Comparative Analysis of Christofides, Nearest Neighbor, and Lin-Kernighan Algorithms for Solving the Traveling Salesman Problem

Varun Chikkala

1. Abstract

In this study, we have compared and evaluated three algorithms (Christofides, Nearest Neighbor, and Lin-Kernighan) which solve the Traveling Salesman Problem (TSP). To assess the efficiency and quality of solutions from each algorithm, we have evaluated both randomly generated data and real-world data using the Kaggle World Cities dataset. Findings from our code indicate that the Christofides algorithm offers a good balance between speed and accuracy, producing approximately optimal solutions with minimal computational resources. On the other hand, the Nearest Neighbor algorithm is quick but generates suboptimal routes, highlighting the need for a trade-off between speed and accuracy. The Lin-Kernighan heuristic is excellent at reducing route distances but requires higher computational resources, making it best suited for situations where solution quality is paramount. This study provides valuable insights into the situational advantages and disadvantages of these algorithms, enabling their strategic application in solving TSP challenges of varying scales and complexities.

2. Methodology

2.1 Christofides Algorithm

2.1.1 Algorithm Description

The Christofides algorithm offers an effective approach to obtaining an estimated solution for the TSP by first determining the minimum spanning tree (MST) of the city graph, followed by identifying a perfect matching with minimum weight among the vertices with odd degrees in the MST. Subsequently, a multigraph is generated by incorporating both the edges of the MST and the matching. Lastly, an Euler tour is developed and transformed into a Hamiltonian cycle by eliminating duplicate vertices, ultimately yielding an approximate solution.

2.1.2 Pseudocode

Pseudocode for the Christofides Algorithm

```
FUNCTION calculate_distance(city1, city2):
    lat1, lng1 = get_latitude_and_longitude(city1)
    lat2, lng2 = get_latitude_and_longitude(city2)
    distance = square_root((lat2 - lat1)^2 + (lng2 - lng1)^2)
    RETURN distance

FUNCTION create_graph(data):
    G = empty Graph
    FOR EACH city1 IN data:
        FOR EACH city2 IN data:
            IF city1 is not equal to city2 THEN
                distance = calculate_distance(city1, city2)
```

```
        add_edge(G, city1, city2, distance)
    RETURN G

FUNCTION eulerian_tour(G):
    tour = empty list
    FOR EACH edge IN eulerian_circuit(G):
        add edge to tour
    RETURN tour

FUNCTION eulerian_to_hamiltonian_tour(eulerian_tour):
    hamiltonian_tour = empty list
    FOR EACH edge IN eulerian_tour:
        IF edge is not in hamiltonian_tour THEN
            add edge to hamiltonian_tour
    add first city of hamiltonian_tour to the end
    RETURN hamiltonian_tour

FUNCTION christofides_tsp(data):
    G = create_graph(data)
    T = minimum_spanning_tree(G)
    odd_degree_nodes = empty list
    FOR EACH node IN nodes of T:
        IF degree_of(node) modulo 2 is not equal to 0 THEN
            append node to odd_degree_nodes
    matching = max_weight_matching(subgraph of G containing odd_degree_nodes, maxcardinality=True)
    M = copy of T
    add edges from matching to M
    euler_tour = eulerian_tour(M)
    hamilton_tour = eulerian_to_hamiltonian_tour(euler_tour)
    RETURN hamilton_tour
```

The pseudocode outlines Christofides' algorithm for the Traveling Salesperson Problem (TSP), achieving a near-optimal solution with a **time complexity** dominated by the maximum weight matching step at **$O(n^3)$** and a **space complexity** of **$O(n^2)$** due to graph storage and operations.

2.2 Nearest Neighbor Algorithm

2.2.1 Algorithm Description

The Nearest Neighbor algorithm is a greedy based solution that builds a tour by starting from an arbitrary node and repeatedly selecting the nearest unvisited neighbor as the next node. Although simple and intuitive, the algorithm does not guarantee an optimal solution, as the tour quality heavily depends on the starting node and graph structure. It serves as an efficient, straightforward approach for generating a feasible solution quickly but may result in significantly longer paths in certain graph configurations, highlighting a trade-off between computational efficiency and solution optimality.

2.2.2 Pseudocode

```
FUNCTION nearest_neighbor(G):
    tour = [ 0 ]
    WHILE length of tour is less than n:
```

```
    i = last node of tour
    min_length = minimum length among edges from i to neighbors of i not in tour
    nearest_neighbors = []
    FOR EACH neighbor j of i:
        IF j is not in tour and length of edge from i to j is equal to min_length:
            append j to nearest_neighbors
    append first node from nearest_neighbors to tour
RETURN tour
```

The `nearest_neighbor` function for the Traveling Salesperson Problem (TSP) iteratively constructs a tour starting from an arbitrary node by repeatedly adding the nearest unvisited neighbor until all nodes are included, with a **time complexity** of **O(n^2)** due to examining up to `n-1` distances for each of the `n` steps, and a **space complexity** of **O(n)** to store the tour.

2.3 Lin-Kernighan Algorithm

2.3.1 Algorithm Description

The Lin-Kernighan heuristic is an optimization algorithm for solving the Traveling Salesman Problem (TSP), which seeks to minimize the total distance of a round-trip tour through a set of cities, visiting each once. Starting from an initial tour and a distance matrix, it iteratively enhances the tour by reversing city subsequences if such reversals yield a shorter route. This search continues until no further improvement is possible, suggesting a solution close to the optimal. The algorithm improves upon the initial solution by systematically exploring and adopting local changes that reduce the overall travel distance, efficiently navigating towards an optimal or near-optimal tour.

