RESEARCH ARTICLE

A 2-Step Nearest Neighbor Approach to TSP for Mobile Data Collection Route Optimization in IoT

Abul Kalam Azad[1,2,*]

[1] Department of Computer Science, The University of Alabama, Tuscaloosa 35401, United States
[2] Department of Computer Science and Telecommunication Engineering, Noakhali Science and Technology University, Noakhali 3814, Bangladesh

Abstract

The Traveling Salesman Problem (TSP) is a well-known NP-hard problem that aims to find the optimal tour among cities, visiting each exactly once and returning to the origin. Over the years, many researchers have proposed various approaches to find near-optimal solutions for TSP. This paper proposed a variant of the nearest neighbor algorithm termed the 2-step nearest neighbor algorithm. Unlike the original nearest neighbor approach, this algorithm considers two nearest nodes as potential next nodes. For each of those nearest nodes, the two nearest nodes have been considered. The sub-tour with the minimum cost among the resulting four has added to the final tour. This process continues until all nodes are covered. The performance of the proposed algorithm compared with that of existing brute-force, nearest-neighbor, and genetic algorithms. The proposed algorithm's utility extends from classical TSP instances to route planning for mobile data collectors in Internet of things (IoT) environments. In such scenarios, a mobile collector visits distributed IoT sensor nodes to gather data and return to its base station, minimizing total travel costs and making the problem analogous to TSP. Result analysis shows that in most scenarios, the proposed approach yields a shorter distance tour than the existing nearest neighbor approach, though with slightly increased computational time. The approach could also be adapted based on the number of nodes, the optimal solution requirement, and available computational resources. The code is available at https://github.com/azad-nstu/CS570-Proj-1-Traveling-Salesman-Problem-TSP.

Keywords: traveling salesman problem, optimization, brute force, nearest neighbor, genetic algorithm, internet of things.

Submitted: 22 June 2025
Accepted: 03 July 2025
Published: 21 July 2025

Vol. 1, **No.** 1, 2025.
10.62762/TMWI.2025.370327

***Corresponding author:**
Abul Kalam Azad
aazad1@crimson.ua.edu

1 Introduction

The TSP problem aims to find the optimal path among a set of cities, allowing a salesman to travel between them with minimum cost and return to the initial city [1]. So the solution lies in finding the shortest path to visit every city exactly once and coming back to the original city. Here, the cost could be money, distance, fuel consumption, time, or any other parameter, even though distance is often considered a cost. The problem is also called finding the minimum distance Hamiltonian cycle [2].

Citation
Azad, A. K. (2025). A 2-Step Nearest Neighbor Approach to TSP for Mobile Data Collection Route Optimization in IoT. *ICCK Transactions on Mobile and Wireless Intelligence*, 1(1), 1–10.

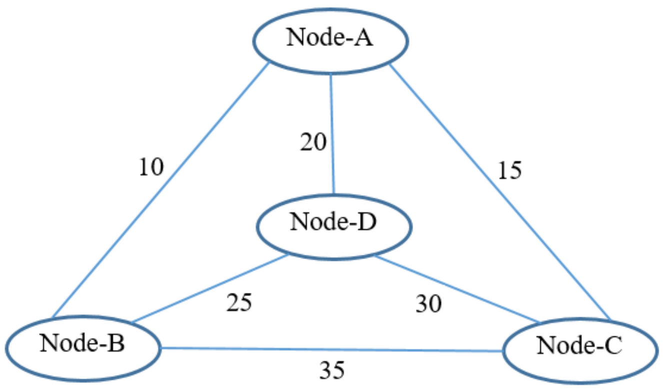


Figure 1. A sample graph with 4 nodes/cities.

Figure 1 illustrates four interconnected nodes, which could represent various entities like cities, sensor nodes, or IoT devices. Finding a TSP solution using a brute force approach would be easy as it has only 4 nodes and there will be $4! = 24$ combinations of tours. Here, if Node-A is considered as the starting node, visiting all other nodes B, C, and D, and coming back to Node A with minimum cost will solve this problem. The optimal solution for the problem is Node-A $\rightarrow$ Node-B $\rightarrow$ Node-D $\rightarrow$ Node-C $\rightarrow$ Node-A or, Node-A $\rightarrow$ Node-C $\rightarrow$ Node-D $\rightarrow$ Node-B $\rightarrow$ Node-A with cost $10 + 25 + 30 + 15 = 80$ is the minimum cost solution.

In TSP, to visit n number of nodes, for a complete graph (having a connection between each node), there will be $n!$ possible tours to cover all the nodes. So the complexity of the problem grows exponentially rather than polynomials. This problem is NP-hard because no optimal algorithms can find the exact solution (the smallest cost route) within a reasonable amount of time for a large TSP with many nodes.

Although the mathematics related to TSP was developed in the 1980s, the origin of TSP is not clear. It was first studied by Karl Menger in the 1930's. In 1954, a research team solved the TSP problem for 49 cities. Finally, in 2004, another research team solved the problem for 24,978 cities. During this period, various research teams successfully solved TSP instances for problems involving a range of cities, specifically between 49 and 24,978 [3].

There are many existing sub-optimal solutions for TSP [4, 5]. The nearest neighbor algorithm is one of the simple but effective approaches that follow the greedy approach concept to find a sub-optimal TSP solution. From the current node, the nearest neighbor considers only one nearest neighbor to add in the final tour that may miss a more effective sub-tour. So, this paper aims to explore 4 sub-tours with lengths of two edges each and then select the best sub-tour to add to the final tour. To minimize the probability of missing better sub-tours, the algorithm systematically explores four sub-tours. It does this by first selecting the two nodes closest to the current node. Subsequently, for each of these two selected nodes, it identifies two further nearest nodes, thereby constructing the four sub-tours.

