents the TSP where the nodes are the cities, and the edges or weight are the distance between two connected cities. It is an NP-hard problem, and many algorithms have tried to solve it, but there is no optimal solution yet (Binyamin Adeniyi Ajayi et al., 2022).

# Self-Marking Sheet

|  |  |  |  |
| --- | --- | --- | --- |
| **Points** | **Area** | **Self-Marking Points** | **Opinions** |
| 10 | Self-Marking Sheet | 10 | Self-Marking Sheet with opinions filled and submitted. |
| 10 | Solve First Training Problem | 10 | Based on the sample and train files provided, TSP has been solved. |
| 10 | Get Optimal Result for all three Training problems. | 10 | Based on the sample and train files provided, TSP has been solved. |
| 10 | Describe Algorithm(s) used | 10 | Clear definitions, implementation and justification of the algorithms used are provided. |
| 10 | Quality of Code | 10 | Java conventions are implemented and respected using university resources. |
| 20 | Get Optimal Results for the First Three Tests. | 20 | Based on the sample and train files provided, TSP has been solved using 3 different algorithms. |
| 20 | Get Optimal Results for First Three Tests in under a minute | 20 | Based on the sample and train files provided, TSP has been solved using 3 different algorithms in under one minute. |
| 10 | Best System on Fourth Test (Path length squared times time). | 10 | The 3-algorithm implemented has been run successfully on the test file provided. |

# Algorithms Used

## Nearest Neighbor

### Description

It is a quick and easy algorithm implemented as an initial heuristic solution for the Traveling Salesman Problem (Nuraiman et al., 2018). However, it is a greedy method, as the algorithm doesn’t start by building an initial tour but starts from a fixed or arbitrary node and adds the nearest unvisited node until there are no unvisited nodes left and the algorithm ends (Emmanuel Oluwatobi Asani, Okeyinka and Ayodele Ariyo Adebiyi, 2020). Using Euclidean distance, it calculates the distance between two nodes and the choice of the starting node may outcome to different results. (Nuraiman et al., 2018).

### Implementation

1. Select an arbitrary node as the starting point for the tour.
2. Create an empty tour list to store the order of visited cities and maintain a set of visited cities.
3. Start from the initial city, add it to the tour list and mark it as visited in the set of visited cities.
4. Repeat the following steps until it visits all available cities
   1. Using the Euclidean distance, it calculates the distance between the current city and each unvisited city.
   2. Select the unvisited city with the shortest distance as the next node
   3. Adding it to the tour list and marking it as visited in the set of visited cities.
5. Check if there are any unvisited cities left. If all cities have been visited, complete the tour by adding the starting city to the tour list.

### Justification

It is widely used due to its simplicity and ease of implementation for a first heuristic algorithm to learn for solving the TSP. For small number of cities typically less than 50 like the sample, train and test files, it provides an accurate solution for the TSP (AlSalibi, Babaeian Jelodar and Venkat, 2012).

## Dijkstra Algorithm

### Description

Also known as the single source shortest path algorithm, the Dijkstra algorithm calculates the shortest distance between two nodes on either a directed or undirected graph (Ginting, Osmond and Aditsania, 2019). First, a graph is initialized to represent the vertex where each city is located and calculate the edge of each city (Binyamin Adeniyi Ajayi et al., 2022). It will analyze the weights of each vertex and select the one with the smallest weight. It will look for the shortest distance from the source vertex to the closest unselected vertex. This process is repeated until the algorithm has visited all the vertices available (Ginting, Osmond and Aditsania, 2019).

### Implementation

1. Create an adjacency matrix representing the distances between each pair of cities.
2. Initialize an empty tour list to store the order of visited cities and a set of visited cities.
3. Set the current city as the starting city by adding it to the tour list and mark as visited.
4. Repeat the following steps until it visited all cities
   1. Use Dijkstra algorithm to find the shortest distances from the current city to all other cities using a priority queue (min-heap).
      1. Check if the edge between the current city and each neighbouring cities in the graph (i.e., graph[u][v] != 0), and if the distance from the source city to the neighbouring city passing through the current city is shorter than the current recorded distance.
      2. If the condition is met, update the priority queue with an updated CityDistance object, which ensures that the shortest distance to this neigbour is maintained.
   2. Finds the index of the nearest unvisited city from the given distances array.
   3. Iterates through distances to locate the city with the shortest distance that has not been visited.
5. Once the priority queue is empty, there will be the tour array containing the shortest distances from the source city to all other cities.
6. Complete the tour by adding the starting city to the tour list.

### Justification

A study has shown that the travel route of Dijkstra reaches an accuracy of up to 100% in distance calculation and multiple tests were undertaken to compare its result with Google Maps (Syahputra et al., 2016). It is also used in IP routing and telephone networks to find the open shortest path (Binyamin Adeniyi Ajayi et al., 2022).