2.3.2 Pseudocode

```
FUNCTION lin_kernighan(tour, distances):
    best_tour = tour
    best_cost = calculate_total_distance(tour, distances)

    CONTINUE_LOOP = TRUE
    WHILE CONTINUE_LOOP:
        best_swap = None
        FOR i = 0 TO length(tour) - 1:
            FOR j = i + 1 TO length(tour):
                new_tour = copy(tour)
                reverse_subsequence(new_tour, i, j)
                new_cost = calculate_total_distance(new_tour, distances)
                IF new_cost < best_cost THEN
                    best_swap = (i, j)
                    best_tour = new_tour
                    best_cost = new_cost

        IF best_swap is None THEN
            CONTINUE_LOOP = FALSE
        ELSE:
            tour = best_tour

    RETURN best_tour
```

```
FUNCTION calculate_total_distance(tour, distances):
    total_distance = 0
    FOR i = 0 TO length(tour) - 1:
        total_distance = total_distance + distances[(tour[i], tour[i+1])]
    RETURN total_distance

FUNCTION reverse_subsequence(tour, start_index, end_index):
    WHILE start_index < end_index:
        SWAP(tour[start_index], tour[end_index])
        INCREMENT start_index
        DECREMENT end_index
```

The `lin_kernighan` function is a sophisticated heuristic for refining a given tour in the Traveling Salesperson Problem (TSP), leveraging an iterative search for edge swaps that can reduce the tour's total distance. Despite its potential for high computational cost, due to its **$O(n^3)$ time complexity** per iteration over a tour of length `n` and **space complexity of $O(n^2)$**, the Lin-Kernighan heuristic is celebrated for its ability to significantly enhance solution quality by escaping local optima, making it a preferred choice for TSP instances where the precision of the solution outweighs computational constraints.

3. Experimental Analysis

3.1 Experimental Procedure

Libraries used:

1. **Python_tsp**: Python package for solving Travelling Salesman Problem (TSP) using various algorithms.
2. **NumPy**: Library for numerical computing, providing support for large, multi-dimensional arrays.
3. **Pandas**: Library for data manipulation and analysis.
4. **Plotly**: Library for creating interactive and publication-quality visualizations, including charts, graphs, and dashboards, with support for various plotting types and customization options.
5. **Netwrokx**: Library for creating, manipulating, and analyzing complex networks or graphs, offering tools for studying the structure, dynamics, and functions of networks in various domains such as social networks, biological networks, and transportation networks.

3.1.1 Real-World Data Experiment

The dataset is taken from kaggle which was taken from simplemaps which has an upto date data taken from authoritative sources like NGIA, US Geological Survey, US Census Bureau, and NASA.

In exploring the efficacy of the Christofides, Nearest Neighbor, and Lin-Kernighan algorithms, I targeted an Australian cities dataset from the Kaggle World Cities database for my real-world data experiment. My initial step involved preprocessing this dataset to sort cities by population into three distinct clusters: high, medium, and low. This categorization was pivotal for a nuanced analysis across different urban densities. I harnessed the python_tsp library to implement the algorithms, with a particular focus on Lin-Kernighan's heuristic, and meticulously constructed distance matrices using great-circle distances for an accurate geographical representation. I hosted my experiment's code on a Jupyter notebook via Google Colab to ensure an interactive and reproducible research setup.

To visualize the outcome, I used Plotly for dynamic mapping of city coordinates and the computed tour paths, providing a clear comparison of each algorithm's route efficiency. Execution times were carefully recorded with Python's time library, enabling a comprehensive evaluation of both the computational efficiency and the solution's quality.

3.1.2 Random Data Experiment

For the random data experiment, I generated random datasets with varying node sizes to test the algorithms' scalability under controlled conditions. Nodes were placed randomly within a normalized space, leading to the construction of a complete graph. This setup allowed me to execute the Christofides and Lin-Kernighan algorithms, alongside capturing their execution times for performance analysis.

Additionally, I applied the Nearest Neighbor algorithm, initiating from a randomly selected node and progressing by selecting the closest unvisited node iteratively. This method provided a basis for assessing the greedy heuristic against its more elaborate counterparts in terms of quick solution provision and path efficiency.

The entire experimental process was designed with a high emphasis on consistency and reproducibility. By setting specific seed values for random number generation, I ensured that each iteration of the experiment could be replicated precisely. Moreover, I prioritized clarity and modularity in my coding approach to facilitate easy understanding and future project extensions. This structured experimentation not only helped me in analyzing the algorithms' efficiency and solution quality both numerically and visually but also in understanding their structural nuances through the tours they produced.

We displayed the complete graph and resulting paths using Plotly's graphical capabilities. This enabled us to not only analyze algorithmic efficiency and solution quality numerically but also visually assess the tour structures produced by each algorithm.

3.2 Experiments Undertaken

3.2.1 Random Data Experiment

The performance of the algorithms was evaluated across varying node sizes using random data to simulate different TSP scenarios. This allowed for a comprehensive analysis of each algorithm's efficiency and solution quality.

Tabular Data:

Nodes	Christofides(CH)	Lin kernighan(LK)	Nearest Neighbor(NN)	Distance Wise Best Path	Time Wise Best Path(t in s)
10	d =4.707603077994739 t=0.013050317764282227	d =3.438885012668645 t = 0.002989530563354492	d = 3.594273913037621 t = 8.153915405273438e-05	LK	NN
25	d = 8.039280046287534 t = 0.00899052619934082	d = 4.786996897614424 t = 0.12700700759887695	d=5.976671497058518 t=0.0009989738464355469	LK	NN
30	d=8.69295558578098 t=0.010630369186401367	d=5.128905655514509 t=0.20109963417053223	d=6.295825880106995 t=0.00032830238342285156	LK	NN
50	d=14.403924285062994 t=0.06866598129272461	d=6.018781180196016 t=1.4894821643829346	d=6.266256008408186 t=0.0009970664978027344	LK	NN
100	d=23.06307333026971 t=0.2669804096221924	d=8.194897322222468 t=16.494138956069946	d=10.834714689417956 t=0.000997304916381836	LK	NN
200	d=37.90863333747361 t=1.947399377822876	d=11.114886125186208 t=268.13975954055786	d=13.734306326193634 t=0.008122682571411133	LK	NN
300	d=56.828425742527955 t=24.29776167869568	d=13.419865842606209 t=1581.9635274410248	d=15.352491700181842 t=0.017824172973632812	LK	NN
500	d=99.54736474939604 t=29.880012273788452	d=17.296421570667345 t=13255.052483558655	d=15.352491700181842 t=0.07654309272766113	LK	NN