TSP has numerous applications across various fields, including engineering, business, and medicine. Some of the TSP applications are in computer wiring, vehicle routing, drilling of printed circuit boards, overhauling gas turbine engines, X-ray crystallography, the order-picking problem in warehouses, mask plotting in PCB production, Genome Sequencing, Starlight Interferometer Program, Scan Chain Optimization, DNA Universal Strings, Power Cables, etc. [6–8].

Beyond its application in classical combinatorial optimization problems, route planning algorithms like TSP have direct relevance in mobile and wireless systems [9, 10], especially in scenarios involving mobile data collectors in wireless sensor networks (WSN) [11] or IoT [12] environments. In such settings, a mobile data collector (such as a drone, vehicle, or robot) traverses distributed IoT sensor nodes to gather data and then returns to a central base station. Efficient path planning in this context helps reduce total travel distance, energy consumption, and data collection latency. This problem is naturally modeled as a Traveling Salesperson Problem (TSP) variant, as the collector must visit each node exactly once and return to the starting point. This study proposes a 2-step nearest neighbor algorithm that can be effectively applied for such applications, providing a flexible and efficient route optimization approach in IoT-based mobile data collection systems [13]. Other potential applications for the proposed algorithm include autonomous vehicle patrol route planning [14], school bus routing problems [15], unmanned aerial vehicle (UAV) inspection missions [16], and agricultural drone crop monitoring [17].

The remainder of the paper is organized as Section 2 will outline three algorithms that will be used to compare the performance with the proposed algorithm. Section 3 covers a detailed discussion of the proposed 2-step nearest neighbor algorithm. A detailed description of the experiment and result analysis is presented in Section 4 before concluding the paper with some future directions in Section 5.

2 Algorithms for TSP

To solve the TSP problem, many algorithms proposed over the last few years. Some of the popular TSP algorithms include the brute force/naive approach, nearest neighbor [3], branch and bound [18], dynamic programming [19], genetic algorithm [20], and simulated annealing [21]. Different algorithms address this problem in different ways: some seek the exact solution, others use heuristics for an approximate solution, and some optimize the solution. In this study, the performance of the proposed approach is compared with that of existing brute-force, nearest-neighbor, and genetic algorithms. So, a brief overview of these algorithms is presented below:

2.1 Brute Force Approach

The brute force approach [22] is also called the naive approach to solving TSP that offers an exact solution. However, this approach is highly inefficient and impractical within a reasonable time for a medium to large number of nodes. The brute force approach finds the shortest Hamiltonian cycle by trying every possible tour. It generates all potential tours, calculates the distance for each, and then selects the tour with the lowest distance as the best solution for the Traveling Salesperson Problem. So, for n nodes (cities), the total combination will be $n!$. For example, consider the complete graph with 4 nodes shown in Figure 2. There are a total of $4!$ (24) possible combinations of tours. Among these, a minimum distance of 15 can be achieved by a couple of tours, such as $ABCDA$. Consequently, the algorithm's complexity is $O(n!)$, which is very high and impractical with many nodes.

2.2 Nearest Neighbor

Nearest neighbor [3] is the most simple and straightforward heuristic solution for a traveling salesman problem that tries to find a satisfactory result other than the optimal solution like brute force. For medium to large numbers of nodes (vertex), nearest neighbor algorithms can compute a decent path within a reasonable time. It follows a greedy approach that always considers the nearest neighbor. Then, move to that neighbor and find its nearest neighbor. Continue the process until all the nodes (cities) visit. Finally, comes back to the starting node and computes the distance.

For example, in Figure 2, if 'A' is the starting node, the nearest node of A is 'B'. Now from 'B', the nearest node (except A and B) is C. From C, the nearest node (Except A, B, and C) is D. Now all the nodes have

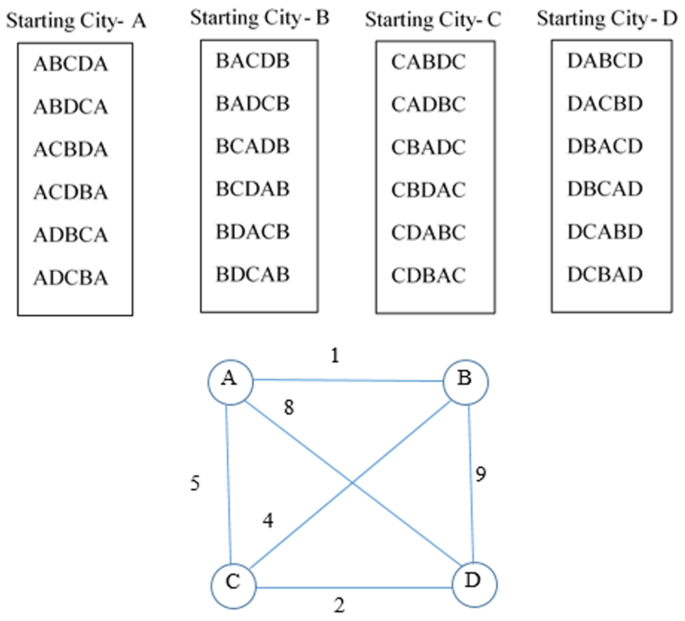


Figure 2. A complete graph with 4 nodes (cities).

visited. So, D to A returns back to the starting node 'A'. So, the shortest nearest neighbor tour is:$ABCDA$ (here, same as brute force but will not be the same always), and the distance is: $1 + 4 + 2 + 8 = 15$. The nearest neighbor time complexity for n is $O(n^2)$.