## Prim’s Algorithm (Minimum Spanning Tree)

### Description

MST is a tree that spans all the nodes of a connected weighted graph with the minimum or smallest possible total edge weight (Siahaan and Mhd. Furqan, 2018). It chooses the smallest weight on each edge and compares it with the other weights to the next node (Siahaan et al., 2018). To construct the MST, Prim's algorithm is the simplest to implement because it does not require complex data structures like Boruvka or Kruskal (Wójcik et al., 2018). Through the sequence of subtree propagation, the initial subtree is represented as a single vertex selected from a set of graph vertices. Then, for each subsequent iteration, until it includes all nodes in the created tree, it will use a greedy approach to find the nearest node not connected to any existing node using a Priority queue (Siahaan and Mhd. Furqan, 2018). To solve the TSP, the MST is only the starting point, and then a heuristic approach for path optimization needs to be taken like the 2-opt algorithm. It will eliminate two edges that cross themselves by reconnecting them through edges that do not cross each other (Siahaan et al., 2018).

### Implementation

1. Create an adjacency matrix representing the distances between each pair of cities.
2. Initialize three arrays, parent, key, and mstSet to store the MST structure and properties.
   1. The parent array keeps track of the parent node for each vertex in the MST.
   2. The key array stores the minimum edge weight to connect each vertex to the MST.
   3. mstSet: To track which vertices are included in the MST.
3. Repeat the following steps until it visited all cities
   1. Use Prim’s algorithm to construct an MST using a min-heap priority queue to efficiently determine the next minimum edge by finding the vertex with the minimum key value among the vertices not yet included in MST
4. Perform a preorder walk of the MST to create an initial TSP route.
   1. Starting from the root node, traverse the MST in a preorder manner, visiting children's nodes in the order they appear.
5. Apply shortcutting to the initial route to ensure that each city is visited only once by iterating through the initial route, skipping cities that have already been visited, and add the remaining unvisited cities to the optimized tour list.
6. Complete the tour by adding the starting city to the tour list.

### Justification

MST helps reduce the total cost or weight of a connected graph by selecting the subset of edges that form a tree and connecting all the vertices (Wójcik et al., 2018). Prim Algorithm can solve the TSP with a better time if there are fewer vertices and edges like in the train and test file provided (Siahaan and Mhd. Furqan, 2018).

# Test Result

## Train 1

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithms** | **Shortest Path** | **Distance** | **Time Taken (nanoseconds)** |
| Nearest Neighbor | 1-2-4-3-1 | 27.42622 | 290262 |
| Dijkstra’s Algorithm | 1-2-4-3-1 | 27.42622 | 2812077 |
| Prim’s Algorithm | 1-2-3-4-1 | 26.069053 | 1288448 |

## Train 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithms** | **Shortest Path** | **Distance** | **Time Taken (nanoseconds)** |
| Nearest Neighbor | 1-6-4-2-3-5-7-8-1 | 66.93059 | 314047 |
| Dijkstra’s Algorithm | 1-6-4-2-3-5-7-8-1 | 66.93059 | 2396247 |
| Prim’s Algorithm | 1-5-7-8-6-4-2-3-1 | 66.09035 | 1605268 |

## Train 3

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithms** | **Shortest Path** | **Distance** | **Time Taken (nanoseconds)** |
| Nearest Neighbor | 1-7-2-9-5-4-3-6-8-1 | 240.90737 | 342014 |
| Dijkstra’s Algorithm | 1-7-2-9-5-4-3-6-8-1 | 240.90737 | 3735551 |
| Prim’s Algorithm | 1-7-2-9-5-4-3-6-8-1 | 240.90737 | 1701509 |

## Test 1

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithms** | **Shortest Path** | **Distance** | **Time Taken (nanoseconds)** |
| Nearest Neighbor | 1-4-8-12-7-10-11-6-9-5-2-3-1 | 638.22892 | 413344 |
| Dijkstra’s Algorithm | 1-4-8-12-7-10-11-6-9-5-2-3-1 | 638.22892 | 5503853 |
| Prim’s Algorithm | 1-3-4-8-5-2-12-7-10-9-11-6-1 | 657.49367 | 1619705 |

## Test 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithms** | **Shortest Path** | **Distance** | **Time Taken (nanoseconds)** |
| Nearest Neighbor | 1-8-10-2-6-7-15-11-5-12-9-4-3-14-13-1 | 820.34467 | 465797 |
| Dijkstra’s Algorithm | 1-8-10-2-6-7-15-11-5-12-9-4-3-14-13-1 | 820.34467 | 5627468 |
| Prim’s Algorithm | 1-8-10-2-6-3-14-13-7-15-11-5-9-4-12-1 | 914.72497 | 1695932 |

## Test 3

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithms** | **Shortest Path** | **Distance** | **Time Taken (nanoseconds)** |
| Nearest Neighbor | 1-10-19-12-13-18-2-3-9-16-15-14-5-8-17-4-6-11-7-1 | 147334.96409 | 565579 |
| Dijkstra’s Algorithm | 1-10-19-12-13-18-2-3-9-16-15-14-5-8-17-4-6-11-7-1 | 147334.96409 | 5130053 |
| Prim’s Algorithm | 1-10-19-12-13-18-2-3-9-17-8-5-14-15-7-16-11-4-6-1 | 130254.33227 | 2032822 |

