On the Nearest Neighbor Algorithms for the Traveling Salesman Problem

Gözde Kizilateş and Fidan Nuriyeva

Department of Mathematics, Faculty of Science, Ege University, Izmir, Turkey
{gozde.kizilates,nuriyevafidan}@gmail.com

Abstract. In this study, a modification of the nearest neighbor algorithm (NND) for the traveling salesman problem (TSP) is researched. NN and NND algorithms are applied to different instances starting with each of the vertices, then the performance of the algorithm according to each vertex is examined. NNDG algorithm which is a hybrid of NND algorithm and Greedy algorithm is proposed considering experimental results, and it is tested on different library instances.

Keywords: traveling salesman problem, hybrid algorithm, nearest neighbor algorithm, greedy algorithm.

1 Introduction

The Traveling Salesman Problem (TSP) is the most famous optimization problem in the set NP-hard [2, 6]. TSP aims to find a route for a salesman who starts from a home location, visits prescribed set of cities and returns to the original location in such a way that the total traveling distance will be minimum and each city will have been visited exactly once [3]. Many problems that are natural applications in computer science and engineering can be modeled using TSP [4, 8, 9, 14].

The algorithms for solving TSP can be divided into four classes: exact algorithms, heuristic algorithms, approximate algorithms and metaheuristics algorithms [7]. The exact algorithms are guaranteed to find the optimal solution. However, these algorithms are quite complex and very demanding of computer power [1]. The heuristic algorithms can obtain good solutions but it cannot be guarantee that the optimal solution will be found [5]. In general, the heuristic algorithms are subdivided into the following three classes: tour construction algorithms, tour improvement algorithms and hybrid algorithms [15]. The tour construction algorithms gradually build a tour by adding a new city at each step, the tour improvement algorithms improve upon a tour by performing various exchanges, and finally hybrid algorithms use both composing and improving heuristics at the same time [10-13]. The best results are obtained by using hybrid approaches [1, 15]. This paper presents a hybrid construction algorithm for solving TSP based on the nearest neighbor algorithm and the greedy algorithm.

D. Nagamalai et al. (Eds.): *Adv. in Comput. Sci., Eng. & Inf. Technol.*, AISC 225, pp. 111–118.
DOI: 10.1007/978-3-319-00951-3_11

2 The Nearest Neighbor Algorithm (NN)

The nearest neighbor (NN) algorithm for determining a traveling salesman tour is as follows. The salesman starts at a city, then visits the city nearest to the starting city. Afterwards, he visits the nearest unvisited city, and repeats this process until he has visited all the cities, in the end, he returns to the starting city.

The steps of the algorithm are as following [1]:

1. Select a random city.
2. Find the nearest unvisited city and go there.
3. Are there any unvisited cities left? If yes, go to step 2.
4. Return to the first city.

We can obtain the best result by running the algorithm over again for each vertex and repeat it for *n* times.

The next theorem, due to Rosenkrantz, Stearns and Lewis [16], shows that this approximation algorithm may have much worse behavior.

Theorem: For all m-city instances I of the traveling salesman problem with triangle inequality,

$$\mathsf{NN(I)} \le \frac{1}{2}\left(\lceil \log_2 m \rceil + 1\right)\mathsf{OPT(I)}$$

Furthermore, for arbitrarily large values of m, there exist m-city instances for which

$$\mathsf{NN(I)} > \frac{1}{3}\left(\log_2 (m+1)\frac{4}{3}\right)\mathsf{OPT(I)}$$

The main import of this theorem can be stated quite succinctly: $R_{NN} = \infty,$ it is not a promising guarantee. So that, $m \to \infty \Rightarrow NN(I) \to \infty$.

The performance of the Nearest Neighbor algorithm clearly leaves much to be desired. A different modification of this algorithm is proposd below. It is interesting to find similar guarantee values with the above theorem for this modification. The NN algorithm is not promising theoretically; however, it yields good results in practice.

3 The Nearest Neighbor Algorithm (NND) from Both End-Points

The algorithm starts with a vertex chosen randomly in the graph. Then, the algorithm continues with the nearest unvisited vertex to this vertex. We will have two end vertices. We add a vertex to the tour such that this vertex has not visited before and it is the nearest vertex to these two end vertices. We update the end vertices. The algorithm ends after visiting all vertices.

The steps of the algorithm are as following:

1. Choose an arbitrary vertex in the graph
2. Visit the nearest unvisited vertex to this vertex.
3. Visit the nearest unvisited vertex to these two vertices and update the end vertices
4. Is there any unvisited vertex left? If yes, go to step 3.
5. Go to the end vertex from the other end vertex.