Analysis:

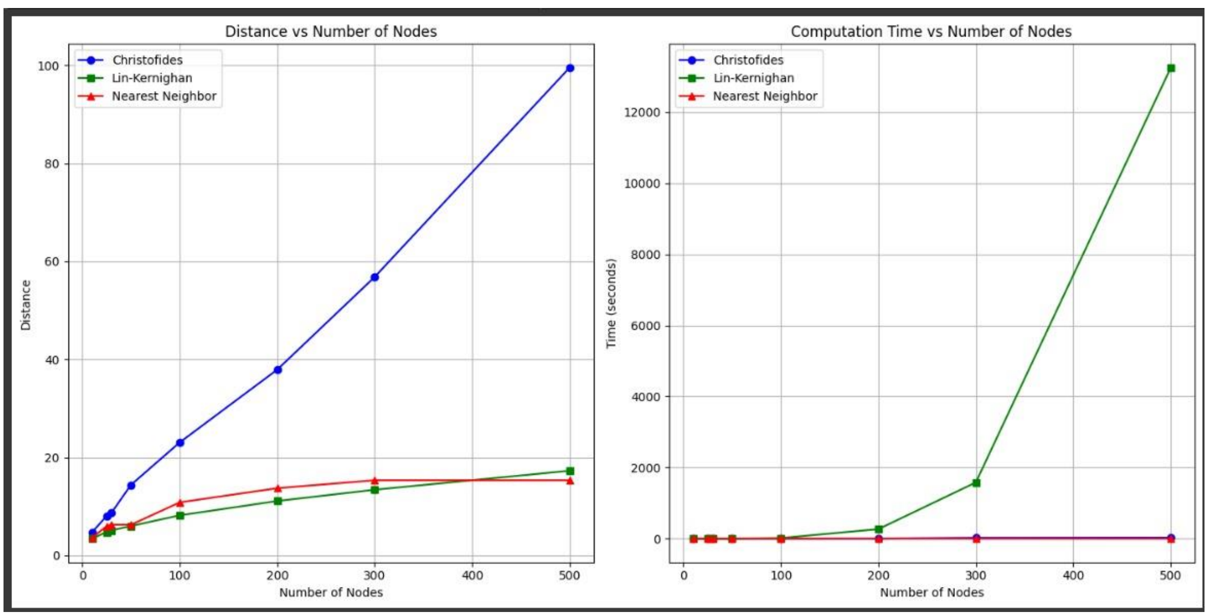
Based on the tabular data, we have done the following analysis and computed graphs as below.

1. Distance vs Number of Nodes Plot:

- The Lin-Kernighan algorithm maintains the shortest path across all node sizes, indicating its effectiveness in path optimization.
- The Nearest Neighbor algorithm shows a similar trend to Lin-Kernighan but at a slightly higher distance, which may still be efficient for certain cases.
- The Christofides algorithm's distance increases significantly with the number of nodes, suggesting it may not scale as well as the other algorithms for larger node sizes.

2. Computation Time vs Number of Nodes Plot:

- The computation time for the Nearest Neighbor algorithm remains relatively flat and minimal, indicating it is the fastest among the three.
- The Christofides algorithm shows a modest increase in computation time as the number of nodes grows.
- The Lin-Kernighan algorithm's computation time escalated dramatically with the number of nodes, especially from 300 to 500 nodes, suggesting that while it may provide the shortest path, it is also the most computationally intensive, particularly for larger datasets.



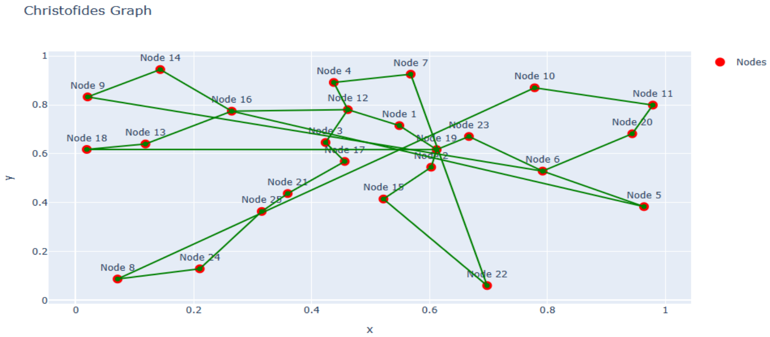
Inferences:

- Lin-Kernighan is the most optimal in terms of path length but may not be suitable for very large datasets due to its high computation time.
- Nearest Neighbor offers a good balance between path length and computation time, possibly making it suitable for time-sensitive applications.
- Christofides may be practical for smaller datasets where computation resources are limited, despite not always providing the shortest path.

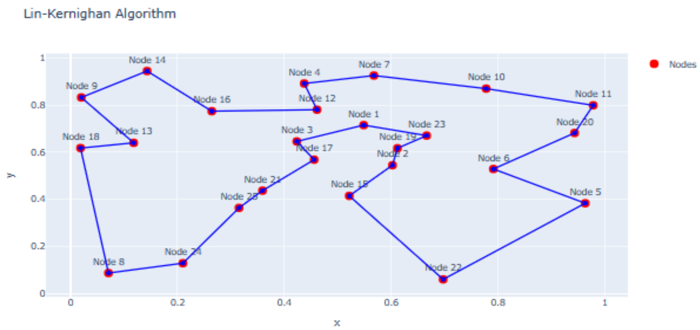
The choice of algorithm should thus consider the trade-off between accuracy in distance and computation time, with Lin-Kernighan being more suitable for accuracy-focused tasks, and Nearest Neighbor or Christofides for tasks where time efficiency is more critical.

