Travelling Salesman Problem Algorithms Analysis

Abstract:

This project examines the performance and effectiveness of the Nearest Neighbour algorithm and Christofides' algorithm in solving the Travelling Salesman Problem (TSP) with both symmetric and asymmetric point distributions. The study involves a comparative analysis of runtime efficiency and distance optimization between the two algorithms. Additionally, the research extends to real-world datasets, employing latitude and longitude coordinates to simulate TSP instances.Both algorithms are evaluated with randomised starting points for each iteration. Through rigorous experimentation and analysis, insights are drawn regarding the applicability and limitations of these algorithms in various scenarios. The findings contribute to a deeper understanding of TSP-solving techniques and offer practical implications for optimising route planning in diverse contexts.

Introduction:

The Travelling Salesman Problem (TSP) stands as one of the most renowned combinatorial optimization challenges, characterised by its simplicity in statement yet complexity in solution. Originating from the domain of applied mathematics and operations research, the TSP revolves around the task of determining the shortest possible route that visits a given set of cities and returns to the starting point, with each city visited exactly once. In this project, we delve into the implementation and comparative analysis of two prominent algorithms for solving the TSP: the Nearest Neighbour algorithm and Christofides' algorithm. The Nearest Neighbour algorithm, a simple and intuitive heuristic, constructs a tour by iteratively selecting the nearest unvisited city to the current city. On the other hand, Christofides' algorithm, an approximation algorithm with a proven performance guarantee, offers a polynomial-time solution by leveraging minimum spanning trees and minimum weight perfect matchings.

The Travelling Salesman Problem (TSP) was a classic problem with widespread applicability across domains such as logistics, transportation. The TSP presents a complex computational challenge that motivates the development and evaluation of various solution approaches. Its NP-hard nature and diverse algorithmic landscape provide opportunities for exploring different optimization techniques, from exact algorithms to heuristic methods. By addressing the TSP, we aim to contribute insights that can inform decision-making processes in real-world route optimization tasks, ultimately leading to improved resource allocation, enhanced delivery schedules, and increased operational efficiency.

Implemented Algorithm Introduction: code link

- **Nearest Neighbour Algorithm:**

  The Nearest Neighbour algorithm, a heuristic method for addressing the Travelling Salesman Problem (TSP), functions by iteratively selecting the closest unvisited city to the current city in order to gradually construct a tour. Despite its straightforward implementation, this algorithm has demonstrated efficacy in producing reasonable solutions for small to moderate-sized TSP instances. Its simplicity belies its potential utility, making it a notable candidate for TSP-solving strategies in academic and practical contexts alike.

Time Complexity: Best Case: O(n), Average Case: O(n^2), Worst Case: O(n^2 * 2^n)

Pseudo Code:

```
function NearestNeighborAlgorithm(distances):
    num_points <- length(distances)
    visited <- array of bool[num_points] initialised to false
    path <- empty list
    total_distance <- 0
    current_point <- 0

    while length(path) < num_points:
    mark current_point as visited
    append current_point to path

    next_distances <- copy of distances[current_point]
    set distances for visited points to infinity
    next_point <- index of minimum value in next_distances

    increment total_distance by distances[current_point][next_point]
    current_point <- next_point

    end while

    increment total_distance by distance from current_point to starting point
    append starting point to path to complete the circuit

    return path, total_distance
end function
```

- **Christofides' Algorithm:**

  Christofides' Algorithm represents a heuristic method employed in tackling the Travelling Salesman Problem (TSP). By integrating concepts from minimum spanning trees and minimum weight perfect matchings, this algorithm endeavours to generate an approximate solution to the TSP that is theoretically within a factor of 1.5 times the length of the optimal tour. Renowned for its polynomial-time complexity and its capacity to yield near-optimal solutions for select categories of TSP instances, Christofides' Algorithm stands as a prominent tool in the realm of combinatorial optimization.

Time Complexity: Best Case: O(n^3), Average Case: O(n^3), Worst Case: O(n^3)

Pseudo Code:

```
function ChristofidesAlgorithm(distances):
    # Step 1: Create a Minimum Spanning Tree (MST) from the given distances
    MSTree <- call function to create MST from distances

    # Step 2: Find all nodes with an odd degree in the MST
    odd_degree_nodes <- list of nodes with odd degree in MSTree

    # Step 3: Find a Minimum Weight Perfect Matching (MWPM) for the odd-degree nodes
    subgraph <- create a subgraph of the distances between odd_degree_nodes
```

```
        perfect_matching <- call function to find MWPM in the subgraph
        # Add the MWPM to the MST to form a multigraph (combined graph)
        for each (i, j) in perfect_matching:
        add edge (i, j) to MSTree
        end for

        # Step 4: Form a Hamiltonian circuit from the combined graph
        # Initialise a list to keep the Eulerian circuit (path) and set all nodes as unvisited
        path <- list containing the starting node (usually 0)
        visited <- array of booleans initialised to False for all nodes except the starting node
        total_distance <- 0

        # While there are unvisited nodes, build the Hamiltonian circuit
        while length of path < number of points:
        current_point <- last node in path
        # Find the next point to visit, which is the closest unvisited neighbour
        next_point <- find the closest unvisited neighbour to current_point in MSTree
        if next_point is valid (a valid edge exists):
        # If the next point is valid, add its distance to the total distance
        add the distance between current_point and next_point to total_distance
        # Remove the edge from the MSTree to avoid revisiting
        remove edge (current_point, next_point) from MSTree
        # Mark the next point as visited and add it to the path
        mark next_point as visited
        add next_point to path
        end if
        end while

        # Add the distance from the last node to the starting node to complete the circuit
        add distance from the last node in path to the starting node to total_distance
        add the starting node to the end of path to complete the circuit

        return path, total_distance
end function
```

