*International Journal of Industry and Sustainable Development (**IJISD**)*
Volume 1, No. 2, PP. 76-84, July 2020, ISSN 2682-4000

Exact Minimum Lower Bound Algorithm for Traveling Salesman Problem

Mohamed Eleiche*

Faculty of Engineering, Egyptian Russian University, Cairo, Egypt

* Corresponding author: mohamed.eleiche@gmail.com

Abstract

The Traveling Salesman Problem (TSP) is defined by a given finite number of (*n*) cities along with the cost of travel between each pair of them. It is required to find the tour with least cost to visit all of the cities and returning to the starting point. Each city has to be visited once and only once. The TSP has direct applications in many engineering disciplines such as telecommunications, electricity, and a lot of network applications. It has high importance in Geoinformatics as it mathematically model the networks and infrastructures. This research presents the Minimum-Travel-Cost Algorithm for computing an exact lower bound for the general case of the (TSP). Although the TSP does not have yet an exact algorithm to determine its optimal path, this algorithm can help on identifying a minimal threshold that the exact unknown cost will exceed. The minimum-travel-cost algorithm is a dynamic programming algorithm to compute an exact and deterministic lower bound for the general case of the traveling salesman problem (TSP). The algorithm is presented with its mathematical proof and asymptotic analysis. It has a (n^2) complexity. A program is developed for the implementation of the algorithm and successfully tested among well known TSP problems and the results were consistent.

Key words: *traveling salesman problem (TSP); graph theory; network analysis.*

1- Introduction

The Traveling Salesman Problem (TSP) is defined by a given finite number of (*n*) cities along with the cost of travel between each pair of them. It is required to find the tour with least cost to visit all of the cities and returning to the starting point. Each city has to be visited once and only once (Applegate, et al. 2006). The travel costs are asymmetric in the sense that traveling from city a to city b does not cost the same as traveling from b to a. It is mathematically presented as a full graph with (*n*) nodes. The TSP has direct applications in many engineering disciplines such as telecommunications, electricity, and a lot of network applications. It has high importance in Geoinformatics as it mathematically model the networks and infrastructures (Eleiche and Markus 2010). In some cases, the length of edges is computed by geodesy as curved lines on the surface of reference ellipsoid. TSP is a prototype of hard combinatorial optimization problem where the possible solutions are *(n-1)!* and is considered NP-hard and NP-complete (Jungnickel 2008). This important problem does not have yet (2020) an exact deterministic algorithm to compute its optimal circuit.

The purpose of this research is to compute a minimum bound of the TSP in an exact algorithm for the general case of the problem which is asymmetrical data where $cost(u,i) \neq cost(i,u)$. The mathematical and asymptotic analysis of the algorithm are presented.

2- Mathematical Formulation

The TSP is composed of a weighted complete directed graph $G = (V, E)$, where (V) are the nodes with size (n), and (E) are the edges with size (n^2). $V = \{1, 2, \ldots, n\}$, and $E = \{(u,v) | u,v \in V\}$ and $\{cost(u,v) \in \mathbb{R} \geq 0\}$ (Bondy and Murty 1976). The graph with edges of negative cost is beyond the scope of this research.

A. Cost Array

The input data are stored in the cost array. It has the format [u, v, $cost(u,v)$], where (u) is the from-node of the edge, (v) is the to-node of the edge, and ($cost(u,v)$) is the distance of the edge. The data type of (u) and (v) is integer, while ($cost$) is a positive real number and may equals to zero. The cost array has a size of ($3\ n^2$) and its structure is shown in Table 1. It has a single row for each edge (u,v).

Table 1. The structure of input cost array

u **(from-node)**	***v*** **(to-node)**	***Cost***
1	*1*	∞
…	…	…
1	*n*	…
2	*1*	…
2	*2*	∞
…	…	…
n	*n*	∞

B. Minimum Travel Array

The Minimum-Travel-Array is the main output of this algorithm. Its structure is [*InCost*, *u*, *i*, *v*, *OutCost*], where (*InCost*) is the cost of the edge (*u*,*i*), (*u*) is the incident node to node (*i*), (*i*) is the node of interest ($1 \leqslant i \leqslant n$), (*v*) is the outgoing node from node (*i*), and (*OutCost*) is the cost of the edge (*i*,*v*). The size of the Minimum-Travel-Array is (*5n*) (Eleiche, Markus 2010), and Table 2 shows its structure.

Table 2. Minimum Travel Array

InCost	***u***	*i*	***v***	***OutCost***
…	…	*1*	…	…
InCost	*u*	*i*	*v*	*OutCost*
…	…	*n*	…	…

3- Hypothesis

The idea of this solution is to divide the problem into two main subproblems, incident side and outgoing side. The incident side is where the node (*i*) is the arrival destination from another node (*u*) with the minimal arrival cost (*InCost1*). The outgoing side is to depart from

the node (*i*) to another node (*v*) with the minimal departure cost (*OutCost1*), as shown in Fig. 1. This is traveling to and from node (*i*) with the minimal cost. It is not possible to travel through node (*i*) with a cost less than ($minCost = InCost + OutCost$), as shown in equation (*1*). This is a direct and simple selection application for the minimal cost value from each side.