Experiment 1 Outputs:

For 25:

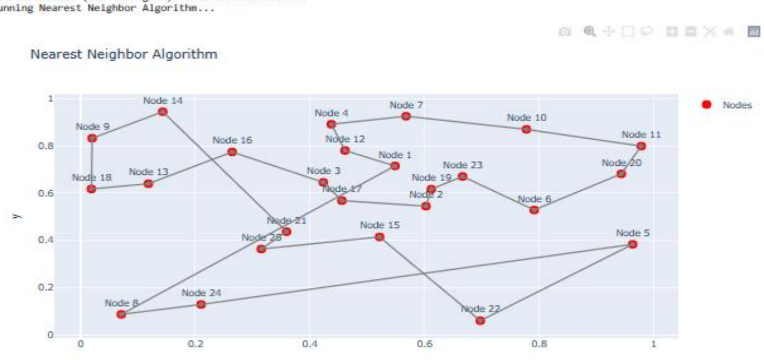



```
Best Path Distance (Christofides): 8.039280046287534
Execution time (Christofides): 0.0219268798828125
Running Lin-Kernighan Algorithm...
```



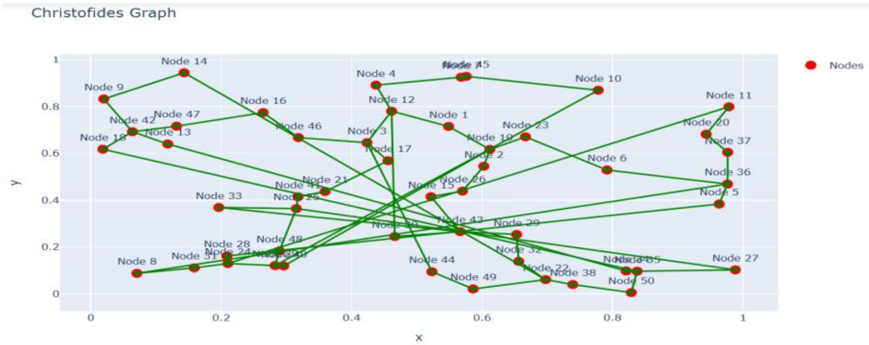

```
Best Path Distance (Lin-Kernighan): 4.786996897614424
Execution time (Lin-Kernighan): 0.12700700759887695
Running Nearest Neighbor Algorithm...
```

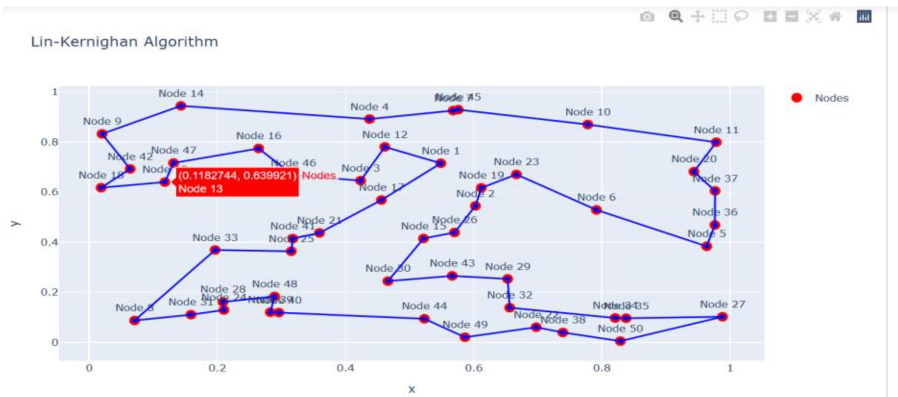
```
Running Nearest Neighbor Algorithm...
```

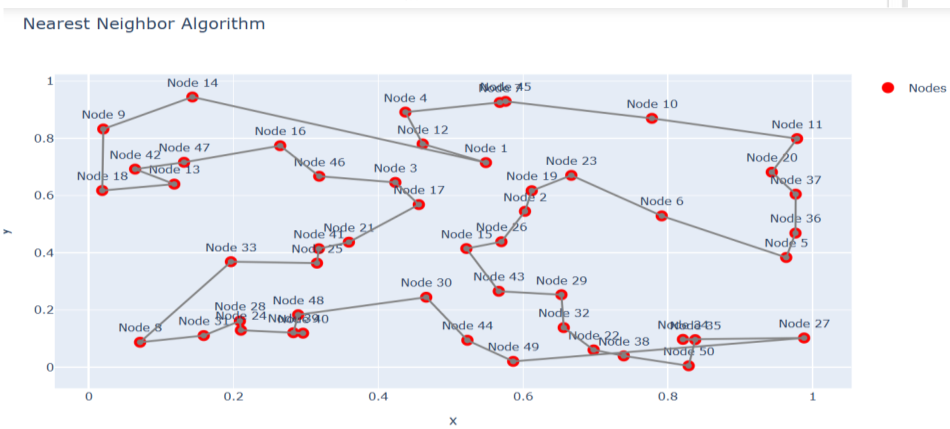



```
Best Path Distance (Nearest Neighbor): 5.976671497058518
Execution time(Nearest Neighbor): 0.0009989738464355469
```

For 50:







Please refer to [code file] for better understanding and clarity of visualizations

3.2.2 Real-World Data Experiment

The algorithms were applied to the Kaggle World Cities dataset. I divided the data into three clusters based on population to reduce the workload on the machine as well as for efficiency and for better insights on how the algorithms work.