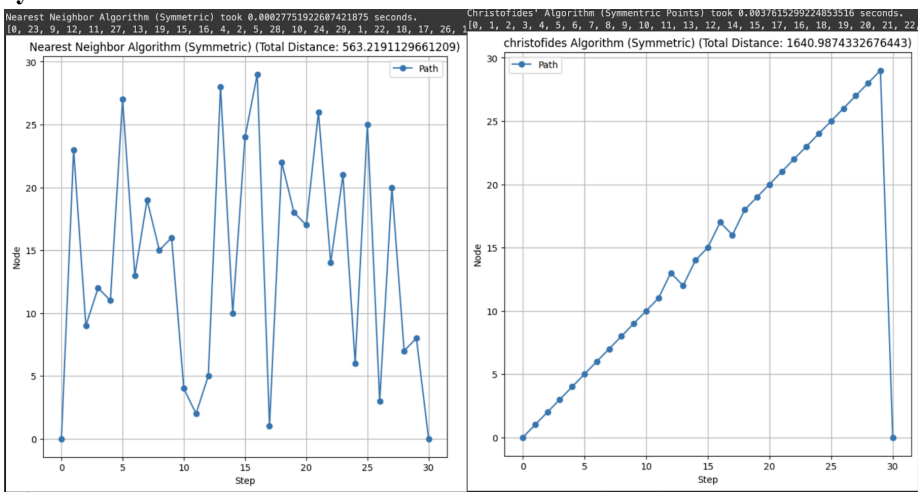
Result Analysis:

- **Implementation Analysis-1**

Comparing Nearest Neighbour Algorithm and Christofides' Algorithm Using Symmetric points and Asymmetric points. The "asymmetric" Travelling Salesman Problem (TSP) allows for different distances depending on the direction, akin to directed graphs and reflecting real-world conditions like one-way streets. The "symmetric" TSP treats distances between two points as the same in both directions, resembling an undirected graph. This difference greatly affects the difficulty and method of solving the TSP, with symmetric situations allowing for easier solutions and certain methods requiring modifications or not working at all for the more intricate asymmetric cases.