2.3 Genetic Algorithm

Genetic algorithms leverage the principle of natural evolution for optimization and intelligent searching. This approach eliminates the need to check all possible solutions, enabling them to find a good result within a reasonable amount of time. It starts with initializing a population of individuals (potential solution, tour in TSP) where each individual is called a chromosome that consists of a sequence of genes (cities in TSP) [20, 23].

In Figure 2, if the population size is 3 (in practice, the population size is high like 50, 100, 200, etc), the initial population could hold chromosome $ABCDA$, $BCADB$, and $DABCD$ usually generated randomly. Then, calculate the fitness (tour distance) of each individual and select individuals from this current population to consider as parents for the next generation. The selection process usually considers the fitness score. After that new individuals are created for the next generation by combining the genes of the selected parents. This step is called crossover or recombination. It also introduces mutation by randomly changing the genes of individuals to maintain genetic diversity and to avoid getting trapped in local minima. After that, it organizes new populations for the next generation by combining new offspring individuals and current individuals. The process continues for a certain number of generations or until reaching a termination condition. Finally, from the last generation, the best individual (a tour that has the best fitness value/ lowest distance) has been selected as the solution for TSP. The time complexity of the genetic algorithm is $O(n^2)$.

3 Proposed 2-step nearest neighbor algorithm

The proposed 2-step nearest neighbor algorithm is a variation of the nearest neighbor algorithm. In the nearest neighbor algorithm, a node considers its nearest node as the next node in the minimum distance tour construction. Then, from this selected nearest node, consider the nearest neighbors (that were not selected before). In this process, although if a node selects its nearest node as the next node in the shortest tour, the next nearest node from the selected node might cost high compared to the selection of another node and from that node, there may be the shortest edge so that total cost for the two edge might be less. For example, in Figure 3, if A is the starting node, using the nearest neighbor, it will select B as the next nearest neighbor and will add it to the tour. Then, from B, it will select its nearest neighbor node E from its neighbor nodes C, D, and E (Excluding A, as it's already in the tour). The distance from $A \rightarrow B \rightarrow E$ is $1 + 7 = 8$. Now if starting node A would select C as the next node and from C, E as the next node, the distance from $A \rightarrow C \rightarrow E$ is: $2 + 1 = 3$.

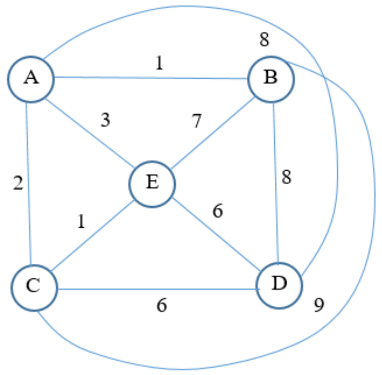


Figure 3. A complete graph with 5 nodes.

To address this issue, this study proposes a 2-step nearest neighbor algorithm. It's called a 2-step nearest neighbor because the selection decision to add nodes in the tour is taken after considering two levels of neighbors. Here, a node initially considers its two nearest neighbors that didn't select before, and then for each nearest neighbor, find two more nearest neighbors (again from those that didn't select before) that result in 4 probable selected sub-tours with two edges each. Out of these for sub-tour, the algorithms select the lowest distance sub-tour and append it for the lowest distance tour construction. After that, the last node of the appended sub-tour finds its two nearest nodes that didn't append before. Then again for each two nearest nodes, find two more nearest nodes, and out of 4 sub-tours, select and append the lowest distance sub-tour. Continue the process until all nodes in the graph are selected in the tour. Finally, append the edge from the last selected nodes to the source node for coming back to the source node and completing the Hamiltonian cycle. The tour and distance using the nearest neighbor for the graph is: $ABECDBA$, and $1 + 7 + 1 + 6 + 8 = 23$.

Now for Figure 3, using the proposed algorithm, source node A initializes $distance = 0$, will consider two nearest neighbors, B and C, and for each B $(distance - 1)$ and C $(distance - 2)$, it will consider two more nearest nodes that not considered yet. So for B, the next two nearest nodes are E $(distance - 7)$ and D $(distance - 8)$. For C, the next two nearest nodes are E $(distance - 1)$ and D $(distance - 3)$. So, there will be four sub-tours ABE $(distance - 1 + 7 = 8)$, ABD $(distance - 1 + 8 = 9)$, ACE $(distance - 2 + 1 = 3)$, ACD $(distance - 2 + 3 = 5)$. Out of these four sub-tours, the algorithm will select the lowest distance sub-tour ACE with $distance$ 3 and will append CE with the starting node A and update the sub-tour from A to ACE. It will also update the $distance$ from 0 to 3. In the next iteration, E will become the starting node. The algorithm will then identify its two nearest neighbors. For each of these neighbors, it will consider two further nearest nodes to form new sub-tours. The sub-tour with the lowest distance will then be selected and appended to the existing tour. It will add the distance of the selected sub-tours with the existing distance to calculate the updated distance. The process continues until all nodes are included in the sub-tour. Finally, the starting node is added to the last node in the sub-tour, and its corresponding distance is also added in. Since only two nodes remain after ACE in this graph, the algorithm will add the next two nearest nodes in ascending order of their $distance$: first D (with a $distance$ of 6), and then B (with a $distance$ of 8). Now, the sub-tour will be $CAEDB$, and the updated distance is $3 + 6 + 8 = 17$. Finally, it will add edge BA to go back to the starting node and the final tour will be $ACEDBA$ and final distance $17 + 1 = 18$. For Figure 3, the proposed 2-step nearest neighbor algorithm could be further described utilizing Figure 4, which shows the selected sub-tours $(ACE\ldots)$ with their distance.