4 Computational Tests for the NND Algorithm

NN and NND algorithms have been tested on well-known library problems. Table 1 and Figure 1 shows the experimental results for ulysses16 given in [17, 18].

Table 1. Comparison of NN and NND in terms of row-sum for ulysses16

Vertex index	Row-sum	NN	NND
1	111,039167	104,73494	84,427643
2	175,806969	85,46698	84,427643
3	168,175063	84,427643	84,427643
4	142,055212	84,427635	84,427643
5	144,262526	86,786415	86,786415
6	106,548756	88,588806	86,786415
7	99,585345	95,366249	86,786415
8	111,970472	104,076958	84,427643
9	147,71774	77,126892	84,269722
10	114,327834	84,652298	84,269722
11	334,310966	77,126884	77,126884
12	91,810072	96,719711	86,786415
13	91,23502	97,199722	86,786415
14	92,118889	92,334679	86,786415
15	104,784133	95,78952	86,786415
16	108,067343	84,433632	84,427643

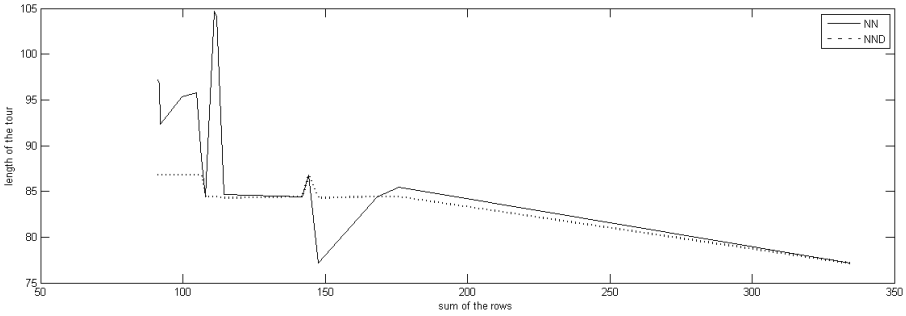

Fig. 1. Comparison of NN and NND in terms of row-sum for ulysses16

As it can be seen in Table 1 and Figure 1 above, the NND algorithm generally gives better results.

5 Greedy Algorithm

The Greedy heuristic gradually constructs a tour by repeatedly selecting the shortest edge and adding it to the tour as long as it doesn't create a cycle with less than *N edges, or increase the degree of any node by* more than 2. We must not add the same edge twice.

The steps of the algorithm are as following [1]:

1. Sort edges by increasing lengths.
2. Select the shortest edge and add it to our tour if it doesn't violate any of the above constraints.
3. Do we have *n* edges in our tour? If no, go to step 2.

6 An Hybrid of NND and Greedy Algorithms (NNDG)

After we create the adjacency matrix for the given problem, we compute the sum of the entries in each row, it is called row-sum here. We continue by finding the vertex which has the smallest row-sum. Then, we apply the NND algorithm proposed above for the vertex that has been found. After applying the NND algorithm, we continue the process by repeating the NND algorithm starting with the last vertex in the tour until we coincide with any vertex which has been added to the tour before. Then, we calculate $t = \lfloor n/k \rfloor$, where n is the number of vertices, and k is the number of implementations of NND. The first t edges are taken respectively from each solution found by NND and the edges which do not form subtours are added to our solution. The algorithm ends by applying Greedy algorithm to the other edges.

The steps of the algorithm are as following:

1. Find the vertex which has the smallest row-sum.
2. Apply NND by starting this vertex.
3. Continue to apply NND starting with the last vertex in the tour until you coincide with any vertex which has been added to the tour before.
4. Calculate $t = \lfloor n/k \rfloor$, where n is the number of vertices, k is the number of implementation of NND.
5. Take first t edges respectively from each solution found by NND and add the edges, which don't form subtours, to the solution
6. Apply Greedy algorithm to the other edges.

7 Computational Tests for NNDG Algorithms

We test the NNDG algorithm on different well-known library problems. Table 2 and Figure 2 shows the experimental results for ulysses22 given in [17, 18].

In these tests, the distance from each vertex to other vertices is computed by summing all entries in the row of the adjacency matrix. The relation between these distances and the results of the algorithm is examined in Table 2 and Figure 2.