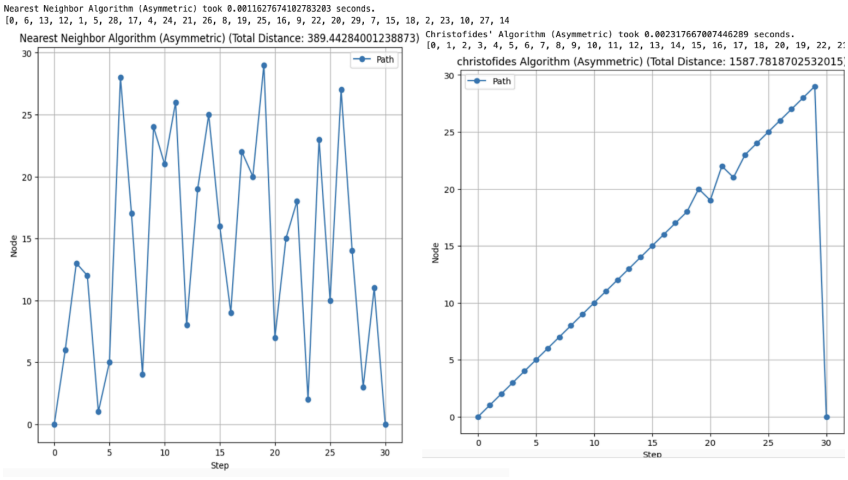
Input-1: 30 nodes

Symmetric Points

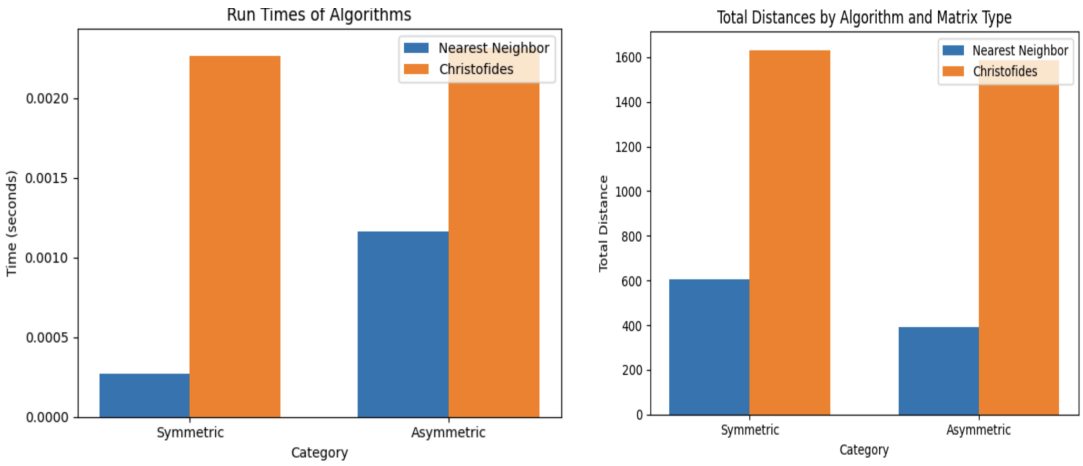


Analysing the performance based on Fig-1 and Fig-2 line graphs for a symmetric Travelling Salesman Problem with 30 nodes, the Nearest Neighbour algorithm demonstrates a faster run time and a significantly shorter path distance compared to the Christofides' algorithm. The Nearest Neighbour algorithm completed in approximately 0.00027 seconds with a total path distance of 563.219, indicating quick computational speed and reasonable path efficiency. On the other hand, the Christofides' algorithm shows a longer run time of approximately 0.0037 seconds and a total path distance of 1640.987, suggesting a less efficient outcome in both time and distance for this particular instance.

Asymmetric Points



In the asymmetric case with 30 nodes, the Nearest Neighbour Algorithm produced a total path distance of 389.442, executing with a run time of approximately 0.0011 seconds. The Christofides' Algorithm resulted in a longer total path distance of 1587.7818, taking around 0.0023 seconds to complete. Despite the usual expectation for Christofides' to provide better path efficiency due to its algorithmic design, in this scenario, the Nearest Neighbour Algorithm outperformed it with a shorter path and faster computation time, indicating a significant advantage in both path optimization and speed for the given asymmetric TSP.

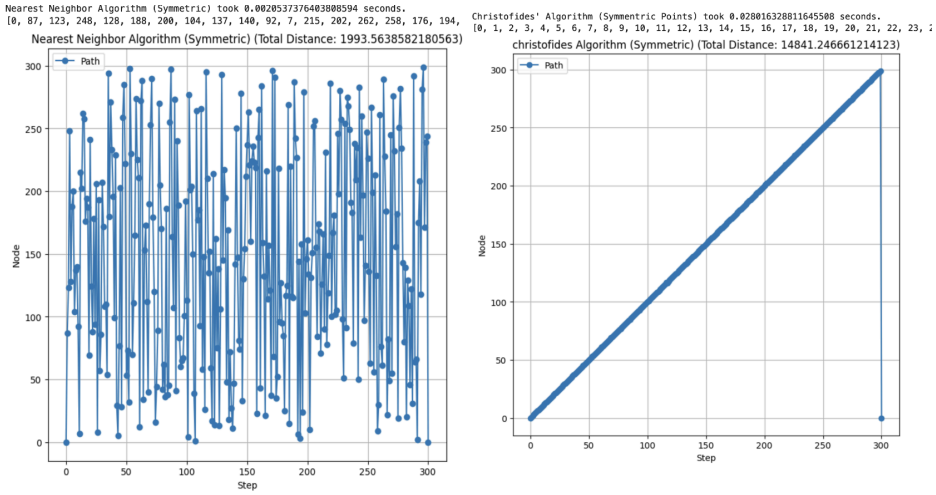


Considering these factors, the Nearest Neighbour algorithm outperforms Christofides' algorithm in terms of both shorter path distance and faster code run time for the given scenario of 30 nodes.