We ran the experiment for country Australia and Pakistan and below is our Analysis and Inferences

Tabular Data :

Country	Population cluster	CH Algorithm	LK Algorithm	NN Algorithm	Distance wise best path(d)	Time wise best path(t)
Australia	High population	t=0.0928812026977539 d=25198520.43	d=12465409.412453061 t=0.004068446807861333	d=13501231.219921868 t=0.0005674362182617188	LK	NN
Australia	Medium population	d=49945917.0033081 t=0.45174694061279197	d=18345761.8760144 t=0.15427827835083008	d=22422375.637012288 t=0.00373744964599609 38	LK	NN
Australia	Low population	d=78096354.69445576 t=0.417501141357422	d=26362573.62841919 t=31.15951354904175	d=27310998.972326618 t=0.00363397598266601 56	LK	NN
Pakistan	HIgh	d=5445386.147144399 t=0.02542567253112793	d=3298777.3142610085 t=0.04314470291137695	d=3262578.9408481633 t=0.00017881393432617188	NN	NN
Pakistan	Medium	d=13952674.019575339 t=0.13047361373901367 t=	d=5835440.809502282 t=1.7842371463775635	d=7716836.980681908 t=0.00121235847473144 53	CH	NN
Pakistan	Low	d=10300384.849921709 t=0.222991943359375	d=5584460.534004124 t=1.5575568675994873	d=6877444.865621893 t=0.00056576728820800 78	LK	NN

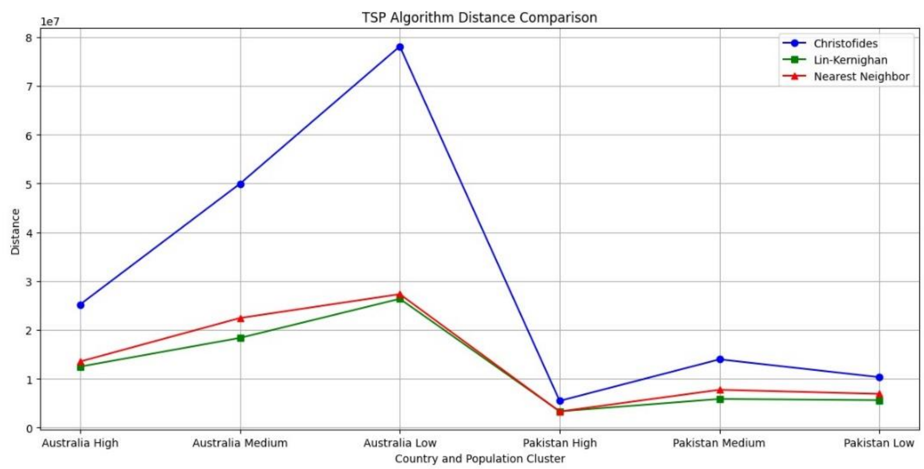
Analysis:

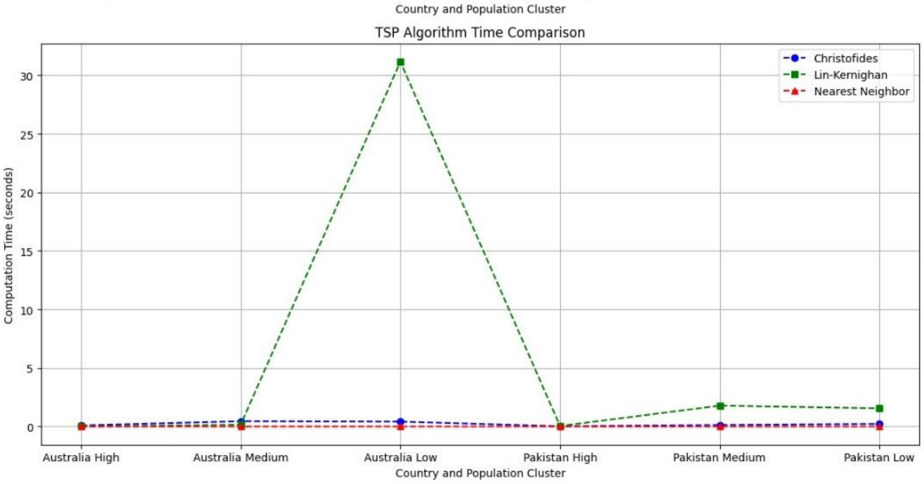
1. Distance Comparison vs Country and Population Clusters Plot:

- The Christofides algorithm consistently results in the longest routes across both countries and all population clusters, which could be an indication of its less optimal pathfinding ability in these cases.
- The Lin-Kernighan algorithm generally finds much shorter paths compared to Christofides, particularly in the Australian clusters, signifying its superior optimization for path length.
- The Nearest Neighbor algorithm provides competitive results, with paths that are closer in length to those found by Lin-Kernighan, suggesting it might offer a good balance between simplicity and efficiency.
- For Australia, the ranking from shortest to longest path is consistent: Lin-Kernighan, Nearest Neighbor, and Christofides. However, in Pakistan, the Nearest Neighbor algorithm seems to be performing the best, especially in high and low population clusters.

2. Time Comparison vs Country Population Clusters Plot:

- The Nearest Neighbor algorithm demonstrates the lowest computation times across all datasets, indicating it is the fastest algorithm among the three.
- The Christofides algorithm shows slightly higher computation times than Nearest Neighbor but remains relatively low, suggesting moderate computational efficiency.
- The Lin-Kernighan algorithm exhibits a dramatic spike in computation time for the low population cluster in Australia, indicating that its run time can significantly increase with the complexity or size of the dataset, potentially making it less practical for larger problems or where computation resources are constrained.





Inferences:

- While Lin-Kernighan seems to be the most effective in distance optimization, its computational expense can be significant, as shown in the Australia low cluster scenario.
- The Christofides algorithm appears to be a middle ground in terms of distance but has the advantage of consistent and low computation times.
- Nearest Neighbor's speed makes it an attractive choice for time-critical applications, although it may sacrifice some distance efficiency.