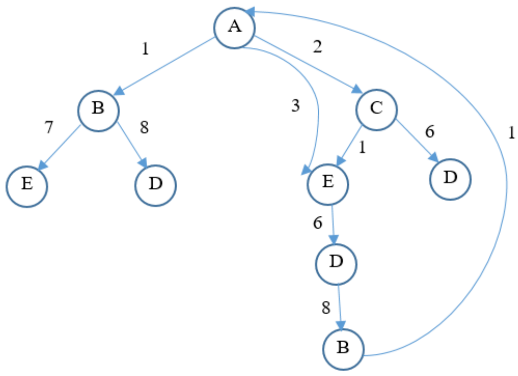


Figure 4. Sub-tour selection process of proposed 2-step nearest neighbor algorithm.

4 Experiment and result analysis

4.1 System Configuration

Brand: HP. Processor: 11th Gen Intel(R) Core i5-1135G7, 2.40GHz, 2419 Mhz, 4 Core(s), 8 Logical Processor(s). RAM: 16 GB.

Programming language: Python version 3.9.13

4.2 Experiment

For the experiment, I have considered 11 graphs with different node numbers: 5 nodes, 10 nodes, 13 nodes, 15 nodes, 25 nodes, 50 nodes, 100 nodes, 250 nodes, 500 nodes, 750 nodes, and 1000 nodes.

For the experiment, the following steps were performed:

- Step 1: Read the given lower triangular matrix file.
- Step 2: Convert it to a complete adjacency matrix.
- Step 3: Draw the complete graph.
- Step 4: Apply four algorithms using the complete graph.
- Step 5: Draw the graph with the algorithms' generated minimum tour.
- Step 6: Store the result of each algorithm for a different number of nodes.
- Step 7: Perform result analysis based on distance and execution time.

Step 1 reads the lower triangular matrix from the input graph file. Then, step 2 makes the complete adjacency matrix. After that, in step 3, the complete graph is drawn. A sample graph for 5 nodes and 10 nodes with the distances among different nodes is presented in Figure 5, and Figure 6. Also, a graph for 100 nodes is presented in Figure 7. With increasing the number

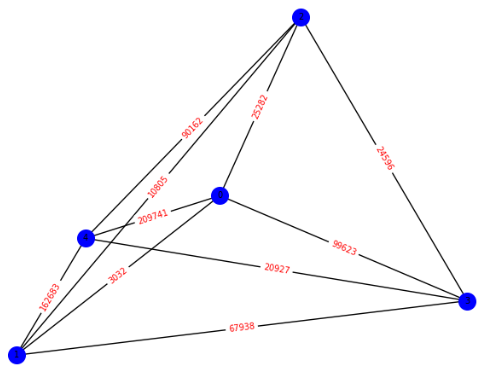


Figure 5. A graph with 5 nodes.

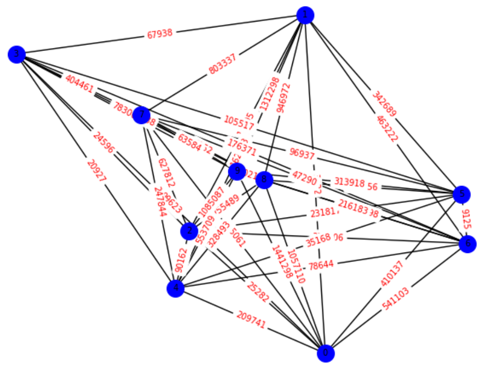


Figure 6. A graph with 10 nodes.

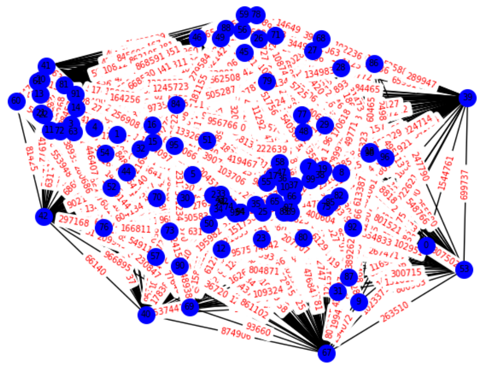


Figure 7. A graph with 100 nodes.

of nodes, visualization becomes challenging to see the distances among different nodes. In the next step (step 4), 4 different algorithms (Brute force, Nearest neighbor, Genetic algorithm, Proposed 2-step nearest neighbor) are implemented. The overall complexity of the proposed algorithm is also $O(n^2)$.

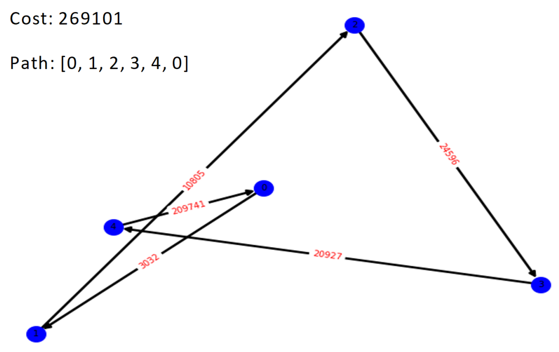


Figure 8. Proposed 2-step NN algorithms generated minimum tour graph for 5 nodes with start node=0.

