SciEn
Journal of Engineering Advancements Vol. 05(01) 2024, pp 19-26 https://doi.org/10.38032/jea.2024.01.004

Improvement of the Nearest Neighbor Heuristic Search Algorithm for Traveling Salesman Problem

Md. Ziaur Rahman[1], *Sakibur Rahamn Sheikh*[2], *Ariful Islam*[2], *and Md. Azizur Rahman*[2,*]

[1]Department of Mathematics, Bangladesh University of Engineering and Technology (BUET), Dhaka-1000, Bangladesh
[2]Mathematics Discipline, Khulna University, Khulna-9208, Bangladesh

Received: December 21, 2023, Revised: February 27, 2024, Accepted: March 01, 2024, Available Online: March 30, 2024

ABSTRACT

The Traveling Salesman Problem (TSP) is classified as a non-deterministic polynomial (NP) hard problem, which has found widespread application in several scientific and technological domains. Due to its NP-hard nature, it is very hard to solve effectively and efficiently. Despite this rationale, a multitude of optimization approaches have been proposed and developed by scientists and researchers during the last several decades. Among these several algorithms, heuristic approaches are deemed appropriate for addressing this intricate issue. One of the simplest and most easily implementable heuristic algorithms for TSP is the nearest neighbor algorithm (NNA). However, its solution quality suffers owing to randomness in the optimization process. To address this issue, this study proposes a deterministic NNA for solving symmetric TSP. It is an improved version of NNA, which starts with the shortest edge consisting of two cities and then repeatedly includes the closest city on the route until an effective route is established. The simulation is conducted on 20 benchmark symmetric TSP datasets obtained from TSPLIB. The simulation results provide evidence that the improved NNA outperforms the basic NNA throughout most of the datasets in terms of solution quality as well as computational time.

Keywords: Combinatorial Optimization, Traveling Salesman Problem (TSP), Heuristic Algorithm, Nearest Neighbor Algorithm, Improved Nearest Neighbor Algorithm

1 Introduction

Combinatorial optimization problems are of interest to several academic disciplines, including theoretical computer science, artificial intelligence, operations research, and discrete mathematics. One of the most prominent combinatorial optimization problems is the traveling salesman problem (TSP), which is studied in several disciplines including mathematics, artificial intelligence, physics, operations research, and biology. The task at hand pertains to the identification of the most efficient route connecting a collection of cities, with the constraint that each city be visited only once before returning to the initial city [1]. The origins of the Traveling Salesman Problem are believed to have been uncovered in Vienna in 1920 [2]. In 1954, Dantzig *et al.* [3] provided a formal elucidation of the traveling salesman problem. Subsequently, this methodology has been extensively employed to simulate and analyze various practical scenarios, encompassing domains such as hardware design, microchip design, radio-electronic device design, data association, data transmission in computer networks, DNA sequencing, vehicle routing, job scheduling, clustering of data arrays, image processing and pattern recognition, crystal structure analysis, transportation, logistics, and supply chain management [4]-[5]. The TSP is characterized by its comprehensibility, although it often poses challenges when attempting to find a solution due to its inclusion of all relevant components inside a combinatorial optimization framework. Undoubtedly, the computational time required to solve the TSP increases exponentially as the number of cities increases, as shown by Hore *et al.* [5]. Hence, the investigation into enhancing the solution method for the TSP has significant theoretical, technical, and practical implications.

In graph theory, the TSP can be defined symmetrically on a full undirected graph $G = (V, E)$ or asymmetrically on a directed graph $G = (V, A)$, where $V = \{1,2,3,\dots n\}$ is the set of vertices, $E = \{(i,j): i,j \in V, i < j\}$ is a set of edge and $A = \{(i,j): i,j \in V, i \neq j\}\}$ is a set of arcs. On E or on A a cost matrix $C = (c_{ij})$ is defined. Each edge is assigned a cost, which is the distance between cities i and j, can be defined as [6]:

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Depending on the distance matrix C, the TSP can be categorized as symmetric or asymmetric. G is symmetric TSP if $c_{ij} = c_{ji}$ and asymmetric TSP if $c_{ij} \neq c_{ji}$. In this paper we use symmetric TSP (sTSP). The objective function Z written as [6]:

$$Z = Min \sum_{i,j\in V, i<j} c_{ij}x_{ij} \tag{1}$$

and decision variable

$$x_{ij} = \begin{cases} 1 \ ; \ \textit{the routes connects cities i and j} \\ 0 \ ; \textit{otherwise} \end{cases} \tag{2}$$

with respect to the following constrains [6]:

$$\sum_{i,j\in V} x_{ij} = n \tag{3}$$

$$\sum_{i<k} x_{ik} + \sum_{k<j} x_{kj} = 2 \tag{4}$$

$$\sum_{i<j} x_{ij} \leq |T| - 1 \ (T \subset V, 2 \leq |T| \leq n - 2) \tag{5}$$

Here, Eq. (1) represents the objective function, which aims to minimize the overall distance, Eq. (2) denotes the decision variable, while Eqs. (3) - (5) provide the constraints that must be satisfied in the model. a binary variable x_{ij} is associated with each edge (i,j) in the graph G, as shown by Eq. (2). The values 1 and 0 of x_{ij} indicate whether each edge (i,j) should be included or excluded from the optimum route. As seen in the Eq. (3) above, it is evident that every possible path, including the optimal path, must consist of exactly n edges. According to Eq. (4), it is necessary to choose exactly two edges for every vertex. This constraint facilitates the establishment of itineraries in which each city is visited just once, with the salesman ultimately returning to the initial location. Eq. (5) serves as a constraint that prohibits the creation of sub routes with fewer vertices than the total number of vertices, denoted as n. This requirement ensures that all cities are visited [6].