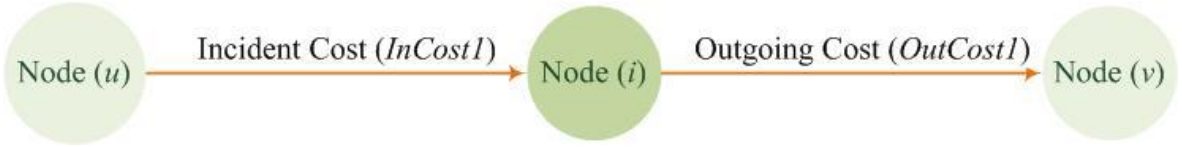

Fig. 1. Incident and outgoing nodes for node (*i*).

$$InCost1 = E(u,i)\ such\ that, E(u,i) = \min\ E([1,n],i)$$
$$OutCost1 = E(i,v)\ such\ that, E(i,v) = \min \mathrm{E}(i,[1,n]) \quad (1)$$
$$\min Travel\ Cost\ for\ node\ (i) = minCost = InCost1 + OutCost1$$

A. Prevent Loop Condition

In some cases, we find that node (*u*) = node (*v*), same node has the minimum cost to arrive to node (*i*) and to depart from it, this is not allowed by the definition of TSP, which states that each node is visited only once (Eleiche 2015). In such case, another computation is required. We select the second node (*u2*), from incident side, such that it has the second minimum cost (*InCost2*) to arrive to node (*i*). Similarly, the node (*v2*) from outgoing side has the second minimum cost (*OutCost2*) to depart from node (*i*), as shown in Fig. 2. It is worth to note that although node (*u)* = node (*v*) = node (*a*), however, InCost1 ≠ OutCost1, as the problem is not symmetrical. The computation of (*minCost*) is shown in equation (*2*).

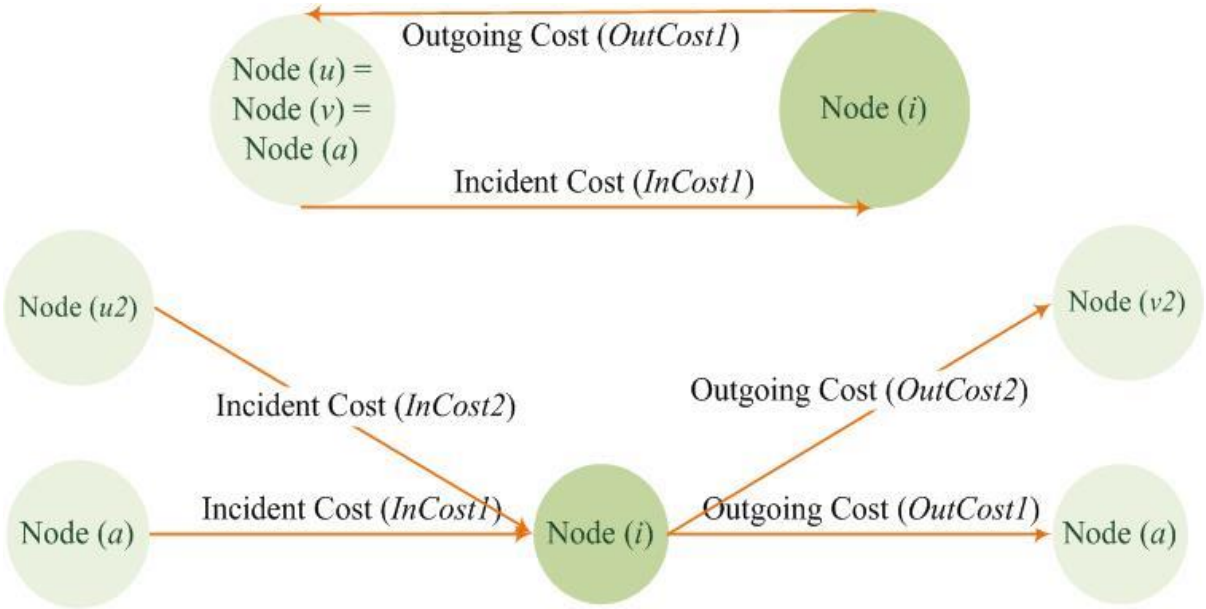

Fig. 2. Second minimum cost for Node (i).