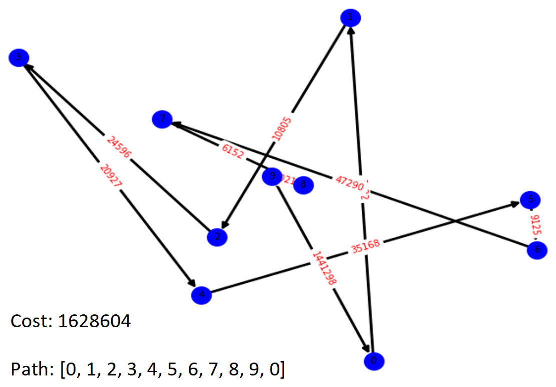


Figure 9. Proposed 2-step NN algorithms generated minimum tour graph for 10 nodes with start node=0.

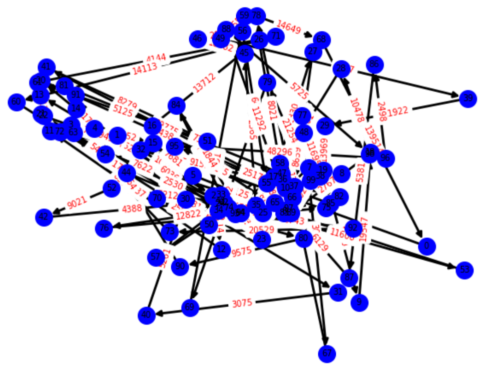
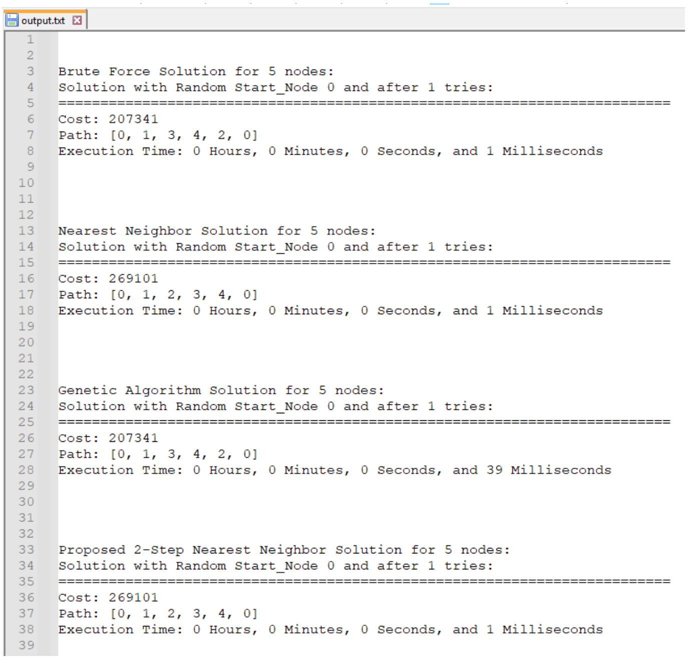


Figure 10. Proposed 2-step NN algorithms generated minimum tour graph for 100 nodes with start node=0.

Then, in step 5, draw and store the same graph using the edges of the tour. This time, the edges were drawn as directed to understand the direction of visit from the starting node to all other nodes and coming back to the starting node. Figures 8, 9, and 10 represent the minimum distance tour returned by the proposed 2-step nearest neighbor algorithms for the completed graph with 5 nodes, 10 nodes, and 100 nodes, and that were presented in Figures 5, 6, and 7.

In step 6, after running the algorithms for different graphs with different numbers of nodes, the output



```
Brute Force Solution for 5 nodes:
Solution with Random Start_Node 0 and after 1 tries:
=======================================================================
Cost: 207341
Path: [0, 1, 3, 4, 2, 0]
Execution Time: 0 Hours, 0 Minutes, 0 Seconds, and 1 Milliseconds



Nearest Neighbor Solution for 5 nodes:
Solution with Random Start_Node 0 and after 1 tries:
=======================================================================
Cost: 269101
Path: [0, 1, 2, 3, 4, 0]
Execution Time: 0 Hours, 0 Minutes, 0 Seconds, and 1 Milliseconds



Genetic Algorithm Solution for 5 nodes:
Solution with Random Start_Node 0 and after 1 tries:
=======================================================================
Cost: 207341
Path: [0, 1, 3, 4, 2, 0]
Execution Time: 0 Hours, 0 Minutes, 0 Seconds, and 39 Milliseconds



Proposed 2-Step Nearest Neighbor Solution for 5 nodes:
Solution with Random Start_Node 0 and after 1 tries:
=======================================================================
Cost: 269101
Path: [0, 1, 2, 3, 4, 0]
Execution Time: 0 Hours, 0 Minutes, 0 Seconds, and 1 Milliseconds
```

Figure 11. Stored results in the output.txt file.

distance (minimum distance), path (minimum tour), and Execution time are stored in an output file named 'output.txt'.

Here, in the test, start node = 0 was assumed, although the code could select the start node randomly. A sample format of the output result is stored in the output.txt file shown in Figure 11. Finally, in step 7, result analysis in terms of minimum distance/cost, and execution time was performed for four algorithms with different numbers of nodes.