Due of its applicability and complexity, several scholars have conceived and developed different optimization techniques in the past few decades to cope with the TSP issue. Heuristic algorithms are the most successful and frequently utilized search approach for tackling the TSP issue among these algorithms [1]. One of the simplest and most easily implementable heuristic algorithms for TSP is the Nearest Neighbor Algorithm (NNA). However, its solution quality suffers from the randomness inherent in the optimization process. In this paper, we improve the basic NNA for symmetric TSP. Indeed, the improved version of NNA is a deterministic approach that begins with an edge of the two closest cities and connects them simultaneously with the next-closest cities one by one until feasible routes are formed. The proposed improved version shows better performance than the basic NNA in terms of solution quality as well as simulation time. The present paper is organized as follows. Some engineering applications of the TSP are presented in Section 2. In Section 3, we review some related works to solve symmetric problem. The methods of study including both improved and basic NNA are presented in detail in Section 4. In Section 5, the results are given and discussed, and Section 6 concludes the study with a future plan.

2 Some Engineering Applications of TSP

The traveling salesman problem (TSP) is an extensively studied problem in computer science and optimization theory, but it also has numerous real-life applications in diverse engineering fields. Here are some engineering applications of TSP:

Circuit Board Manufacturing: In electronics manufacturing, the TSP can be used to optimize component placement on circuit boards. By finding the shortest path that visits all the required connection points (components), engineers can minimize the length of interconnect lengths, reduce signal delays, and optimize the layout for space and efficiency.

Robotics: TSP algorithms are used in robotics for motion planning, task allocation, and multi-robot coordination. These algorithms help discover a shortest path for a robot to traverse multiple points in a given environment while satisfying constraints such as avoiding obstacles and obeying motion limits (e.g., maximum speed, acceleration). This is crucial for tasks such as robotic exploration, surveillance, and delivery in known or unknown terrain environments.

DNA Sequencing: In bioinformatics, the TSP has been adopted to solve DNA sequencing problems, where the goal is to determine the most efficient order in which to sequence fragments of DNA to reconstruct the original sequence. By leveraging TSP algorithms, researchers can enhance the efficiency, accuracy, and scalability of DNA sequencing workflows and data analysis pipelines.

Wireless Sensor Networks (WSNs): TSP algorithms provide powerful optimization tools to address various challenges in WSNs, including data collection, energy efficiency, coverage optimization, fault detection, and dynamic network management. With the TSP algorithms, researchers and engineers can design more efficient and robust WSNs for a wide range of applications, including environmental monitoring, smart infrastructure, and IoT systems.

Vehicle Routing and Logistics: One of the most common applications of TSP is optimizing routes for delivery vehicles, such as trucks, drones, or even autonomous vehicles. By finding the shortest route that visits a set of locations (cities or delivery points), companies can minimize fuel consumption, reduce travel time, and improve overall efficiency in logistics operations while meeting various operational constraints.

Urban Planning: In urban planning, TSP algorithms can assist in optimizing routes for garbage collection trucks, street cleaning vehicles, and other municipal services, leading to more efficient use of resources and reduced traffic congestion.

VLSI Chip Design: In VLSI (Very Large Scale Integration) chip design, TSP algorithms are utilized for tasks such as wire routing and layout optimization. By finding the shortest paths to connect different components on the chip, engineers can reduce cable length, reduce signal delay, and optimize chip area and power consumption.

These are just a few examples, and the applications of the TSP in engineering are diverse and continually evolving as new challenges arise in various fields.

3 Related Works

Nearest Neighbor Algorithm (NNA) is one of the simplest heuristic route construction algorithms. For a long time, researchers have been working on this route construction algorithm for solving TSP. In this section, we review some works that researchers have done recently on the route construction algorithm for TSP.

Hore *et al.* [5] offered a greedy algorithm-based solution to the traveling salesman issue. The greedy algorithm is like the Nearest Neighbor Algorithm (NNA), and the route starts from that particular sub-route with two cities, which has the shortest distance among all such feasible sub-routes. Although such an algorithm usually does not give the global optimum solution, it has been considered the initial solution. These algorithms are compared to their proposed algorithm. The suggested approach outperformed the conventional approaches and was determined to be more effective than the VNS-1 and VNS-2 algorithms on average. In their paper, Naser *et al.* [7] introduced a deterministic methodology that used a multi-perfect matching and partitioning technique to approximate the solution of the symmetric traveling salesman problem (STSP). The first step was identifying the most cost-effective combination of sub-routes that encompassed all cities and had a minimum of four edges for each sub-route. The performance of the proposed method is assessed and contrasted with the optimum values achieved by other established strategies for solving the Symmetric Traveling Salesman Problem (STSP). The simulation results presented in this paper indicate that the methodology used by the researchers yields solutions that are either optimum or very close to optimal within a polynomial time frame.

Halim and Ismail [8] conducted a comparative analysis of heuristic strategies in the Traveling Salesman Problem (TSP). The study focused on six heuristic approaches, namely Nearest Neighbor, Genetic Algorithm, Simulated Annealing, Tabu Search, Ant Colony Optimization, and Tree Physiology Optimization. The comparison of computation, accuracy, and convergence has been conducted in this research. Bentley [9] employed a double-sided NN method, which allows the route to improve on both ends. This method uses nearest neighbour (NN) search on both ends of the route to find the path with the shortest length. On the other hand, Klug *et al.* [10] have recently expanded the NN technique to k-RNN for solving STSP and ATSP. The simulation results indicated that the solution quality of the 2-RNN algorithm remains rather consistent, ranging from around 10% to 40% higher than the optimal solution.

