Journal of Modern Technology and Engineering
Vol.6, No.1, 2021, pp.34-40

Jomard Publishing

AN ITERATIVE HEURISTIC ALGORITHM FOR TRAVELLING SALESMAN PROBLEM

Eshref Alizade[1], **Fidan Nuriyeva**[2,3*]

[1]Istanbul Technical University, Faculty of Management, Department of Industrial Engineering, Istanbul, Turkey
[2]Dokuz Eylul University, Faculty of Science, Department of Computer Science, Izmir, Turkey
[3]Institute of Control Systems of ANAS, Baku, Azerbaijan

Abstract. The Traveling Salesman Problem is the best-studied NP-hard combinatorial optimization problem. The problem plays an important role in the literature due to a large number of application areas and there are many real-life problems that can be formulated as Traveling Salesman Problem. In this paper, a new iterative algorithm is proposed for the Symmetric Traveling Salesman Problem. The proposed algorithm is coded in C ++ language and tested on library samples and compared with many other well-known algorithms. Experimental outcomes show that better results are obtained with the proposed algorithm.

Keywords: travelling salesman problem, heuristic algorithms, iterative algorithm, greedy algorithm, priority degree.

AMS Subject Classification: 90C27, 90C59.

Corresponding author: Fidan, Nuriyeva, Dokuz Eylul University, Faculty of Science, Department of Computer Science, Izmir, Turkey, e-mail: *fidan.nuriyeva@deu.edu.tr, nuriyevafidan@gmail.com*

Received: 12 February 2021; Revised: 21 March 2021; Accepted: 4 April 2021; Published: 30 April 2021.

1 Introduction

Traveling Salesman Problem (TSP) is a combinatorial optimization problem studied in the field of operations research (Davendra, 2010; Punnen, 2002). The aim of TSP is to find a route for a salesman who starts from a home location, visits a prescribed set of cities and returns to the original location in such a way that the total distance traveled is minimum and each city is visited exactly once (Davendra, 2010). As a graph problem TSP can be modeled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and distances between the cities is the edge's weight. TSP is a minimization problem starting and finishing at a specified vertex after having visiting each other vertex exactly once (Applegate et al., 2007).

The TSP is classified as symmetric travelling salesman problem (sTSP), asymmetric travelling salesman problem (aTSP), and multiple travelling salesman problem (mTSP). In the symmetric TSP, the distance between two cities are the same in each opposite direction, forming an undirected graph. In the asymmetric TSP, paths may not exist in both directions or the distances may be different (Gutin & Punnen, 2007; Punnen, 2002).

The TSP has several applications, such as vehicle routing, drilling of printed circuit boards, overhauling gas turbine engines, x-ray crystallography, computer wiring and etc (Davendra, 2010; Hubert & Baker, 1978; Lenstra & Kan , 1975; Punnen, 2002; Reinelt, 1994).

Travelling Salesman Problem is an NP-hard problem (Applegate et al., 2007). The difficulty becomes apparent when one considers the number of possible tours, an astronomical figure even

for a relatively small number of cities (Reinelt, 1994). The Heuristic Algorithms, Metaheuristic Algorithms, Approximate Algorithms and Exact Algorithms are some approaches that were developed for solving TSP. The best known exact methods for solving TSP are: brute force search, branch and bound method, dynamic programming and etc. (Applegate et al., 2007; Reinelt, 1994). But these approaches are expensive to calculate and takes a long time for the number of cities greater than 20.

Solving the TSP optimally takes too long, instead one normally uses approximation algorithms, or heuristics (Aarts & Lenstra, 1997). The best-known approximate algorithms for TSP are Christofides Algorithm (guaranteed value 3/2), Minimum-Spanning Tree (MST) based algorithms (guaranteed value 1/2), and others (Johnson & McGeoch, 1997). Heuristic Algorithms give no guarantee to find the optimal solution to the given problem, but in many cases, find a solution of good quality and obtain it in an acceptable time.

Heuristic algorithms for TSP can be classified into three groups: heuristics composing the tour, heuristics improving the tour and hybrid heuristics using both (Kızılateş & Nuriyeva, 2013; Nuriyev et al., 2018).

In this paper, a new heuristic iterative algorithm preented for the TSP.