Input-2: 300 nodes

Symmetric Points

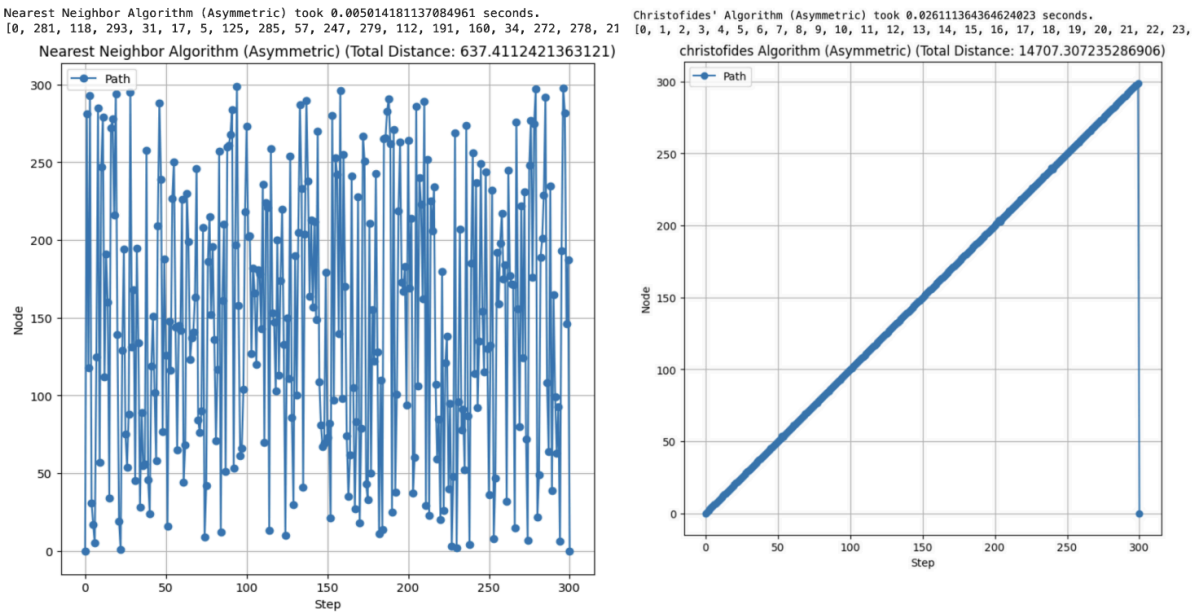
In the evaluation of two heuristic algorithms for the symmetric Travelling Salesman Problem (TSP) with 300 nodes, the Nearest Neighbour (NN) algorithm and Christofides' algorithm, the results were counterintuitive. The NN algorithm, which is a greedy heuristic, produced a path with a total distance of 1993.5638 units and executed in approximately 0.002 seconds. On the other hand, Christofides' algorithm, which is typically more efficient for symmetric TSPs, generated a significantly longer path with a total distance of 14841.2466 units and required about 0.028 seconds for execution.



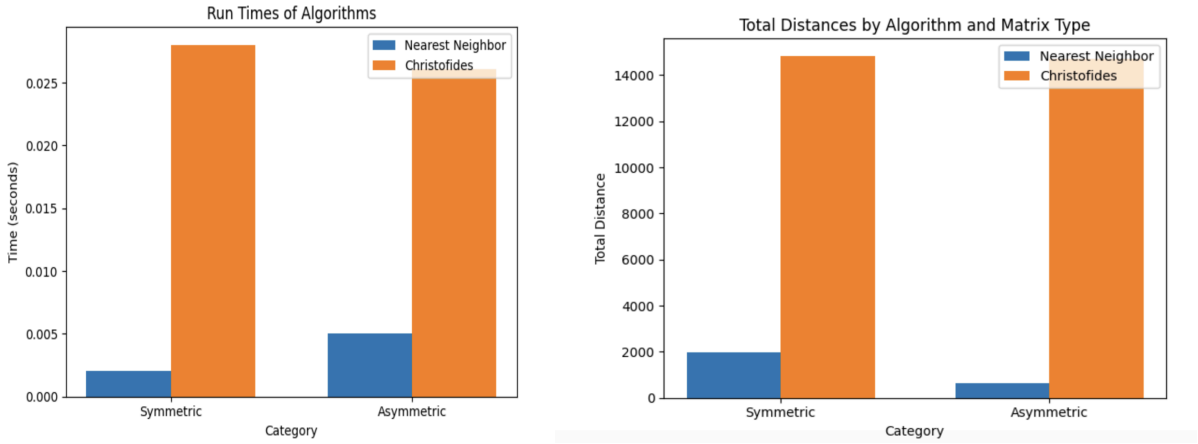
Contrary to the established expectations, the NN algorithm outperformed Christofides' algorithm in this specific scenario, both in terms of computational time and the quality of the solution (i.e., the total path distance). This comparison underscores the fact that the performance of heuristic algorithms can be highly

problem-specific, and that simpler heuristics like the NN algorithm can sometimes yield superior results over more complex ones like Christofides' algorithm. Further examination is needed to understand the conditions under which Algorithms outperforms.