Bakar and Ibrahim [11] used a heuristic shortest route methodology in order to determine the optimal solution for the TSP. This study proposes a modified strategy that combines the heuristic shortest distance method and fuzzy approach for addressing a network with an erroneous arc length. The investigation focused on the determination of the network's arc length, as well as the analysis of the interval number and triangular fuzzy number. Subsequently, the revised methodology was used to address a particular instance of the Traveling Salesman Problem (TSP). The overall shortest distance obtained using this strategy was then compared to the total distance generated by employing a conventional nearest neighbor heuristic technique. The findings indicate that the modified methodology yields a sequence of visited cities that is equivalent to the conventional technique. Additionally, it provides a reliable measure of the total shortest distance, which is less than the total shortest distance calculated by the old approach. Consequently, the findings of this study have the potential to enhance the existing methodologies used in addressing the TSP.

The heuristic approach proposed by Lin *et al.* [12] for the TSP is highly commendable. The authors introduced a heuristic approach that demonstrated significant success in generating optimal and near-optimal solutions for the STSP. The methodology was formulated employing a comprehensive heuristic approach that possesses the potential to address diverse combinatorial optimization problems. The proposed methodology successfully produced optimal solutions for all the examined problems, encompassing both "traditional" problems documented in existing literature and randomly generated problems. The scope of the problems ranged up to a maximum of 110 cities. In terms of absolute values, it was observed that a typical issue involving 100 cities required less than 25 seconds for a single example (GE635) and approximately three minutes to reach the optimal solution with a confidence level exceeding 95%. In addition, some papers are reviewed on the Nearest Neighbor Algorithm (NNA) for TSP are mentioned in [13]-[17].

4 Methods of Study

4.1 Basic Nearest Neighbor Algorithm

The most elementary algorithm employed in the Traveling Salesman Problem (TSP) is the Nearest Neighbor Algorithm (NNA). The aforementioned approach efficiently produces a concise route, albeit infrequently yielding the optimal solution [18]. The basic NNA is used to determine a traveling salesman's itinerary. The salesperson begins in one city (at random), then travels to the city closest to the beginning city. After that, he travels to the nearest unexplored city and continues the procedure until all of the cities have been visited, at which point he returns to the beginning city. The basic NNA algorithm is as follows:

1. Make all vertices unvisited by default
2. Select a random vertex and make it the current vertex $\boldsymbol{u}$. Make a note that $\boldsymbol{u}$ has been visited
3. Find the shortest path between current vertex $\boldsymbol{u}$ and a previously visited vertex $\boldsymbol{v}$
4. Set the current vertex $\boldsymbol{u}$ to $\boldsymbol{v}$. Make a note that $\boldsymbol{v}$ has been visited
5. Terminate when all of the domain's vertices have been visited. Otherwise, proceed to step 3
6. Return to your starting city

However, its solution quality suffers owing to randomness. Due to the problem, we have proposed a revised version of the basic NNA. The basic NNA is mostly probabilistic because it cannot always provide the shortest route. But the improved algorithm is deterministic. The improved NNA is a route-construction algorithm. First, it chooses a random city from a list of cities in NNA. Then, using the shortest distance, travel to the nearest unexplored city. This process will be repeated until all cities have been visited and the player is forced to return to the starting point.

4.2 Improved Nearest Neighbor Algorithm

It began its route on the improved NNA with a short distance. Firstly, it sorts all edges, and then it takes a short edge with two vertices (or cities). This is the main difference between improved NNA and basic NNA. The next steps of the improved NNA are like those of the basic NNA. The mathematical analysis and algorithm of an improved NNA are discussed in the following subsection.

Let n be the number of cities and TSP can be defined symmetrically on a full undirected graph $G = (V, E)$, where $V = \{A_1, A_2, A_3, \dots, A_n\}$ is the set of vertices, $E = \{(A_i, A_j): A_i, A_j \in V, i > j\}$ is a set of edge. Then, calculate the distance of every possible edge and select the shortest edge contains two closest cites A_i and A_j which is our initial edge and expressed as $A_i \leftrightarrow A_j$.

The set of routes is,

$$X_1 = min\{A_iA_j : i = 1,2, \dots, n \text{ and } j = 1,2, \dots, n-1 \text{ and } A_iA_j \in E\}$$

Now, choose the nearest city A_k from the initial edge and connect the city which is expressed as

$$A_i \leftrightarrow A_j \leftrightarrow A_k$$

and the set of the route is,

$$X_2 = min\{X_1A_k : k = 1,2, \dots, n-2 \text{ and } A_k \in V - X_1\}$$

then, choose the nearest city A_l from the last visited city A_k and add with the current route which expressed as

$$A_i \leftrightarrow A_j \leftrightarrow A_k \leftrightarrow A_l \;.$$

The set of the route is,

$$X_3 = min\{X_2A_l : l = 1,2, \dots, n-3 \text{ and } A_l \in V - X_2\}.$$

Similarly taking every city, return to the initial edge where a city isn't connected (suppose A_i) and the route expressed as

$$A_i \leftrightarrow A_j \leftrightarrow A_k \leftrightarrow A_l \leftrightarrow . \;. \;. \leftrightarrow A_z \leftrightarrow A_i \text{ and}$$

the set of the route is,

$$X_{n-1} = min\{X_{n-2}A_z: z = 1,2,\dots,n-(n-1) \text{ and } A_z \in V - X_{n-2}\}$$

In addition, for more visualization, a flowchart of the improved NNA is shown in Fig. 1.