The remainder of the paper is structured as follows. In section 2 we outline the well-known heuristics to solve the TSP. Section 3 presents an algorithm for the TSP based on the greedy heuristic. Section 4 describes experimental results. We conclude in Section 5 with final remarks.

2 Heuristics for TSP

There are various heuristics and approximate solution approaches, which had been devised during last decades, to find solution within reasonable time. This section presents two of them.

2.1 Nearest Neighbour Algorithm

The Nearest Neighbour (NN) algorithm was one of the first algorithms to find a solution for the travelling salesman problem (Johnson & McGeoch, 1997). In this algorithm, the salesman starts in a random city and repeatedly visits the nearest city until all cities are visited. The algorithm stops when all cities are on the tour. It finds a shortest path, but solution is not the optimal one (Kızılateş & Nuriyeva, 2013).

The steps of the algorithm are as follows:
Step 1. Start in any random city as current city.
Step 2. Find out the shortest path connecting current city and an unvisited city.
Step 3. Set current city as visited city and mark it as visited.
Step 4. If all the cities are visited then terminate. Else go to step 2.

2.2 The Greedy Algorithm

The Greedy heuristic begins with the edge of least weight and makes it part of the circuit. Then it selects the edge of second-smallest weight, and so on. Once a vertex has two selected edges, no more edges of that vertex are considered and we must avoid creating a circuit prematurely (Johnson & McGeoch, 1997) The steps of the algorithm are as follows:
Step 1. Pick the edge with the smallest weight available.
Step 2. Pick the next edge with the smallest weight available and mark it.
Step 3. Continue picking and marking the edge with the smallest weight that does not close a circuit or create three edges coming out of a single vertex.
Step 4. Connect the last two vertices to close the circuit.

3 Proposed Algorithm

Let TSP defined as a graph model given below:

$$V = \{v_1, v_2, \ldots, v_n\}, E = \{e_{12}, e_{13}, \ldots, e_{1n};\ e_{21}, e_{23}, \ldots, e_{2n};\ e_{n1}, e_{n2}, \ldots, e_{nn-1}\}.$$

Here, V is the set of vertices (cities), E is the set of edges (distances) between the cities, n is the number of vertices, m is the number of edges.

So our graph may not also be symmetrical, when it is symmetrical then $e_{ij} = e_{ji}$.

$$|V| = n, \quad |E| = m = n \times (n-1)$$

The distance between the vertices v_i and v_j is denoted by d_{ij}.

Distances between the vertices is given in adjacency matrix given below:

Vertices	v_i	v_i	...	v_n	MİN	MAX
v_1	0	d_{12}	...	d_{1n}	d_1^-	d_1^+
v_2	d_{21}	0	...	d_{2n}	d_2^-	d_2^+
...	...	...	...	...	...	...
v_n	d_{n1}	d_{n2}	...	0	d_n^-	d_n^+

$$d_i^- = min_j \{d_{ij}\}, \quad i = 1, 2, \ldots, n;$$

$$d^- = max_i \left\{d_i^-\right\};$$

$$d_i^+ = max_j \{d_{ij}\}, \quad i = 1, 2, \ldots, n;$$

$$\alpha_i = \frac{d_i^-}{d^-}, \quad i = 1, 2, \ldots, n;$$

Priority degrees of edges shown with (u_{ij}) are calculated with the formula given below:

$$u_{ij} = \alpha_i \cdot \frac{d_i^+}{d_{ij}}; \quad i, j = 1, 2, \ldots, n \tag{1}$$

We proposed above a heuristic approach to determine the Priority Level. Now we briefly explain this approach.

1. Firstly for each vertex we determine the edges with the smallest weight incident to this vertex: $d_i^- = min_j \{d_{ij}\}$, $i = 1, 2, \ldots, n$;

2. Then the edge with the largest weight among these edges is determined: $d^- = max_i \left\{d_i^-\right\}$,

3. To determine the priority degree of the edges, the ratio of the weight of the edge with smallest weight incident to the vertex which is incident to the edge under consideration, to the maximum of the minimum weights of the edges incident to other vertices is taken into account and the edge with the higher ratio has the priority.