Table 2. The results for ulysses22 obtained by choosing a starting vertex with respect to the row-sum

Row-sum	Vertex index	Tour length
123,2764	13	83.018
123,4718	12	82.861
126,0229	14	82.875
133,8718	21	87.166
133,9167	20	87.166
136,9844	7	87.166
138,2799	16	86.899
138,9798	19	87.166
140,9329	1	86.899
142,1713	8	86.899
147,0848	15	87.166
150,0691	6	87.166
151,096	10	87.166
163,0422	22	87.166
175,6999	18	87.166
175,9675	4	87.166
188,9961	17	87.166
206,9365	9	87.166
207,4082	5	87.166
213,945	3	79.671
223,5139	2	87.166
478,9774	11	87.166

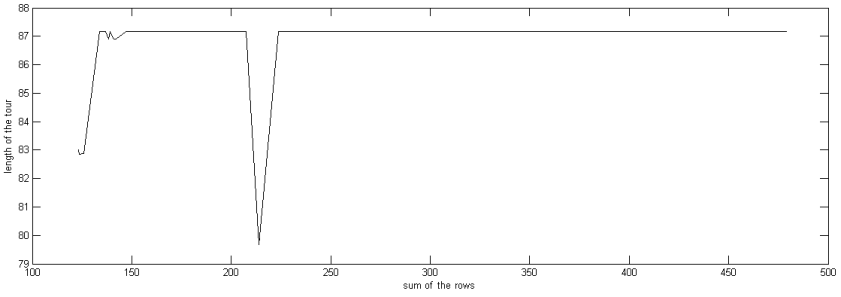

Fig. 2. The graph of the results for ulysses22 obtained by choosing a starting vertex with respect to the row-sum

According to Table 2 and Figure 2, it is seen that better results are obtained when starting from the vertices which have smaller sums of distances.

In Table 3, the experimental results obtained for 11 different values of *t* parameter on 5 problems in [17, 18] are given.

Table 3. The results depending on different values of t

graph	ulyses16	ulysses22	eil51	berlin52	eil76
optimum	74.108	75.665	429.983	7544.365	545.387
k	3	3	3	3	6
$t = \lfloor n/k \rfloor - 5$	77.126	79.671	521.961	8542.874	620.270
$t = \lfloor n/k \rfloor - 4$	77.126	87.166	521.961	8542.874	620.270
$t = \lfloor n/k \rfloor - 3$	77.126	87.166	521.961	8542.874	620.270
$t = \lfloor n/k \rfloor - 2$	77.126	87.166	513.562	8542.874	620.270
$t = \lfloor n/k \rfloor - 1$	77.126	85.501	513.562	8542.874	620.270
$t = \lfloor n/k \rfloor$	77.126	83.018	513.562	8542.874	620.270
$t = \lfloor n/k \rfloor + 1$	77.126	83.018	513.562	8542.874	620.270
$t = \lfloor n/k \rfloor + 2$	77.126	83.018	513.562	8542.874	620.270
$t = \lfloor n/k \rfloor + 3$	77.126	83.018	513.562	8542.874	620.270
$t = \lfloor n/k \rfloor + 4$	77.126	87.166	530.621	8542.874	620.270
$t = \lfloor n/k \rfloor + 5$	77.126	87.166	530.621	8542.874	620.270

According to Table 3, the best results are obtained for $t = \lfloor n/k \rfloor$.

In Table 4, the computational results which are obtained by the implementation of the NNDG algorithm for 5 different origin vertices, are given for 27 problems in [17, 18].

Table 4. Computational results for different problems and different origin vertices

Problem	Optimum	The vertex which has smallest row-sum	$\lfloor n/4 \rfloor$ th vertex according to row-sum	$\lfloor n/2 \rfloor$ th vertex according to row-sum	$\lfloor 3n/4 \rfloor$ th vertex according to row-sum	The vertex which has biggest row-sum
ulysses16	74.108	77.126	77.126	77.126	77.126	77.126
ulysses22	75.665	83.018	87.166	87.166	87.166	87.166
bayg29	9074.148	10810.481	10810.481	9963.589	10810.481	10810.481
att48	33523.708	40068.035	40068.035	40068.03	40068.035	41182.40
eil51	429.983	513.562	527.034	524.255	539.811	530.370
berlin52	7544.365	8542.874	8542.874	8542.874	8542.874	8542.874
st70	678.597	797.815	778.911	778.911	797.815	797.815
eil76	545.387	620.270	519.751	620.270	605.206	620.270
pr76	108159.43	134284.812	139582.203	139582.203	139582.203	139582.203
rd100	7910.396	9900.226	9591.957	9474.017	9474.017	9516.281
kroA100	21236.951	27008.218	27008.218	27008.218	26261.531	26264.730
kroB100	22141	26383.210	26383.210	26383.210	26383.210	26383.210
kroC100	20750.762	23401.554	24761.794	25193.234	25759.095	25388.687
kroD100	21294.290	25752.627	27006.322	26193.382	23729.011	26600.332
kroE100	22068	26288.539	26288.539	26288.539	26288.539	26288.539
eil101	642.309	771.399	771.399	771.399	771.399	771.399
lin105	14382.995	17362.769	17067.777	17362.769	17362.769	17362.769
pr107	44303	46872.058	46872.058	46872.058	46872.058	46617.173
gr120	1666.508	2002.396	2002.396	1993.938	2002.396	1877.776
pr124	59030	67344.921	65940.471	67344.921	67344.921	67344.921
ch130	6110.860	7255.568	7255.568	7255.568	7017.948	7255.568
pr136	96772	118856.343	118856.343	118856.343	115966.906	118856.343
pr144	58537	62543.738	62543.738	62543.738	62543.738	62543.738
ch150	6528	7028.432	7028.432	7028.432	7028.432	7028.432
kroA150	26524	33104.113	33104.113	33104.113	33104.113	33104.113
kroB150	26130	32176.169	32176.169	32176.169	32176.169	32017.155
pr152	73682	84780.023	84780.023	84780.023	84780.023	84780.023