## Test 4

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithms** | **Shortest Path** | **Distance** | **Time Taken (nanoseconds)** |
| Nearest Neighbor | 1-10-9-17-35-30-18-12-23-8-5-19-32-31-39-22-29-33-16-27-26-36-38-28-11-20-13-6-14-34-15-25-37-4-2-24-3-7-21-1 | 199025.65960 | 1453541 |
| Dijkstra’s Algorithm | 1-10-9-17-35-30-18-12-23-8-5-19-32-31-39-22-29-33-16-27-26-36-38-28-11-20-13-6-14-34-15-25-37-4-2-24-3-7-21-1 | 199025.65960 | 9692151 |
| Prim’s Algorithm | 1-9-17-30-18-12-23-8-5-19-28-38-11-36-27-16-26-6-13-20-34-15-25-37-4-2-24-3-14-32-31-29-33-39-22-7-21-35-10-1 | 222942.70401 | 3500284 |

# Conclusion

In conclusion, the choice of the best algorithm implemented for solving the Traveling Salesman Problem (TSP) will depend.

If path pattern consistency is a paramount consideration, then "Dijkstra's Algorithm" emerges as the preferred choice, closely followed by "Nearest Neighbor." Both of these algorithms consistently produce the same path pattern in the provided test results.

However, if the primary objective is to obtain the shortest possible path and prioritize optimization of the travel route, then "Prim's Algorithm" stands out as the top choice. "Prim's Algorithm" consistently delivers optimal or near-optimal results in terms of the shortest path distance, ensuring that the solutions generated are among the most efficient routes, even if the path patterns may vary.

# Reference List

1. AlSalibi, B., Babaeian Jelodar, M. and Venkat, I. (2012) A Comparative Study between the Nearest Neighbor and Genetic Algorithms: A revisit to the Traveling Salesman Problem. International Journal of Computer Science and Electronics Engineering (IJCSEE) Volume 1, Issue 1 (2013) ISSN 2320–4028.
2. Binyamin Adeniyi Ajayi, Mohammed Abubakar Magaji, Musa, S., Rashidah Funke Olanrewaju and Abdullahi Audu Salihu (2022). A Comparative Analysis of Optimization Heuristics Algorithms as Optimal Solution for Travelling Salesman Problem. doi:https://doi.org/10.1109/ited56637.2022.10051627.
3. Emmanuel Oluwatobi Asani, Okeyinka, A.E. and Ayodele Ariyo Adebiyi (2020). A Construction Tour Technique For Solving The Travelling Salesman Problem Based On Convex Hull And Nearest Neighbour Heuristics. doi:https://doi.org/10.1109/icmcecs47690.2020.240847.
4. Ginting, H.N., Osmond, A.B. and Aditsania, A. (2019). Item Delivery Simulation Using Dijkstra Algorithm for Solving Traveling Salesman Problem. *Journal of Physics: Conference Series*, 1201(1), p.012068. doi:https://doi.org/10.1088/1742-6596/1201/1/012068.
5. Nuraiman, D., Ilahi, F., Dewi, Y. and Hamidi, E.A.Z. (2018). *A New Hybrid Method Based on Nearest Neighbor Algorithm and 2-Opt Algorithm for Traveling Salesman Problem*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICWT.2018.8527878.
6. Siahaan, A.P.U. a, Azhar, Z., Siahaan, M.D.L., Iqbal, M., Ramadhan, Z., Fitriani, W., Sitorus, Z., Mayasari, N., Sulistianingsih, I., Purwanto, D., Wijaya, R.F., Kurniawan, H., Hardinata, R.S., Muslim, M., Sari, R.D., Furqan, M., Ikhwan, A., Zuhanda, M.K., Lubis, A.H. and Kan, P.L.E. (2018). Comparative study of prim and genetic algorithms in minimum spanning tree and travelling salesman problem. *International Journal of Engineering & Technology*, [online] 7(4), pp.3654–3661. doi:https://doi.org/10.14419/ijet.v7i4.21749.
7. Siahaan, U. and Mhd. Furqan (2018). A Review of Prim and Genetic Algorithms in Finding and Determining Routes on Connected Weighted Graphs. *INA-Rxiv (OSF Preprints)*. doi:https://doi.org/10.31227/osf.io/e6rj7.
8. Syahputra, M.F.A., Devita, R.N., Siregar, S.A. and Kirana, K.C. (2016). Implementation of Traveling Salesman Problem (TSP) based on Dijkstra’s Algorithm in the Information System of Delivery Service. *JAVA Journal of Electrical and Electronics Engineering*, [online] 14(1). Available at: <http://telematics.its.ac.id/index.php/java/article/view/64/60> [Accessed 6 Dec. 2023].
9. Wójcik, W., Tito, J.E., Yacelga, M.E., Paredes, M.C., Utreras, A.J. and Ussatova, O. (2018). Solution of travelling salesman problem applied to Wireless Sensor Networks (WSN) through the MST and B&B methods. *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2018*. doi:https://doi.org/10.1117/12.2501579.