4.3 Result Analysis

Table 1 shows the generated minimum distance, and execution time by four algorithms for different numbers of nodes. Out of the four algorithms, brute force was run using up to 13 nodes because of its high computational complexity of $O(n!)$, which will take a very long time to finish. The other three algorithms were executed for all 11 selected graphs. For the genetic algorithm, the population size and number of generations were set to 200 for graphs up to 100 nodes, and for graph size from 100 to 1000 nodes, the population size and number of generations were set to 500. For the mutation, 2 random nodes in the individuals were selected and swapped to introduce the diversity. Also, before selecting the new individuals, the probability (function of fitness and temperature) of the new individual was calculated and compared with a threshold.

To compare the performance of different algorithms

Table 1. Brute force, NN, Genetic algorithm, and proposed 2-step NN generated distance and consumed time for different number of nodes.

Number of Nodes	Brute Force Distance	Brute Force Time	NN Distance	NN Time	Genetic Algorithm Distance	Genetic Algorithm Time	Proposed- 2-Step NN Distance	**Proposed 2-Step NN Time**
5	207341	1 ms	269101	1 ms	207341	39 ms	269101	**1 ms**
10	647105	259 ms	1628604	2 ms	647105	50 ms	1628604	**2 ms**
13	628904	7 Min, 42 s, and 921 ms	1801463	2 ms	661905	61 ms	1823748	**2 ms**
15	Didn't analyze	Didn't analyze	1820321	2 ms	628203	62 ms	1820321	**3 ms**
25	Didn't analyze	Didn't analyze	950756	3 ms	1134269	62 ms	862827	**3 ms**
50	Didn't analyze	Didn't analyze	1371802	3 ms	6812631	69 ms	1139795	**3 ms**
100	Didn't analyze	Didn't analyze	3179955	5 ms	24086044	109 ms	3088761	**6 ms**
250	Didn't analyze	Didn't analyze	2609719	13 ms	58526357	469 ms	1944536	**20 ms**
500	Didn't analyze	Didn't analyze	1999082	46 ms	142035502	1s, and 113 ms	1703980	**73 ms**
750	Didn't analyze	Didn't analyze	2824495	96 ms	262161280	1s, and 634 ms	2387702	**157 ms**
1000	Didn't analyze	Didn't analyze	1780054	169 ms	389579130	2 s, and 320 ms	4126961	**283 ms**

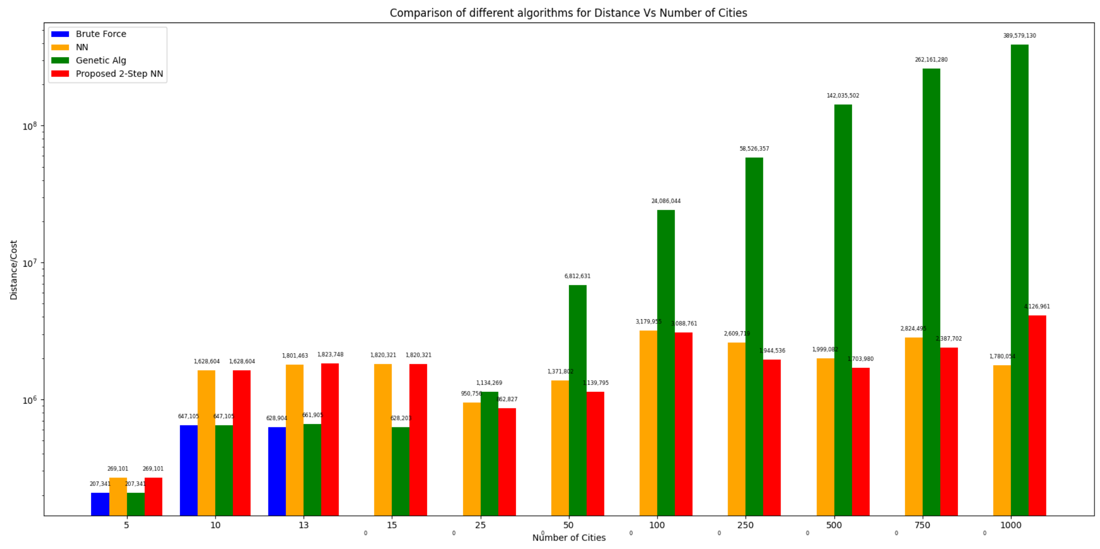


Figure 12. Comparison of different algorithms generated shortest distance for different graph sizes.