   This allows to select the vertices from "bads" to "goods":

   $\alpha_i = \frac{d_i^-}{d^-}$, $i = 1, 2, \ldots, n$;

4. The edge with the largest weight is determined for each vertex: $d_i^+ = max_j \{d_{ij}\}$, $i = 1, 2, \ldots, n$;

5. To determine the priority degree of each edge, the ratio of the weight of the edge with the largest weight, incident to the vertex which is incident to the edge under consideration, to the weight of the edge under consideration is taken into account and the edge with higher one is given priority. This allows to order the edges from "bads" to "goods":

$$u_{ij} = \alpha_i \cdot \frac{d_i^+}{d_{ij}}; \quad i, j = 1, 2, \ldots, n$$

Thus the proposed approach first orders vertices from "bads" to "goods" and then for each vertex the edges are chosen by ordering from "bads" to "goods".
The found best solution is denoted by X, and the proper value of the objective function is denoted by $F(X)$.

3.1 NI Algorithm

Steps of the proposed algorithm are as follows:
NI-1: $k = 0$, $u_{ij}^{(0)}$, $\left(i, j = \overline{1, n}\right)$ is calculated with the formula (1).
NI-2: Edges sorted in descending order of $u_{ij}^{(0)}$ $\left(i, j = \overline{1, n}\right)$.
If there exist $u_{ij}^{(0)}$ $\left(i, j = \overline{1, n}\right)$ with equal values, corresponding values of d_{ij} 's are compared and the d_{ij} with a smaller value comes first. If still there are equalities, then $u_{ij}^{(0)}$ $\left(i, j = \overline{1, n}\right)$ with bigger first index comes first.

$$u_{i_1 j_1}^{(0)} \geq u_{i_2 j_2}^{(0)} \geq \cdots \geq u_{i_m j_m}^{(0)} \tag{2}$$

NI-3: $\beta_0 = \left(u_{i_1 j_1}^{(0)} - u_{i_m j_m}^{(0)}\right)/m$
NI-4: The initial greedy solution is found according to the sorting of the formula (2): X^0, F^0

$$X = X^0, F = F^0, t^{(0)} = 1$$

NI-5: $k = k + 1$
NI-6: If $k > m$ goto **NI-14**
NI-7: Priority values $u_{ij}^{(k)}, (i, j = 1, 2, \ldots n)$ is updated as follows:

$$u_{ij}^{(k)} = \begin{cases} u_{ij}^{(k-1)} - \beta_{k-1} \cdot t^{(k-1)}, & if x_{ij}^{(k-1)} = 1 \\ u_{ij}^{(k-1)}, & \text{otherwise} \end{cases}$$

NI-8: Edges are sorted with descending order of $u_{ij}^{(k)}$ $(i, j = \overline{1, n})$:

$$u_{i_1 j_1}^{(k)} \geq u_{i_2 j_2}^{(k)} \geq \cdots \geq u_{i_m j_m}^{(k)} \tag{3}$$

NI-9: $\beta_k = \left(u_{i_1 j_1}^{(k)} - u_{i_m j_m}^{(k)}\right)/(k+1)$
NI-10: The greedy solution in k-th iteration $(X^{(k)}, F^{(k)})$ is found according to the ordering formula (3).
NI-11: $t^{(k)} = F^{(k)}/F$
NI-12: If $t^{(k)} < 1$ then $F = F^{(k)}$, $X = X^{(k)}$
NI-13: Goto **NI-5**
NI-14: Print X, F.
NI-15: Stop.

The proposed algorithm terminates after working for a specified number of iterations (above this number coincides with the number of edges). If desired, the condition for stopping the algorithm can be changed with different parameters such as if the best solution is repeated certain times.

Table 1: Results of proposed algorithm

G	**Optimal**	**NN Time(s)**	**Greedy Time(s)**	**Proposed Algorithm**
eil51	429.983	505.774 0.016	481.518 0.125	464.127377 0.120000
berlin52	7544.365	8182.192 0.000	9954.062 0.281	9384.357476 0.312000
st70	678.597	761.689 0.000	746.044 0.485	757.186879 0.350000
eil76	545.387	612.656 0.016	617.131 0.672	599.523516 1.406000
rat99	1211	1369.535 0.016	1528.308 1.875	1357.842252 1.942000
kroA100	21236.951	24698.497 0.016	24197.285 1.937	25313.585594 3.143000
kroB100	22141	25882.973 0.016	25815.214 2.469	26064.973238 2.504000
kroC100	20750.762	23566.403 0.015	25313.671 2.610	23777.840072 3.569000
kroD100	21294.290	24855.799 0.016	24631.533 2.359	24670.662409 2.670000
kroE100	22068	24907.022 0.016	24420.355 2.609	25477.295998 3.503000
rd100	7910.396	9427.333 0.015	8702.605 2.922	9272.783169 2.506000
eil101	642.309	736.368 0.015	789.112 2.609	704.912213 1.511000
lin105	14382.995	16939.441 0.015	16479.785 3.187	17161.161369 2.030000
pr107	44303	46678.154 0.016	48261.816 2.109	46053.729945 3.542000
ch130	6110.860	7198.741 0.016	7142.045 7.688	7404.914468 10.667999
kroA150	26524	31482.020 0.047	31442.994 11.094	30419.926627 3.503000
kroB150	26130	31320.340 0.047	31519.083 11.156	30710.393802 11.483000
rat195	2323	2628.561 0.109	2957.176 29.719	2733.523561 13.508001
kroA200	29368	34547.691 0.125	37650.812 45	34554.491633 56.243999