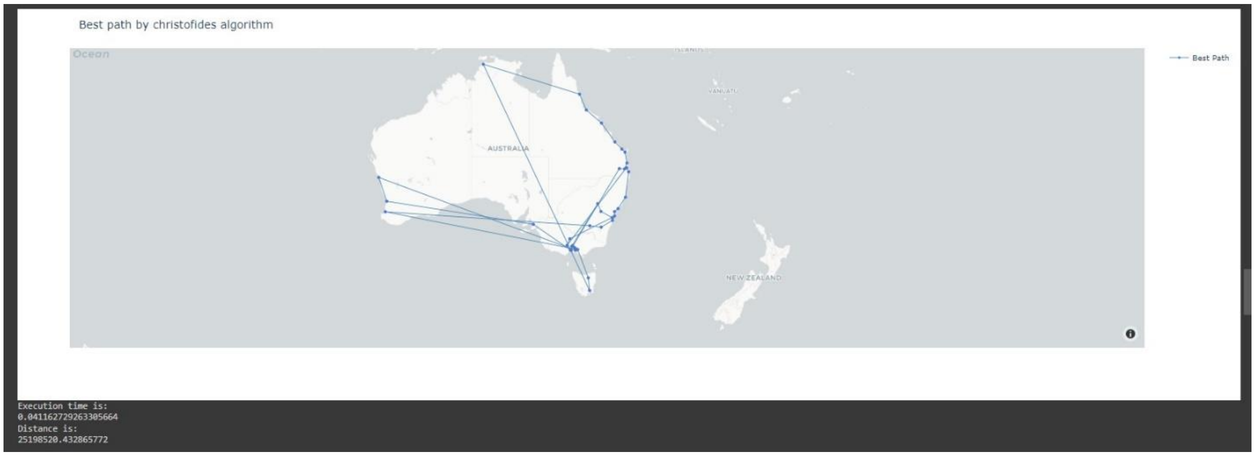
It's also noteworthy that the dataset's characteristics, such as population clustering, have a notable impact on the performance of the algorithms, which suggests that algorithm choice should be tailored to the specific nature of the dataset for optimal performance.

Experiment 2 Outputs:

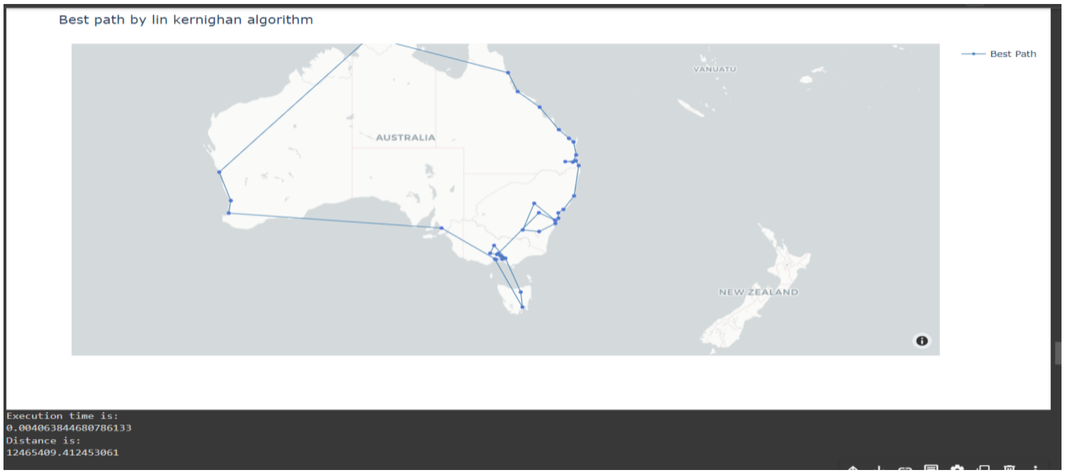
For Australia

1. High Population

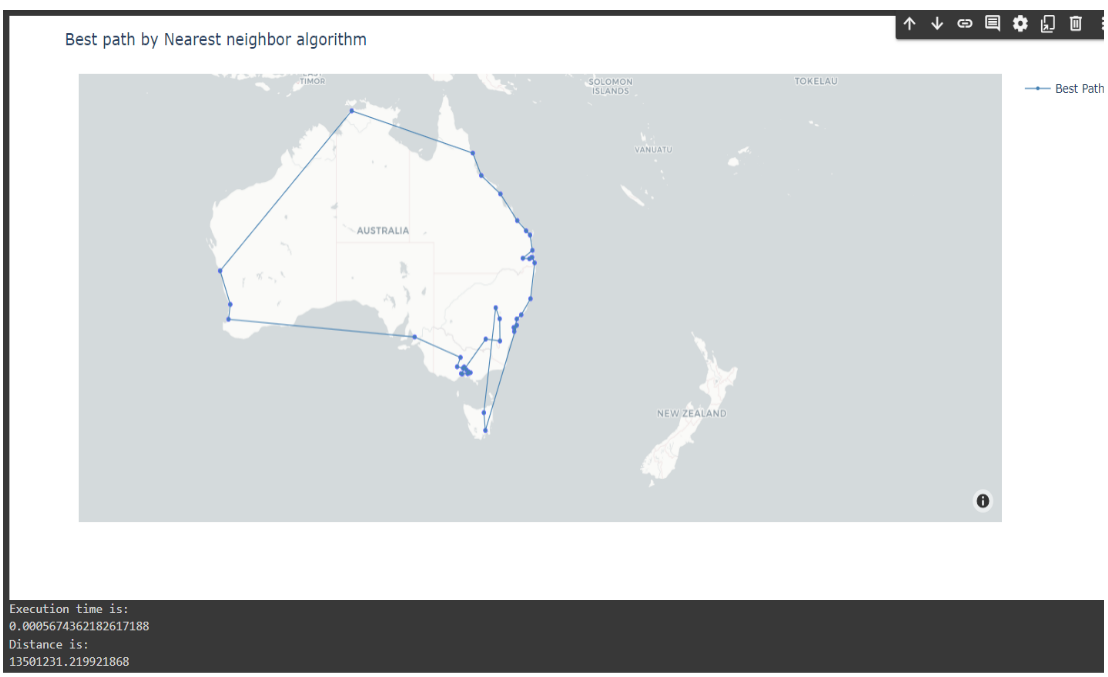
Christofides:



Lin kernighan:

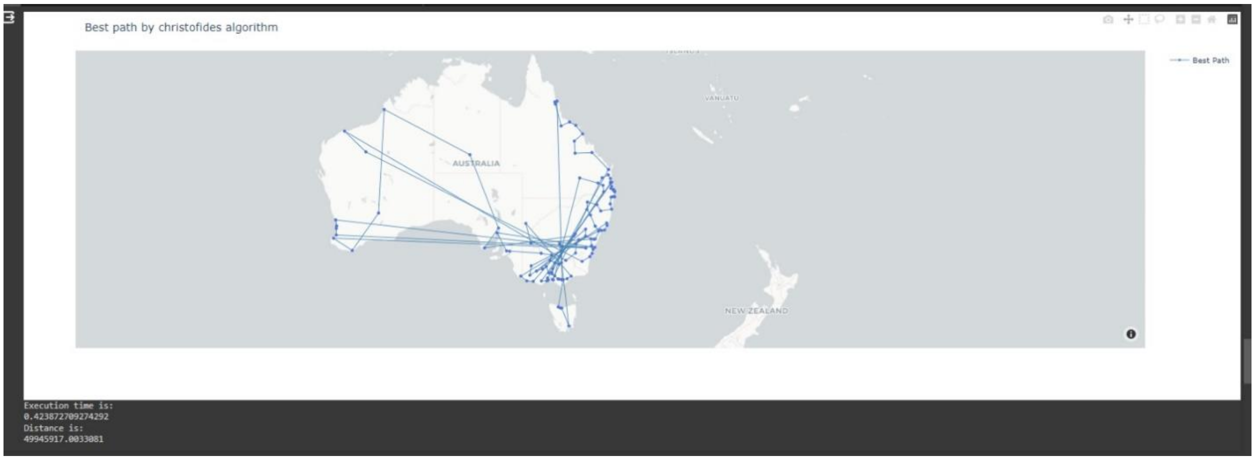


Nearest neighbor:

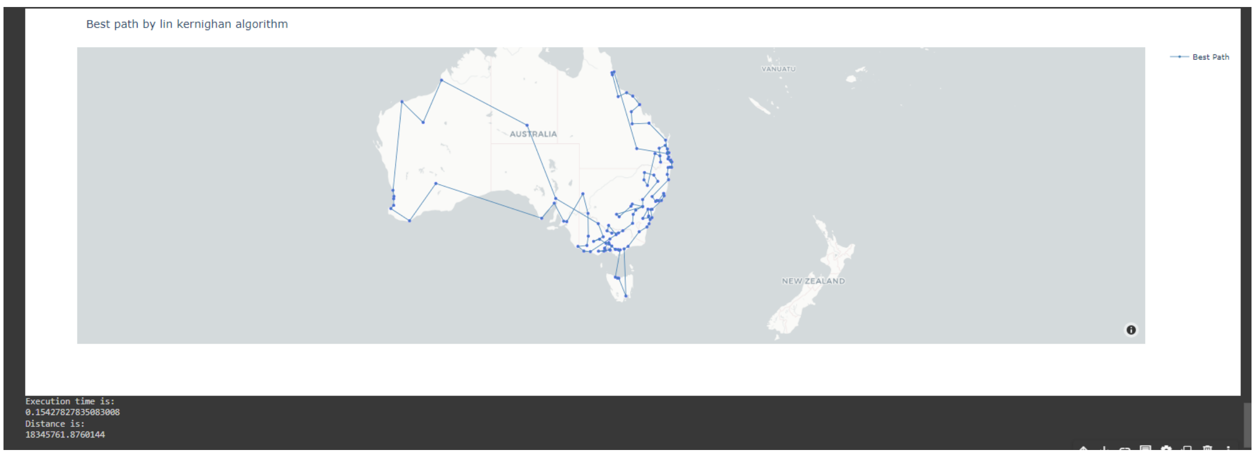


2. Medium Population

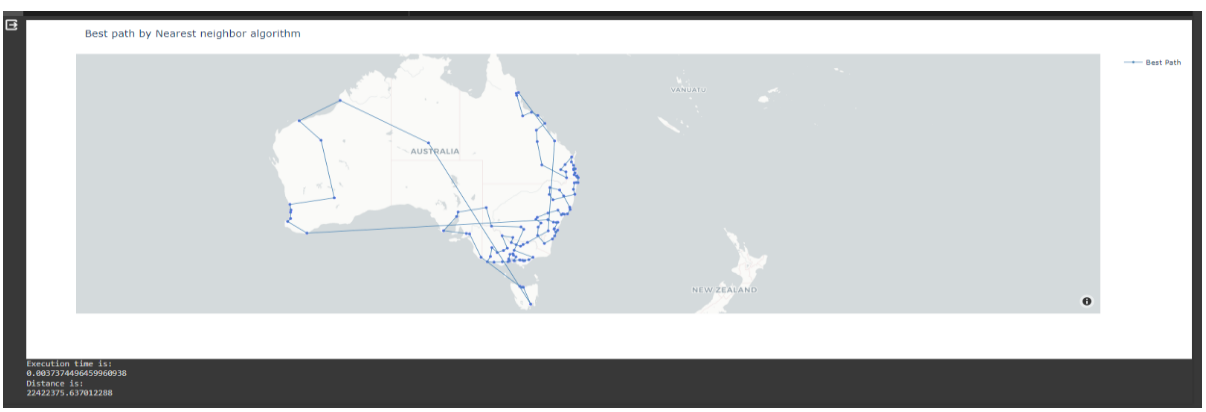
Christofides:



Lin kernighan:



Nearest neighbor:

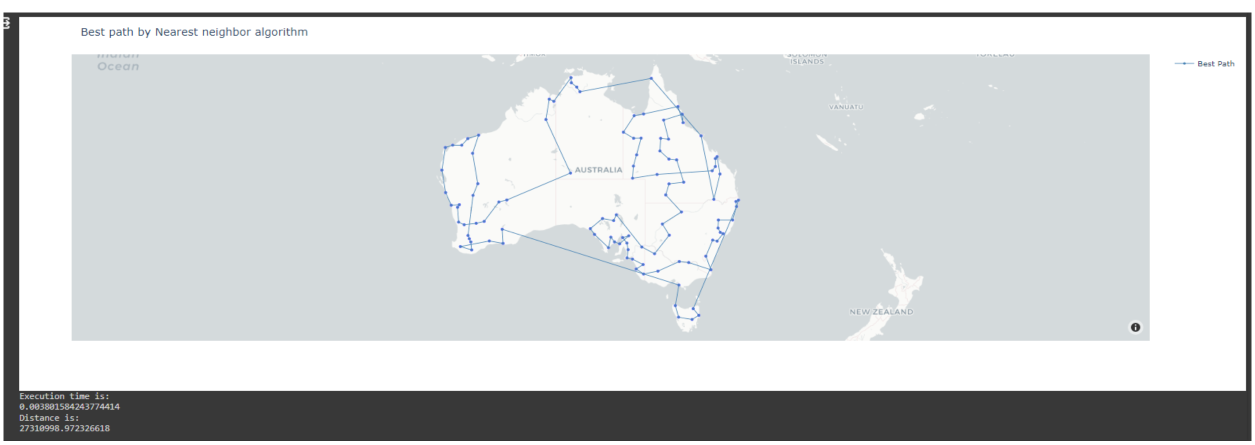


3. Low Population

Christofides:

Lin kernighan:

Nearest neighbor:



Please refer to [code file] for better understanding and clarity of visualizations

4. Experimental Analysis and Results Interpretation

The comprehensive evaluation across both random and real-world datasets **underscored the distinctive strengths and applications** of the Christofides, Nearest Neighbor, and Lin-Kernighan algorithms in solving the Traveling Salesman Problem (TSP). **The Lin-Kernighan heuristic demonstrated superior performance** in minimizing the total distance of the tour across various node sizes, showcasing its robustness in generating efficient solutions. However, **this efficiency comes at a computational cost**, as observed in the increased execution time for larger instances. The algorithm's intricate edge-swapping mechanism, while potent in refining the tour, necessitates significant computational resources, particularly as the number of cities escalates.

Conversely, **the Christofides algorithm emerged as a more balanced option,** offering a near-optimal solution with considerably lower computational demands. Its approximation method, which guarantees a solution within 1.5 times the optimal length for metric spaces, presents a pragmatic approach for instances where computational efficiency is prioritized over the absolute shortest path. This characteristic makes Christofides particularly valuable for large-scale problems where a slightly longer path is acceptable in exchange for faster computation.

The **Nearest Neighbor algorithm**, with its greedy selection process, **proved to be the fastest in terms of computation time**. However, its simplicity and reliance on local decisions without revisiting previous choices can lead to suboptimal solutions, especially in more complex or densely connected networks. This algorithm serves well as a quick, initial solution that can be further refined using more sophisticated methods or in scenarios where the computational budget is extremely limited.

Source code: https://github.com/tago893/Algoproject-TSP/tree/main

5. Conclusion

We conclude the following:

1. The choice among these algorithms' hinges on the specific requirements and constraints of the TSP instance at hand**. For applications demanding the utmost efficiency** in the solution's quality, regardless of computational expenses, **the Lin-Kernighan algorithm stands out** as the preferred choice. Meanwhile, **the Christofides algorithm** offers a **compelling compromise** between **solution quality and computational efficiency**, making it suitable for a wide array of practical applications, especially when dealing with very large datasets. Lastly, the **Nearest Neighbor algorithm** provides an expedient solution that can be **beneficial for quick approximations or as a baseline** for more complex heuristics**.**

2. **This analysis** underscores the **importance of** selecting an **appropriate algorithm** based on the **specific demands** and constraints of the task, with considerations for both the solution's quality and the computational resources available. Through our experiments, it becomes evident that while **no single algorithm** universally **excels in all aspects**, the strategic application of these algorithms based on the problem context can lead to effective and efficient solutions for the Traveling Salesman Problem.

6. Future works

Our current results indicate that the existing library implementations of the Lin-Kernighan heuristic surpass our algorithm's performance. Moving forward, we aim to refine our approach by intensifying the iterative nature of our Lin-Kernighan heuristic, closely aligning it with the sophisticated strategies employed by library versions. The focus will be on improving the iterative exchange mechanism, thereby increasing the solution's accuracy and computational efficiency. Our goal is to enhance our algorithm to achieve parity with, or exceed, the performance metrics of the established library implementations.

7. Citations

[1] S. Lin and B. W. Kernighan, "An experimental analysis of some computer-based algorithms for the Traveling Salesman Problem," 1971.
[2] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," 1973.
[3] N. Christofides, "Worst-case analysis of a simple heuristic for the Traveling Salesman Problem," 1976.
[4] Genova, K., & Williamson, D. P. (2017). An experimental evaluation of the best-of-many Christofides' algorithm for the traveling salesman problem. Algorithmica, 78(4), 1109-1130.
[5] G. Gutin and A. P. Punnen, "The Traveling Salesman Problem: A Computational Study," 2002.
[6] D. S. Johnson, G. Gutin, L. A. McGeoch, A. Yeo, W. Zhang, and A. Zverovitch, "A Computational Study of the Traveling Salesman Problem," 2002.
[7] D. S. Johnson and L. A. McGeoch, "An Experimental Evaluation of the Lin-Kernighan Traveling Salesman Heuristic," 1997.
[8] Simplemaps, "World Cities Data," available at https://simplemaps.com/data/world-cities.