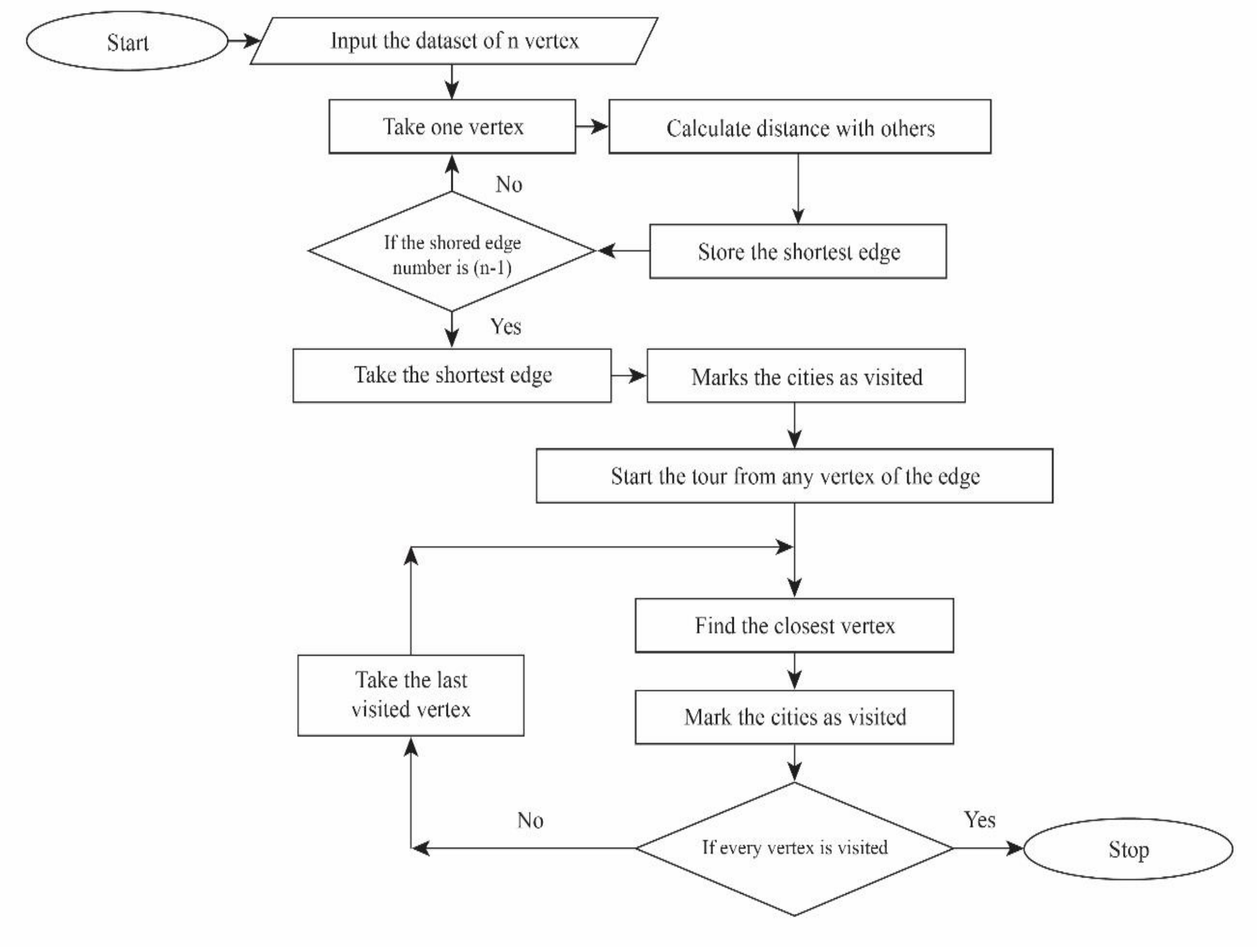
Start
Input the dataset of n vertex
Take one vertex
Calculate distance with others
No
If the shored edge number is (n-1)
Store the shortest edge
Yes
Take the shortest edge
Marks the cities as visited
Start the tour from any vertex of the edge
Find the closest vertex
Take the last visited vertex
Mark the cities as visited
No
Yes
If every vertex is visited
Stop

Fig. 1 Flowchart of the proposed improved nearest neighbor search algorithm for TSP

The improved nearest neighbor search algorithm can be explained by the following step by step procedure:

Step:1. Sort all edges from *n* cities.
Step:2. Select an edge with a shortage distance.
Step:3. From the remaining cities, find the nearest unvisited city and combine it with the existing edge.
Step:4. Make a note of the most recent city visited.
Step:5. Find the closest city to the most recent city visited.
Step:6. Add the closest city to the tour and mark as visited.
Step:7. Is there any city that has not yet been visited? If you responded yes, go to step 5.
Step:8. Return to the first chosen edge's starting vertex

5 Results and Discussion

In this particular section, a series of simulations were conducted on various datasets to assess and compare the performance of of the improved NNA to that of the basic NNA. In order to fulfil the simulation objectives, a collection of real-world symmetric Travelling Salesman Problem (TSP) datasets from TSPLIB is taken into consideration. Consider 20 benchmark symmetric TSP datasets with dimensions ranging from 52 to 2103. The dataset name is assigned a number value that corresponds to its dimension. As an example, the alphanumeric identifier "berlin52" represents the numerical value assigned to a specific node consisting of 52 geographical places inside the city of Berlin. Once the datasets have been gathered, it is necessary to compute a symmetric distance matrix in which the diagonal members are set to zero. The distance matrix provides a measure of the distance between the nodes. The evaluation of basic NNA included the computation of the best, worst, and average outcomes, as well as the measurement of the time taken to run the procedure across all datasets. In the improved NNA, each individual test case inside the simulation is executed autonomously, taking into consideration the size of the dataset. In contrast, it should be noted that in the simulation, every test case is executed autonomously, resulting in a twofold increase in the dataset lengths for the basic NNA. Both the basic NNA and improved NNA are implemented using MATLAB R2021a. The simulations are conducted on a computer system equipped with a CORE i5 processor operating at a frequency of 1.80 GHz and 4 GB of RAM.