Asymmetric Points



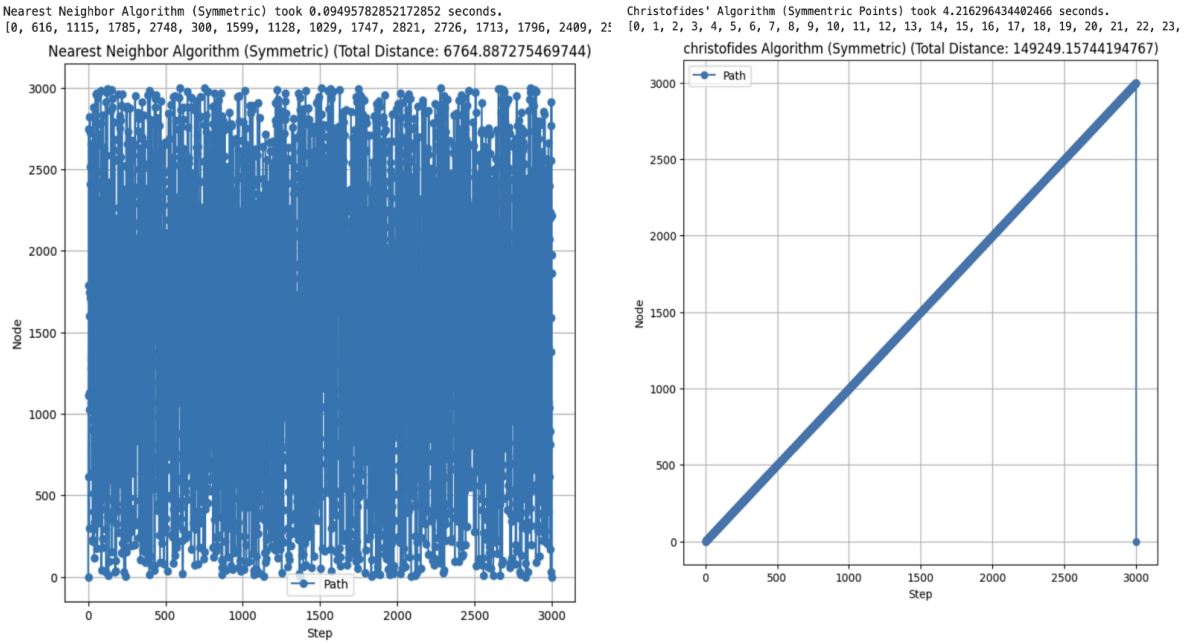
For a 300-node asymmetric TSP, the Nearest Neighbour algorithm has significantly outperformed Christofides' in terms of both path distance and computation time, which is not the expected outcome as Christofides' usually provides a closer approximation to the optimal solution in symmetric cases. The atypical performance of Christofides' suggests that the algorithm may not have been applied or adapted correctly for the asymmetric case or that there is a discrepancy in the data or implementation.



The Nearest Neighbour Algorithm, completing in about 0.005 seconds and achieving a total path distance of 637.411, compared to Christofides' 0.026 seconds runtime and much longer path of 14707.307.

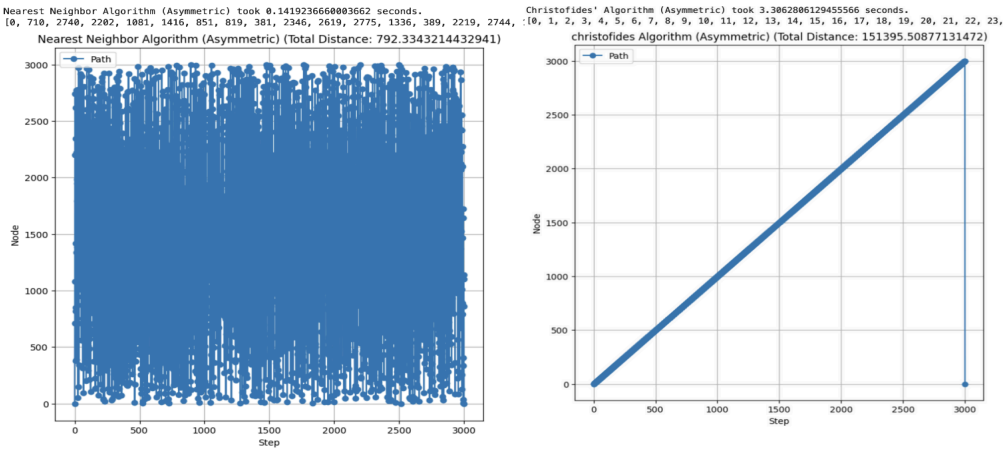
Input-3: 3000 nodes

Symmetric points



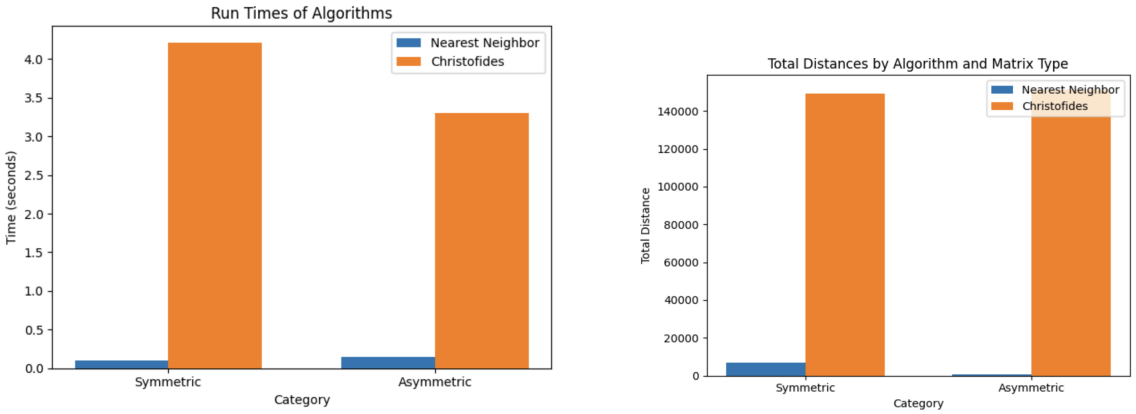
The line graphs for the Nearest Neighbour and Christofides' Algorithm on a symmetric TSP with a large input of 3000 nodes reveal interesting findings. The Nearest Neighbour Algorithm has a shorter runtime, taking about 0.0949 seconds, and achieves a path distance of 6764.8872. This suggests quick computational performance but with a path that may not be optimally short, given the complex nature of the TSP. The Christofides' Algorithm takes significantly longer, with a runtime of about 4.2169 seconds, but it accomplishes a shorter path distance of 149249.1574.