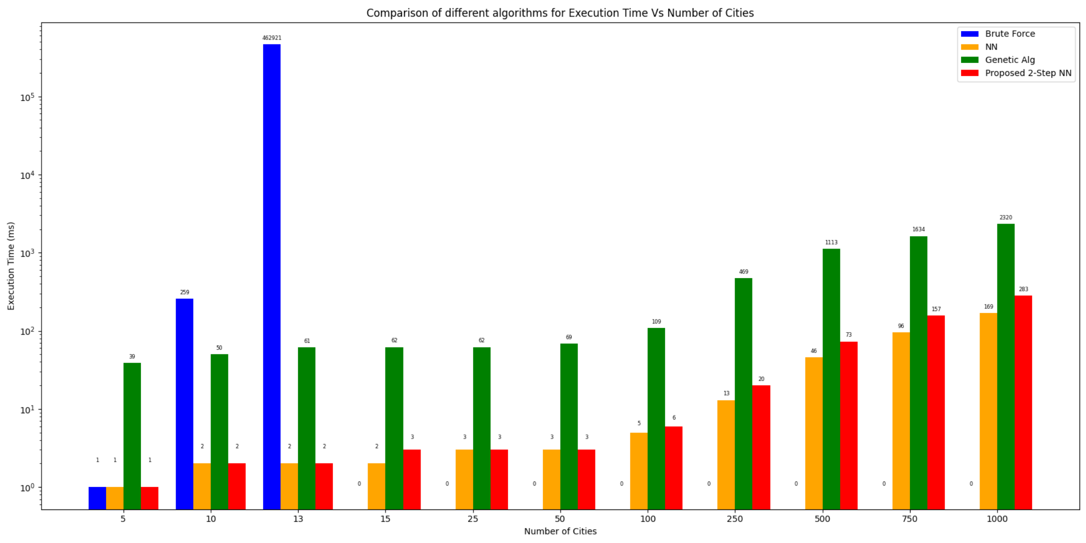


Figure 13. Comparison of different algorithms consumed execution time for different graph sizes.

in terms of generated minimum distance for different graph sizes, Figure 12 was drawn using the distance data from Table 1. Here, distances were represented in a logarithmic scale for better visualization. Figure 12 shows that for small graphs, the genetic algorithm performed well as brute force, but with increasing the graph size, the performance degrades rapidly. We can also see that for small graph sizes (5 to 15 nodes), the performance of the proposed 2-step NN algorithm is almost the same as the original nearest neighbor algorithm. But with increasing the number of nodes from 25-750, the proposed 2-step NN performed well compared to the nearest neighbors and in most cases, the genetic algorithm as well. It happens due to the idea described in Figure 3 in Section 3. However, genetic algorithm performance could improve with optimized parameters, which were not fully explored here [24]. The performance of proposed 2-step algorithms degrades with further increasing the number of nodes like 1000. However, a better performance could also be achieved with a large graph size if the algorithm explores more number of nearest neighbors and in more depth (than 2 in this experiment) so that it could consider many potential sub-tours (here, it was 4) before selecting the lowest distance sub-tour. With increasing the number of neighbors and sub-tour lengths, the algorithm will become closer to brute force. So, the proposed algorithm could adapt between brute force and nearest neighbors depending on the requirement and available resources.

Then, to compare the execution time of different algorithms for different numbers of nodes, execution times from Table 1 were taken (all converted to millisecond (ms) units) and drawn in Figure 13. Again, execution time was represented in logarithmic scale for better visualization. It also illustrates the exact execution time (in ms) of each algorithm for different graph sizes. From Figure 13, we can see that the brute force execution time increases very rapidly with increasing the number of nodes from 5 to 13. In [5], stated that if a system can evaluate a path in 1 nanosecond, it will take 10 million years to evaluate all the possible paths in a graph having only 25 nodes. It discouraged testing the algorithm for the graph with more nodes. Overall, the execution time of the genetic algorithm is higher than the nearest neighbor and the proposed 2-step nearest neighbor algorithm. Between, the nearest neighbor and the proposed algorithm, the proposed algorithm consumes a little more time than the nearest neighbor because it considers 4 sub-tours before taking 1. For all algorithms, the execution time increased with increasing the number of nodes in the graph.

Overall, for most graphs (especially with nodes up to 750), the proposed 2-step nearest neighbor algorithm achieves a lower distance tour than the nearest neighbor with the expense of a little more execution time.

5 Conclusion and future work

In conclusion, this paper introduces a 2-step nearest neighbor algorithm, a variant of the traditional nearest neighbor approach. It aims to find a more optimal solution by exploring four sub-tours, each comprising two edges, before making a selection. The two nearest neighbors from the current node and two more nearest neighbors for each of them contribute to the four sub-tour generation process. Performance analysis proves its effectiveness for finding low-cost tours than existing nearest neighbors for graphs with low to medium numbers of nodes (5 to 750). For a high number of nodes, like 1000, the performance dropped because 4 sub-tours might not be sufficient considering many potential sub-tours. The issue could be resolved by exploring more sub-tours with high lengths (more than two), and that would be studied as future work. However, for the execution time, the proposed algorithm needs a little more computational time than the existing nearest neighbor.

Moreover, the proposed approach can be adapted for route planning in IoT environments that rely on mobile data collectors. In such systems, where a collector device sequentially visits multiple distributed IoT sensor nodes for data gathering and returns to a base station, the problem naturally aligns with the TSP formulation. Future work will evaluate the performance of the proposed algorithm with other existing TSP solutions. The investigation will also integrate this algorithm into mobile edge computing frameworks and IoT-based delay-tolerant wireless environments to evaluate its practical impact in real-world IoT mobile data collection applications.

Data Availability Statement

The source code and relevant data supporting the findings of this study are publicly available at the following GitHub repository: https://github.com/azad-nstu/CS570-Proj-1-Traveling-Salesman-Problem-TSP.

Funding

This work was supported without any funding.

Conflicts of Interest

The author declares no conflicts of interest.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Dahiya, C., & Sangwan, S. (2018). Literature review on travelling salesman problem. *International Journal of Research*, *5*(16), 1152-1155.