The simulation findings and subsequent analysis including 20 benchmark datasets have been subjected to testing, comparing the performance of both the basic NNA and improved NNA. Table 1 illustrates the performance comparison between the basic NNA and improved NNA for 20 benchmark datasets. The first column in the table provides a description of the dataset names. The second column provides information pertaining to the quantity of cities. The third column describes the best-known optimal solution. After that, the fourth column describes the optimal solution, execution time of the CPU (in seconds), and error (%) for an improved NNA. The best, average, and worst results, the execution time of the CPU (in seconds), and the error rate (%) for the average and best of each dataset for basic NNA are described in the last column. Here, datasets are also arranged in an ascending order of nodes.

The formula for finding error of improved NNA is

$$\text{Error } (\%) = \frac{\text{result of improved NNA} - \text{optimum}}{\text{optimum}}$$

and formula we use for finding error of best and average solution of basic NNA are

$$\text{Error}_{\text{best}}\ (\%) = \frac{\text{best result of NNA} - \text{optimum}}{\text{optimum}}$$

$$\text{Error}_{\text{avg}}\ (\%) = \frac{\text{averge result of NNA} - \text{optimum}}{\text{optimum}}$$

Table 1 Performance comparison between basic NNA and improved NNA

Datasets	Nodes	Optimum Solution	Improved NNA			Basic NNA					
			Best	Time(s)	Error (%)	Best	Average	Worst	Time(s)	$\text{Error}_{\text{best}}$ (%)	$\text{Error}_{\text{avg}}$ (%)
berlin52	52	7542	9161	0.01368	21.4664	8149	9396	10188	0.03394	8.04823	24.583
rat99	99	1211	1577	0.01020	30.2229	1507	1695	1911	0.07070	24.4426	39.969
kroC100	100	20749	25519	0.00994	22.9890	26043	27925	30014	0.06900	25.5145	34.587
lin105	105	14379	17363	0.01073	20.7524	19759	21221	23448	0.12903	37.4157	47.582
pr107	107	44303	47233	0.01757	6.61354	46563	53181	60539	0.08233	5.10124	20.032
pr124	124	59030	69066	0.01107	17.0015	67302	75851	84709	0.10606	14.0131	28.496
ch130	130	6110	7341	0.01750	20.1472	7461	7988	8894	0.08022	22.1119	30.734
pr152	152	73682	85243	0.01570	15.6903	86665	94459	107039	0.08970	17.6201	28.192
u159	159	42080	55200	0.02709	31.1787	54509	60038	63711	0.19008	29.5369	42.678
rat195	195	2323	2624	0.03069	12.9573	2751	3023	3277	0.13011	18.4245	30.134
d198	198	15780	18485	0.01764	17.1419	18233	22085	24076	0.17593	15.5449	39.956
kroA200	200	29368	36824	0.01874	25.3881	35161	38291	42451	0.12714	19.7255	30.384
ts225	225	126645	149243	0.07770	17.8435	146769	160600	177560	0.31578	15.8908	26.811
pr264	264	49135	57663	0.02865	17.3562	56947	60446	65021	0.20568	15.8995	23.022
lin318	318	42029	52883	0.04197	25.8250	53621	56250	60102	0.33150	27.5805	33.831
fl417	417	11861	14773	0.05673	24.5510	14603	16323	17539	0.63161	23.1178	37.610
rat575	575	6773	8100	0.10071	19.5924	8345	8774	9245	1.69813	23.2090	29.547
p654	654	34643	43492	0.11859	25.5433	43457	49486	54001	2.27642	25.4426	42.845
fl1400	1400	20127	26461	0.87500	31.4701	26854	28935	31599	30.2443	33.4226	43.761
d2103	2103	80529	88547	5.34190	9.95666	86554	93753	99944	249.528	7.48176	16.424

In Table 1, we show the error (%) comparison between the improved NNA and the basic NNA with the best-known solution. The error (%) of the best result in improved NNA is lower than the average in all 20 cities listed above, and the worst result is in basic NNA. For example, the dataset "rat99" represents the 99 nodes of this city. By using the improved NNA, the optimal is 1577, and the average result of that dataset is 1695 for the basic NNA. The error in improved NNA is 30.2229%, whereas the error in basic NNA is 24.4426% and 39.969% for the best and average results, respectively. Again, for "lin318," using our improved NNA, the optimal solution is 52883. Using the basic NNA, the average result is 56328. The error in improved NNA is 20.7524%, whereas the error in basic NNA is 37.4256% and 47.582% for the best and average results, respectively. Even for the higher number of cities, the same characteristics hold. For example, the dataset "d2103" denotes the number of nodes as 2103. This is a huge number of nodes. By using an improved NNA, the optimal solution for that dataset is 88547. Using the basic NNA, the average result is 93753. The error in improved NNA is 9.95666%, whereas the error in basic NNA is 7.48176% and 16.424% for the best and average results, respectively.