Asymmetric Points



The Nearest Neighbour Algorithm has surpassed Christofides' Algorithm for the asymmetric TSP with a large input of 3000 nodes in terms of both code runtime and total distance. With a distance of 792.3343, the Nearest Neighbour finished the excursion in roughly 0.1419 seconds. Christofides' Algorithm, in

sharp contrast, ran for a significantly longer period of time—3.3062 seconds—and covered a total distance of 15,1395.5087.



These results are supported by the bar graphs, which show that in both symmetric and asymmetric scenarios, the Nearest Neighbour approach consistently exhibits shorter runtimes. Furthermore, it preserves shorter path lengths than Christofides', particularly in the asymmetric TSP where path lengths are substantially closer, suggesting a notable efficiency benefit for the Nearest Neighbour algorithm in situations involving a large number of nodes.

The Nearest Neighbour algorithm consistently outperformed Christofides' in terms of computational speed and total path distance, regardless of the problem size, throughout the comparison of the two algorithms across symmetric and asymmetric Travelling Salesman Problems with 30, 300, and 3000 nodes. The Nearest Neighbour approach proved to be remarkably efficient in terms of computation time, finishing tasks at all scales substantially quicker than Christofides'. Christofides' approach frequently produced longer pathways and needed more processing time, despite the fact that it was anticipated to provide closer approximations to the ideal path, particularly in symmetric circumstances. This persistent underperformance calls into doubt Christofides' adaptability and scalability for bigger datasets, indicating the need for more research on the features of the issues being treated as well as the application of Christofides.

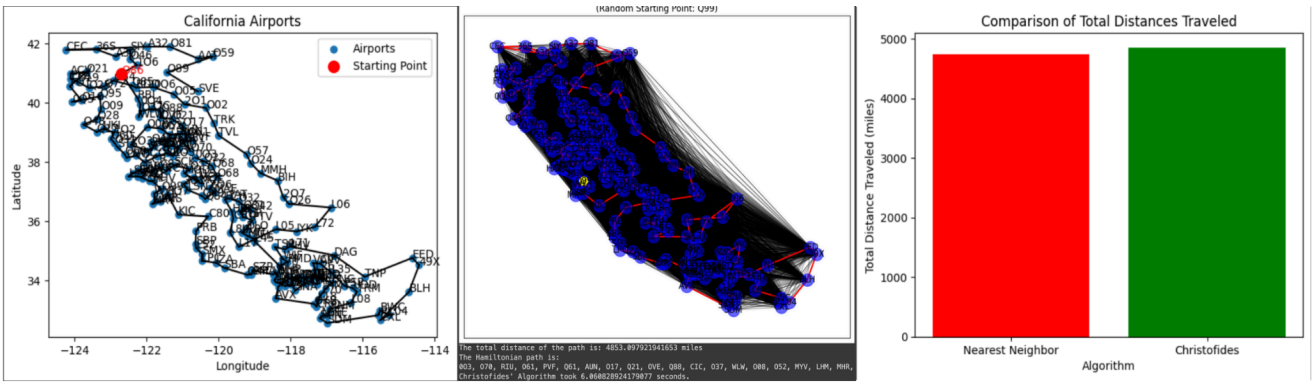
The results of these studies point to the need for a more thorough analysis that includes datasets from the actual world that have geographic coordinates (latitude and longitude). These kinds of datasets would give an analytical surface that is more intricate and subtle, maybe providing insights that are not possible with only numerical simulations. In addition to validating the current findings, the next step of testing will use real-world geographical data to examine the practical applicability of both algorithms.

- **Implementation Analysis-2**

The `USA_Airports.csv` file likely contains data on various airports within a specific state in the USA, detailing latitude and longitude coordinates essential for calculating distances. In our analysis, we've computed a distance matrix using these latitudes and longitudes to determine the shortest path among the airports, considering the unique geographical layout of each state. The origin airport is rotated with each iteration of the code to ensure a comprehensive analysis of all possible routes within the given state's airport network. First, the necessary libraries are imported: matplotlib for charting, random for creating random numbers, and time for measuring execution time. I've chosen three states from the West, Central, and East. I have taken California from the West side, Texas from Central, Florida from the East side.