As it is seen from the table above, the results are also better when we start from the vertices which have smaller distances.

8 Conclusion

In this study, a new modification (NND) of the NN algorithm has been proposed for solving TSP. Computational experiments have been conducted on known library problems for this algorithm. Moreover, we have presented a new hybrid algorithm

(NNDG) based on NND (which is presented for improving the results) and Greedy algorithms. Computational experiments are conducted on known library problems for this algorithm, as well. These experiments have shown that the proposed algorithm (NNDG) is efficient. We are planning to improve the proposed hybrid algorithm in future works.

References

1. Appligate, D.L., Bixby, R.E., Chavatal, V., Cook, W.J.: The Travelling Salesman Problem, A Computational Study. Princeton Univesity Press, Princeton (2006)
2. Davendra, D.: Travelling Salesman Problem, Theory and Applications. InTech (2010)
3. Gutin, G., Punnen, A. (eds.): The Traveling Salesman Problem and Its Variations. Combinatorial Optimization, vol. 12. Kluwer, Dordrecht (2002)
4. Hubert, L.J., Baker, F.B.: Applications of Combinatorial Programming to Data Analysis: The Traveling Salesman and Related Problems. Psychometrika 43(1), 81–91 (1978)
5. Johnson, D., Papadimitriou, C.: Performance guarantees for heuristics. In: Lawler, et al. (eds.), ch. 5, pp. 145–180 (1985)
6. Johnson, D.S., McGeoch, L.A.: The Traveling Salesman Problem: A Case Study, Local Search in Combinatorial Optimization, pp. 215–310. John Wiley & Sons (1997)
7. Lawler, E.L., Lenstra, J.K., Rinnoy Kan, A.H.G., Shmoys, D.B.: The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization. John Wiley & Sons (1986)
8. Lenstra, J., Kan, A.R.: Some simple applications of the travelling salesman problem. Operational Research Quarterly 26(4), 717–733 (1975)
9. Lenstra, J.K.: Clustering a Data Array and the Traveling-Salesman Problem. Operations Research 22(2), 413–414 (1974)
10. Lin, S., Kernighan, B.: An effective heuristic algorithm for the traveling-salesman problem. Operations Research 21(2), 498–516 (1973)
11. Nuriyeva, F.: New heuristic algorithms for travelling salesman problem. In: 25th Conference of European Chapter on Combinatorial Optimization (ECCO XXV), Antalya, Turkey, April 26-28 (2012)
12. Nuriyeva, F., Kizilates, G., Berberler, M.E.: Experimental Analysis of New Heuristics for the TSP. In: IV International Conference “Problems of Cybernetics and Informatics, Baku, Azerbaijan, September 12-14 (2012)
13. Nuriyeva, F., Kızılateş, G.: A New Hyperheuristic Algorithm for Solving Traveling Salesman Problem. In: 2nd World Conference on Soft Computing, Baku, Azerbaijan, December 3-5, pp. 528–531 (2012)
14. Punnen, A.: The Traveling Salesman Problem: Applications, Formulations and Variations. In: Gutin, Punnen (eds.), ch. 1, pp. 1–28 (2002)
15. Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications. Springer, Germany (1994)
16. Rosenkrantz, D.J., Stearns, R.E., Lewis, P.M.: An Analysis of SeveralHeuristicsfortheTravellingSalesman Problem. SIAM J. Comput. 6, 563–581 (6.1)
17. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
18. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/