Similarly, for all other datasets. The fourth and fifth columns of Table 1 show a comparison of the computational time between improved NNA and basic NNA for any random dataset. The required time for execution of the improved NNA is less than the basic NNA. For example, using improved NNA, the time of execution of "kroC100" is 0.009948 s, and for basic NNA, it is 0.069040s. Also, by using improved NNA, the time of execution of the dataset "ch130" is 0.017507s, and for basic NNA, it is 0.080252s. Even for the higher number of cities, the same characteristics hold. For dataset 'fl1400', the execution time is 0.875745s for improved NNA and 30.244633s for basic NNA. Improved NNA, on the other hand, outperforms basic NNA in terms of computational time. According to the simulation results, improved NNA outperforms both the average and the worst results of basic NNA in terms of values and computational time.

Even with respect to the best-known solution of datasets, the error (%) of some cities in improved NNA is less than the error (%) of the best result in basic NNA. In improved NNA, the optimal solution of 8 datasets provides a better result than the best result of basic NNA. For the dataset "kroC100," the optimal solution is 25519, whereas the best solution for the basic NNA is 26043. For this dataset, the error in the improved NNA is 22.9890%, whereas the error in the basic NNA is 25.5145% for the best result with respect to the best-known solutions to the dataset. Again for "rat195", the optimal solution for improved NNA is 8100, where the best solution for basic NNA is 8345. For this dataset, the error in the improved NNA is 12.9573%, whereas the error in the basic NNA is 18.4245%. Likewise for the other datasets. Therefore, the improved NNA provides better results than the basic NNA, both in the aspect of values and of computational time and reduces error. In Fig. 2 the bar chart shows the error (%) comparison between the improved NNA and the basic NNA with respect to the best-known solutions of 20 symmetric datasets. The colors blue, orange, and yellow represent the best solution error for improved NNA, the best solution error for basic NNA, and the best solution error for basic NNA, respectively.

The bar chart (Fig. 3) shows the error (%) comparison between improved NNA and basic NNA for the 8 best symmetric datasets, which is best against basic NNA. For further visualization, Fig. 4 graphically represents the difference between the improved NNA and the basic NNA of the Best 8 dataset. There are two figures for each dataset in Fig. 4: one for improved NNA and another for basic NNA. The route to an optimal solution is shown in the two figures. Comparing table, graph, and bar charts, it is demonstrated that the improved NNA outperforms the basic NNA in terms of solution quality as well as computational time.

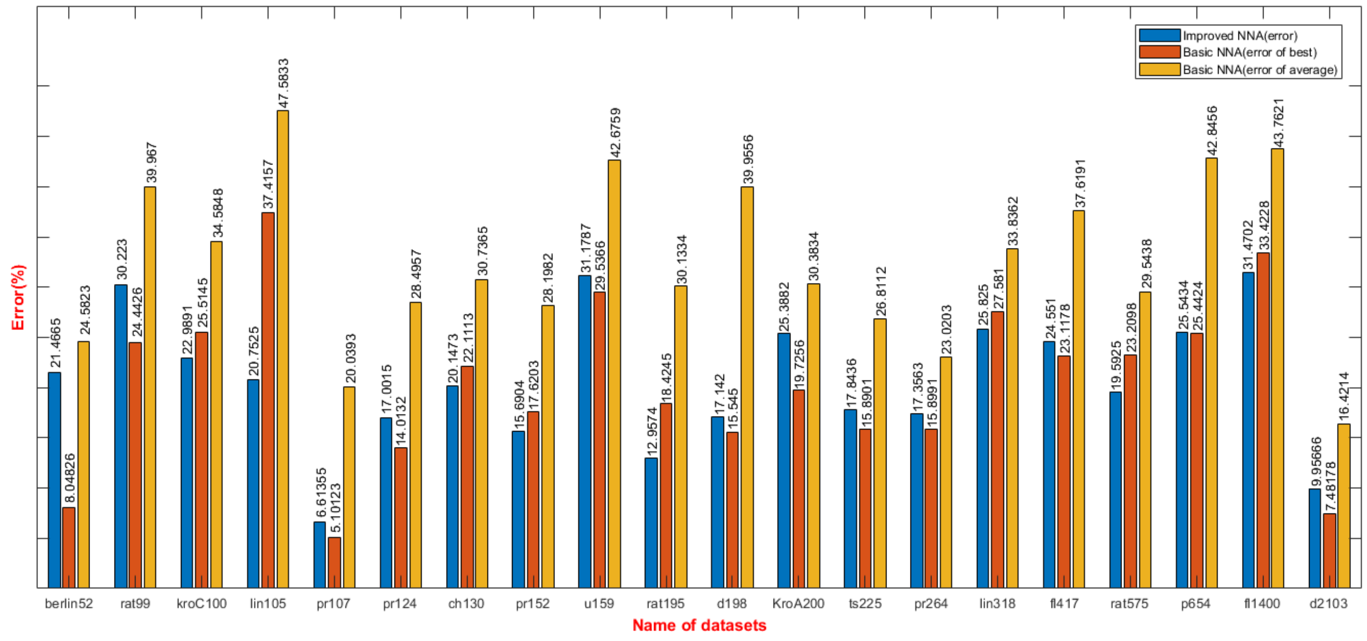

Fig. 2 Error (%) comparison bar charts for the best solution of improved NNA and the average and best solution of basic NNA with respect to the best-known solution for 20 benchmark STSP datasets.