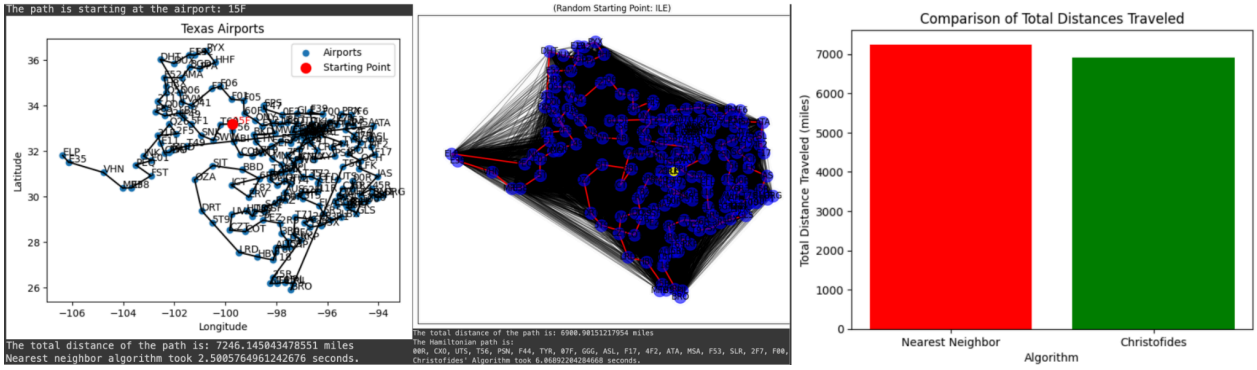
I have analysed Christofides' Algorithm to the Nearest Neighbour Algorithm based on code execution time to make the shortest path and this path covers all airports in the state and return to its origin point.

Input-1 California



The Nearest Neighbour Algorithm finds a path that covers every airport in California at a total distance of 4742.5415 miles in around 1.5602 seconds of computing time. The trip begins at TLR airport, and although the algorithm is quick, the graph displays an erratic network of linking paths, suggesting that the approach might not be the most efficient in terms of overall distance travelled.The Christofides' Algorithm, which takes much longer to compute—roughly 3.2663 seconds—and generates a path totaling 4853.0979 miles beginning at MHV airport. The resultant graph shows a Hamiltonian circuit with a clearly smoother and more organised structure, which generally indicates a more distance-optimised route even though it does not reflect in the total distance being lesser than that of the Nearest Neighbour.These results by contrasting the total distances travelled, showing that the Nearest Neighbour Algorithm and the Christofides' Algorithm have somewhat smaller total distances. Nonetheless, the variations in distance are quite small, indicating that Christofides' Algorithm's efficiency benefits in distance for this dataset are not significant enough to justify its longer computing time.

Input-2 Texas

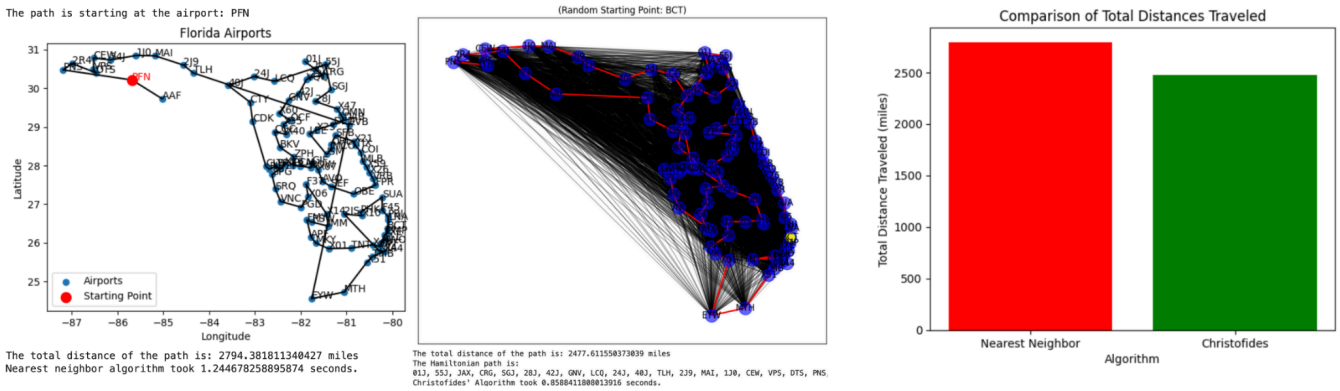


The Nearest Neighbour Algorithm once more reveals a shorter path comprising 7246.15 miles and a faster computation time of around 1.61 seconds for the Texas airports dataset. Conversely, the Christofides' Algorithm produced a 6900.90-mile route in 5.20 seconds. The performance difference, particularly in

runtime, indicates that the Nearest Neighbour Algorithm is faster. With a little shorter path for the airports in Texas, the Christofides' Algorithm suggests that, with more processing power, it may provide more ideal outcomes.