4 Experimental Results

The proposed algorithm is coded in C language. Experimental results were performed on a computer with a 4.00 GHz Intel Core i7-6700 CPU processor and 8 GB of RAM operating system 64-bit Windows. Since the running time of the algorithm is common to all algorithms, it does not include the graph reading and the creation of the distance matrix.

The performance of the proposed algorithm has been tested on samples in the TSPLIB library (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, 2021). The results are compared with greedy and nearest neighbour algorithms.

Experimental results for the iterative heuristic algorithm suggested are given in Table 1. For each row in the table, the first column contains the name and size of the problem, the second column is the length of the shortest path, and the remaining columns contain the results of the nearest neighbour algorithm, the greedy algorithm and the proposed algorithm, and the running times of the programs, respectively. The iteration value for the proposed algorithm is m.

5 Conclusion

In this paper, the Traveling Salesman Problem, one of well-known problems of Combinatorial Optimization, which has applications in many fields, has been discussed, a new iterative algorithm is proposed for the solution of this problem.

Proposed algorithm has been coded in the C++ programming language and calculations have been made. The proposed algorithm has been tested on test problems taken from international problem libraries and compared with the nearest neighbour algorithm and greedy heuristic.

As it is seen from Table 1, the proposed algorithm finds much better results than the nearest neighbour and greedy algorithms.

References

Aarts, E., Lenstra, J.K. (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons, London, 512.

Applegate, D.L., Bixby, R.E., Chavatal, V. & Cook, W.J. (2007). *The Travelling Salesman Problem: A Computational Study*. Princeton University Press, Princeton and Oxford, 608.

Davendra, D. (Ed.) (2010). *Travelling Salesman Problem: Theory and Applications*. BoD–Books on Demand.

Gutin, G., Punnen, A. (eds.). (2007). *The Traveling Salesman Problem and Its Variations.*, Springer US, 830.

Hubert, L.J., Baker, F.B. (1978). Applications of Combinatorial Programming to Data Analysis: The Traveling Salesman and Related Problems. *Psychometrika*, *43*(1), 81-91.

Johnson, D.S., McGeoch, L.A. (1997). The Traveling Salesman Problem: A Case Study in Local Optimization. *Local Search in Combinatorial Optimization*, *1*(1), 215-310.

Kızılateş, G., Nuriyeva, F. (2013). On the Nearest Neighbour Algorithms for the Traveling Salesman Problem. *In: Nagamalai D., Kumar A., Annamalai A. (eds) Advances in Computational Science, Engineering and Information Technology. Advances in Intelligent Systems and Computing*, *225*, Springer, Heidelberg, 111-118.

Nuriyev, U., Uğurlu, O. & Nuriyeva, F. (2018). Self Orginizing Iterative Algorithm for Travelling Salesman Problem. *18th IFAC Conference on Technology, Culture and International Stability (TECIS-2018), ELSEVIER, ScienceDirect IFAC PapersOnline*, *51*(30), 268-270.

Lenstra, J., Kan, A.R. (1975). Some simple applications of the travelling salesman problem. *Operational Research Quarterly*, *26*(4), 717-733.

Punnen, A. (2002). *The Traveling Salesman Problem: Applications, Formulations and Variations*. Kluwer Academic Publishers, Netherlands.

Reinelt, G. (1994). *The Traveling Salesman: Computational Solutions for TSP Applications*. Kluwer Springer-Verlag, Germany.

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/