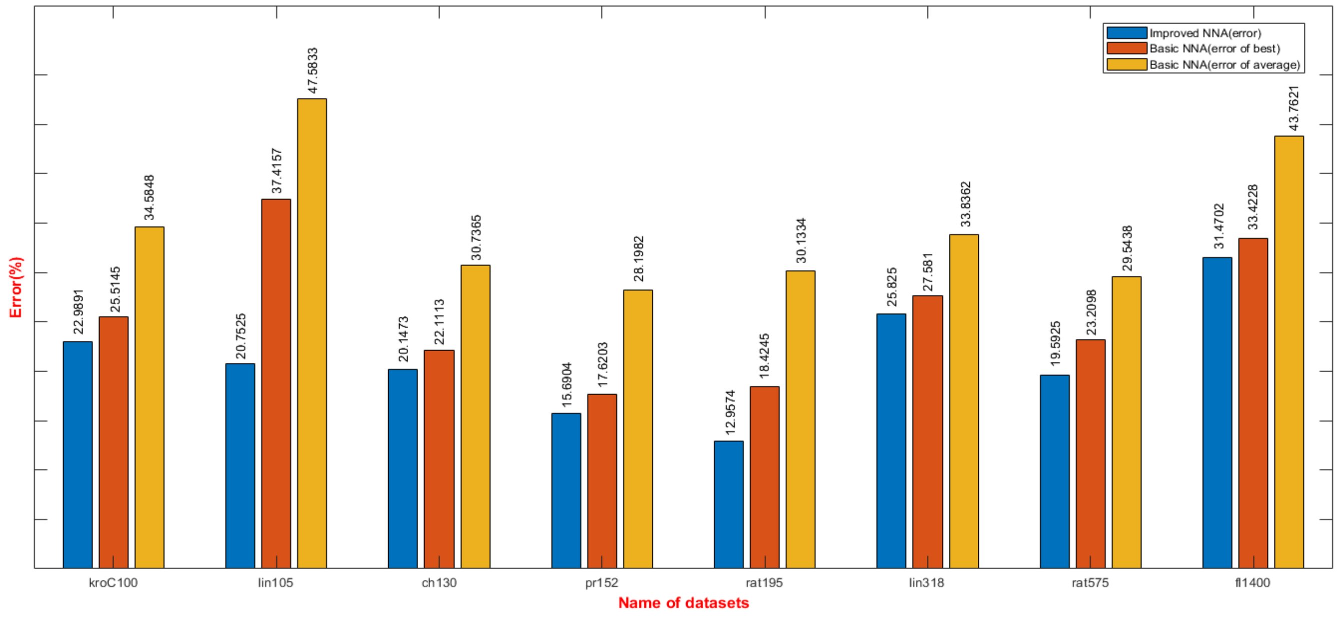

Fig. 3 Error comparison bar charts of improved NNA and basic NNA for best 8 benchmark STSP datasets.