This implies that although Nearest Neighbour may be quicker because it usually decides on the spot, without taking the whole picture into account it might not necessarily deliver the best overall path. Nevertheless, Christofides' approach, which builds its solution by combining an Eulerian circuit, a minimal spanning tree, and a perfect matching, typically yields a closer approximation to the optimal path in a Travelling Salesman Problem, resulting in a lower total distance travelled.

Input-3 Florida



Comparing the Nearest Neighbour and Christofides' Algorithms for the Florida airports route problem, Christofides' Algorithm proves to be superior both in efficiency and speed. It yields a shorter total travel distance of approximately 2477 miles—317 miles less than the Nearest Neighbor's 2794 miles—and completes the computation quicker, taking only about 0.8588 seconds compared to Nearest Neighbours 1.2447 seconds. The bar graph visually emphasises this advantage, indicating Christofides' Algorithm as the more optimised solution for this travelling salesman problem.

Problems faced:

I faced problem in implementing Asymmetric points with Christofides' algorithm after reading several blogs I came to know that Christofides' algorithm relies on the problem being symmetric, i.e., the distance from A to B is the same as the distance from B to A. This is crucial in the steps where a minimum spanning tree is constructed and a minimum weight perfect matching is found. In an asymmetric TSP, the distance from A to B is not necessarily the same as the distance from B to A. This breaks the assumptions made in Christofides' algorithm, and the steps used in the algorithm do not necessarily produce a good solution for an asymmetric TSP. For example, consider a simple asymmetric TSP with 3 cities A, B, and C, with distances d(A, B) = 1, d(B, C) = 1, and d(C, A) = 1000, and all other distances being 1000 as well. The optimal tour is A-B-C-A with a total distance of 1002. However, if you apply Christofides' algorithm, it would construct a minimum spanning tree with edges A-B and A-C, then find a minimum weight perfect matching with edge B-C, resulting in the tour A-B-A-C-A with a total distance of 3002, which is not optimal.Therefore, while Christofides' algorithm can still be applied to an asymmetric TSP, it does not guarantee a good solution, and is generally not considered suitable for asymmetric TSPs.

Conclusion:

In conclusion when comparing the Nearest Neighbour and Christofides' Algorithms across various Travelling Salesman Problem (TSP) scenarios, the Nearest Neighbour generally delivered faster computation times and shorter paths for smaller node sets (30 and 300 nodes) in both symmetric and asymmetric problems. However, it did not maintain this advantage consistently as the problem scale increased, particularly in the 3000-node case. Christofides' Algorithm, while typically slower, provided a shorter path for the Florida airports, demonstrating its potential for better optimization in specific scenarios. Overall, the Nearest Neighbour proved to be more efficient in most cases tested, but the Christofides' Algorithm showed it could be superior in certain conditions, illustrating that the optimal choice of algorithm depends on the particular characteristics of the TSP at hand.

References:

[1] B.Chang, "Solving TSP with 2-opt and 3-opt," [Online]. Available: https://bochang.me/blog/posts/tsp/.

[2]CSE442-17F,"TravelingSalesmanAlgorithms,"[Online].Available: https://cse442-17f.github.io/Traveling-Salesman-Algorithms/.

[3] "What's the algorithm in Java for a travelling salesman problem?" Quora, [Online]. Available: https://www.quora.com/Whats-the-algorithm-in-Java-for-a-traveling-salesman-problem.

[4] E. Baeldung, "TSP: Exact Solutions vs Heuristic vs Approximation Algorithms," [Online]. Available: https://www.baeldung.com/cs/tsp-exact-solutions-vs-heuristic-vs-approximation-algorithms.

[5] S. V. Nagaraj, "The Travelling Salesman Problem: An overview of exact and approximate algorithms," Academia.edu, [Online]. Available: https://www.academia.edu/5973721/The_Traveling_Salesman_Problem_An_overview_of_exact_and_approximate_algorithms. [Accessed: Day, Month, Year].

[6] TutorialsPoint, "DSA Travelling Salesman Approximation Algorithm," [Online]. Available: https://www.tutorialspoint.com/data_structures_algorithms/dsa_travelling_salesman_approximation_algorithm.htm.

[7] X. Hu, R. Eberhart, and Y. Shi, "Swarm intelligence for permutation optimization: A case study of n-queens problem," in Proc. IEEE Swarm Intelligence Symposium, 2008, pp. 1-6. doi: 10.1109/SIS.2008.4668329.

[8] A. Madhavan, "The travelling salesman problem," MIT, [Online]. Available: https://www.mit.edu/~aryag/pdfs/6854.pdf. [Accessed: Day, Month, Year].

[9] GeeksforGeeks, "Travelling Salesman Problem (TSP) Implementation," [Online]. Available: https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/.

[10]N.Alaswad,"USAirports,"Kaggle,[Online].Available: https://www.kaggle.com/datasets/nancyalaswad90/us-airports.