[2] Kühn, D., & Osthus, D. (2012). A survey on Hamilton cycles in directed graphs. *European Journal of Combinatorics*, *33*(5), 750-766. [CrossRef]

[3] Hahsler, M., & Hornik, K. (2008). TSP—infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, *23*, 1-21. [CrossRef]

[4] Nemani, R., Cherukuri, N., Rao, G. R. K., Srinivas, P. V. V. S., Pujari, J. J., & Prasad, C. (2021, November). Algorithms and optimization techniques for solving tsp. In *2021 Fifth international conference on I-SMAC (IoT in social, mobile, analytics and Cloud)(I-SMAC)* (pp. 809-814). IEEE. [CrossRef]

[5] Yang, L., Wang, X., He, Z., Wang, S., & Lin, J. (2023, December). Review of traveling salesman problem solution methods. In *International Conference on Bio-Inspired Computing: Theories and Applications* (pp. 3-16). Springer Nature Singapore. [CrossRef]

[6] Lenstra, J. K., & Kan, A. R. (1975). Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, *26*(4), 717-733. [CrossRef]

[7] Matai, R., Singh, S. P., & Mittal, M. L. (2010). Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, *1*(1), 1-25.

[8] Akshatha, P. S., Vashisht, V., & Choudhury, T. Comparison Of Various Mutation Operators Of Genetic Algorithm To Resolve Travelling Salesman Problem.

[9] Suriya Praba, T., Sethukarasi, T., & Venkatesh, V. (2020). Krill herd based TSP approach for mobile sink path optimization in large scale wireless sensor networks. *Journal of Intelligent & Fuzzy Systems*, *38*(5), 6571-6581. [CrossRef]

[10] Pavlenko, O., Tymoshenko, A., Tymoshenko, O., Luntovskyy, A., Pyrih, Y., & Melnyk, I. (2022, February). Searching extreme paths based on travelling salesman's problem for wireless emerging networking. In *IEEE International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering* (pp. 284-304). Cham: Springer Nature Switzerland. [CrossRef]

[11] Azad, A. K., Alam, M. S., & Shawkat, S. A. (2019). DCDS-MAC: A Dual-Channel Dual-Slot MAC Protocol for Delay Sensitive Wireless Sensor Network Applications. *J. Commun.*, *14*(11), 1049-1058.

[12] Aouedi, O., Vu, T. H., Sacco, A., Nguyen, D. C., Piamrat, K., Marchetto, G., & Pham, Q. V. (2024). A survey on intelligent Internet of Things: Applications, security, privacy, and future directions. *IEEE communications surveys & tutorials*, 27(2), 1238-1292. [CrossRef]

[13] Heidari, A., Shishehlou, H., Darbandi, M., Navimipour, N. J., & Yalcin, S. (2024). A reliable method for data aggregation on the industrial internet of things using a hybrid optimization algorithm and density correlation degree. *Cluster Computing*, *27*(6), 7521-7539. [CrossRef]

[14] Chen, X. (2012). Fast patrol route planning in dynamic environments. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *42*(4), 894-904. [CrossRef]

[15] Park, J., & Kim, B. I. (2010). The school bus routing problem: A review. *European Journal of operational research*, *202*(2), 311-319. [CrossRef]

[16] Jordan, S., Moore, J., Hovet, S., Box, J., Perry, J., Kirsche, K., ... & Tse, Z. T. H. (2018). State-of-the-art technologies for UAV inspections. *IET Radar, Sonar & Navigation*, *12*(2), 151-164. [CrossRef]

[17] Hafeez, A., Husain, M. A., Singh, S. P., Chauhan, A., Khan, M. T., Kumar, N., ... & Soni, S. K. (2023). Implementation of drone technology for farm monitoring & pesticide spraying: A review. *Information processing in Agriculture*, *10*(2), 192-203. [CrossRef]

[18] Balas, E., & Toth, P. (1983). Branch and bound methods for the traveling salesman problem.

[19] Simonetti, N. O. (1998). *Applications of a dynamic programming approach to the traveling salesman problem*. Carnegie Mellon University.

[20] Bryant, K. (2000). Genetic algorithms and the travelling salesman problem.

[21] Aarts, E. H., Korst, J. H., & van Laarhoven, P. J. (1988). A quantitative analysis of the simulated annealing algorithm: A case study for the traveling salesman problem. *Journal of Statistical Physics*, *50*, 187-206. [CrossRef]

[22] Baidoo, E., & Oppong, S. O. (2016). Solving the TSP using traditional computing approach. *International Journal of Computer Applications*, *152*(8).

[23] Sharma, V., Kumar, R., & Tyagi, S. (2020). Optimized solution of TSP (Travelling Salesman Problem) based on mendelian inheritance. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, *13*(5), 909-916. [CrossRef]

[24] Rexhepi, A., Maxhuni, A., & Dika, A. (2013). Analysis of the impact of parameters values on the Genetic Algorithm for TSP. *International Journal of Computer Science Issues (IJCSI)*, *10*(1), 158.

Abul Kalam Azad is currently a Ph.D. student in Computer Science at The University of Alabama, Tuscaloosa, USA. He earned a B.Sc. (Engg.) degree from the Department of Computer Science and Telecommunication Engineering of Noakhali Science and Technology University, Bangladesh, in 2010, and an M.S. (research-based) degree from the Institute of Information and Communication Technology of Bangladesh University of Engineering and Technology, Bangladesh, in 2016. Before joining The University of Alabama, he served as an Assistant Professor in the Department of Computer Science and Telecommunication Engineering at Noakhali Science and Technology University, Bangladesh. His research interests include ML, cybersecurity, the Internet of Things (IoT), and wireless sensor networks. (Email: aazad1@crimson.ua.edu)