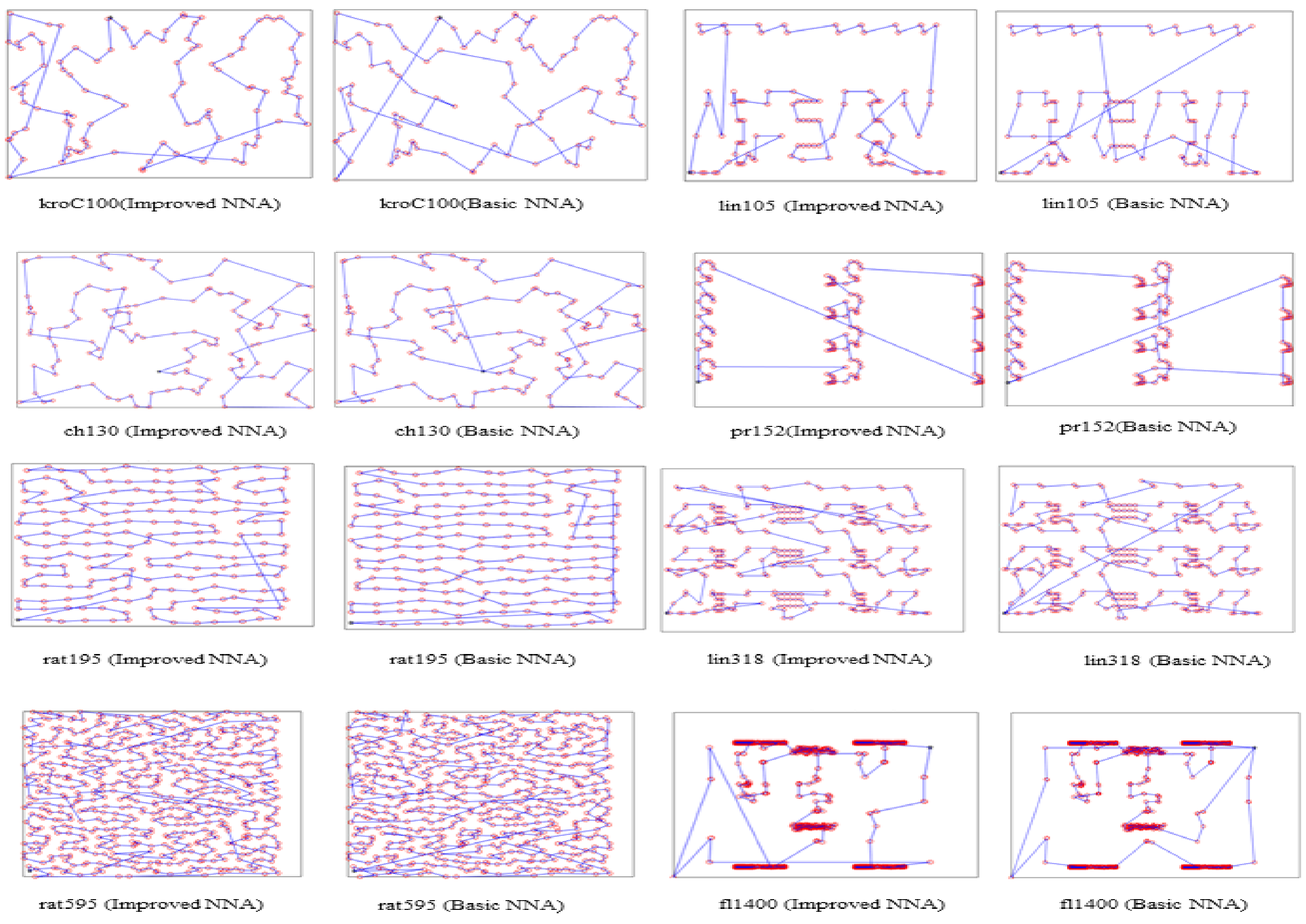

Fig. 4 Obtained optimum route for 8 datasets both of improved NNA and basic NNA

6 Conclusions

The present study improved the basic NNA for solving symmetric TSP. It compared improved NNA with basic NNA because both algorithms are route construction algorithms. However, the main difference between them is that the improved NNA is deterministic, i.e., the results produced by this algorithm are same at every single run. On the other hand, the basic NNA is probabilistic, i.e., this algorithm generates different results at every single run. All datasets have been subjected to comprehensive testing. The simulation findings and subsequent analysis demonstrate that the improved NNA yields superior outcomes compared to the basic NNA. It has been discovered that when the initial solution is selected randomly, the improved NNA outperforms and exhibits more efficiency compared to the basic NNA. Based on the aforementioned datasets, simulation design, and environmental conditions, it can be inferred that improved NNA yields superior outcomes with reduced time consumption. In subsequent analyses, the improved NNA will be juxtaposed with other route construction algorithms. Furthermore, it is possible to develop a hybrid method by integrating the improved NNA with other route construction algorithms. As a result, it will achieving superior outcomes compared to an improved NNA.

References

[1] Rahman, M.A. and Parvez, H., 2021. Repetitive nearest neighbor based simulated annealing search optimization algorithm for traveling salesman problem. *Open Access Library Journal*, *8*(6), pp.1-17.

[2] Applegate, D., Bixby, R., Cook, W. and Chvátal, V., 1998. On the solution of traveling salesman problems.

[3] Dantzig, G., Fulkerson, R. and Johnson, S., 1954. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4), pp.393-410.

[4] Deng, W., Chen, R., He, B., Liu, Y., Yin, L. and Guo, J., 2012. A novel two-stage hybrid swarm intelligence optimization algorithm and application. *Soft Computing*, *16*, pp.1707-1722.

[5] Hore, S., Chatterjee, A. and Dewanji, A., 2018. Improving variable neighborhood search to solve the traveling salesman problem. *Applied Soft Computing*, *68*, pp.83-91.

[6] Matai, R., Singh, S.P. and Mittal, M.L., 2010. Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling Salesman Problem, Theory and Applications*, *1*(1), pp.1-25.

[7] Naser, H., Awad, W.S. and El-Alfy, E.S.M., 2019. A multi-matching approximation algorithm for Symmetric Traveling Salesman Problem. *Journal of Intelligent & Fuzzy Systems*, *36*(3), pp.2285-2295.

[8] Halim, A.H. and Ismail, I., 2019. Combinatorial optimization: comparison of heuristic algorithms in

travelling salesman problem. *Archives of Computational Methods in Engineering, 26*, pp.367-380.

[9] Bentley, J.J., 1992. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4), pp.387-411.

[10] Klug, N., Chauhan, A., V, V. and Ragala, R., 2019. k-RNN: Extending NN-heuristics for the TSP. *Mobile Networks and Applications, 24*, pp.1210-1213.

[11] Bakar, S.A. and Ibrahim, M., 2017, August. Optimal solution for travelling salesman problem using heuristic shortest path algorithm with imprecise arc length. In *AIP Conference Proceedings*, *1870*(1). AIP Publishing.

[12] Lin, S. and Kernighan, B.W., 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21*(2), pp.498-516.

[13] Chen, Y. and Zhang, P., 2006. Optimized annealing of traveling salesman problem from the nth-nearest-neighbor distribution. *Physica A: Statistical Mechanics and Its Applications, 371*(2), pp.627-632.

[14] Raya, L., Saud, S.N., Shariff, S.H. and Bakar, K.N.A., 2020. Exploring the performance of the improved nearest-neighbor algorithms for solving the euclidean travelling salesman problem. *Advances in Natural and Applied Sciences, 14*(2), pp.10-19.

[15] Rosenkrantz, D.J., Stearns, R.E. and Lewis, II, P.M., 1977. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing, 6*(3), pp.563-581.

[16] Sahin, M., 2023. Solving TSP by using combinatorial Bees algorithm with nearest neighbor method. *Neural Computing and Applications, 35*(2), pp.1863-1879.

[17] Pop, P.C., Cosma, O., Sabo, C. and Sitar, C.P., 2023. A comprehensive survey on the generalized traveling salesman problem. *European Journal of Operational Research, 314*(3), pp.819-835.

[18] Gutin, G., Yeo, A. and Zverovitch, A., 2002. Exponential neighborhoods and domination analysis for the TSP. In *The Traveling Salesman Problem and Its Variations* (pp. 223-256). Boston, MA: Springer US.