$$
\begin{aligned}
&InCost2 = E(u2,i)\ such\ that,\\
&\qquad E(u2,i) = \min\ E([1,n],i)\ \&\&\ (E(u2,i) \geq\ E(u,i)\ )\\
&OutCost2 = E(i,v2)\ such\ that\\
&\qquad E(i,v2) = \min \mathrm{E}(i,[1,n])\ \&\&\ (E(i,v2) \geq\ E(i,v))\\
&minCost = min\begin{cases}(InCost1 + OutCost2), Incost = incost1, OutCost = OutCost2\\ (InCost2 + OutCost1), Incost = incost2, OutCost = OutCost1\\ (InCost2 + OutCost2), Incost = incost2, OutCost = OutCost2\end{cases}
\end{aligned}
\tag{2}
$$

B. Exact Minimum Bound for TSP

Let C_{TSP} be the required cost for optimal circuit which is unknown, and $C_{minBound}$ is an exact minimum bound for TSP, such that ($C_{minBound} \leqslant C_{TSP}$) is shown in equation (3).

$$
C_{minBound} = max\begin{cases}\sum_{1}^{n} InCost\\ \sum_{1}^{n} OutCost\end{cases}
\tag{3}
$$

The exact minimum bound $C_{minBound}$ equals to the higher value from ($\sum_1^n InCost$) and ($\sum_1^n OutCost$), which are the summation of the cost of incident edges and outgoing edges respectively from the Minimum-Travel-Array in Table 2.

4- Mathematical Proof

The Minimum-Travel-Array stores the minimum exact cost to pass through each node, and by summation of costs from both sides, exact minimum bound for the TSP is computed. Let us consider the TSP_Array is similar in structure to the Minimum-Travel-Array, and it contains the required solution for TSP, so that (C_{TSP}) is the required cost for optimal circuit, $C_{in}(i)$ and $C_{out}(i)$ are the cost of incident edge and outgoing edge to node (i) in optimal solution, as shown in Table 3.

Table 3. The structure of TSP_Array (optimal solution)

C_{in}	u	i	v	C_{out}
…	…	*1*	…	…
C_{in}	u	i	v	C_{out}
…	…	*n*	…	…

In case of the optimal solution, the following equation (*4*) is applied

$$
C_{TSP} = \sum_{1}^{n} C_{in}(i) = \sum_{1}^{n} C_{out}(i)
\tag{4}
$$

Let us consider the following simple mathematical rule, Consider that : ($C = a$), as shown in equation (*5*):

$$
\begin{aligned}
&\mathrm{C} = a = a + (b - b)\ \ (where\ \{(a,b,C) \in \mathbb{R} \geq 0\ and\ a \geq b\ \}\\
&\mathrm{C} = a = (a - b) + b \qquad \{(a - b) \geq 0\ \}\text{, then}\\
&\mathrm{C} \geq b
\end{aligned}
\tag{5}
$$

In the previous example, (C) is the required cost for the minimum cycle (C_{TSP}) and it is

unknown, and it equals to the quantity (a). The quantity (b) is the sum of the minimum travel cost for each vertex ($C_{minBound}$) and it is a known quantity. It is evident that both $\{(a, b) \geq 0 \; and \; a \geq b\}$ by adding and subtracting (b) still the equation is valid, and still (a-b) is unknown but $\{(a - b) \geq 0\}$. This means that (C) must be greater than (or equal) to (b). By applying this concept to the minimum travel cost and assuming that the TSP_Array in Table 3 represents the tour of least cost, then as shown in equation (*6*),

$$
\begin{aligned}
C_{TSP} &= \sum_{1}^{n} C_{in}(i) = \sum_{1}^{n} (C_{in}(i) - InCost) + \sum_{1}^{n} InCost \\
C_{TSP} &= \sum_{1}^{n} C_{out}(i) = \sum_{1}^{n} (C_{out}(i) - OutCost) + \sum_{1}^{n} OutCost \\
C_{TSP} &\geq \textstyle\sum_{1}^{n} InCost \geq 0 \\
C_{TSP} &\geq \textstyle\sum_{1}^{n} OutCost \geq 0 \\
C_{minBound} &= max \begin{cases} \displaystyle\sum_{1}^{n} InCost \\ \displaystyle\sum_{1}^{n} OutCost \end{cases}
\end{aligned}
\tag{6}
$$

In the equation (*6*), for each vertex (i), the minimum incident cost ($InCost$) was added and subtracted, which does not change the value of the equation. Then, the value of the incident cost to the vertex (i) from minimum cycle (C_{in}) is represented as the known minimum incident cost ($InCost$) in addition to another quantity ($C_{in} - InCost$). It is evident that the quantity (C_{in}) is unknown, while the other quantity ($InCost$) is well known.

It is evident that the Minimum-Travel-Array does not represent the required least tour for TSP, and many nodes will have travel cost higher than their minimum-travel-cost within the least tour. However, the Minimum-Travel-Array is important characteristic for the TSP, and provides exact minimum bound that the least cost will exceed.

A. Algorithm of Minimum Bound for TSP and Main Results

The pseudo code of the algorithm

Input: A weighted complete directed graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, and $E = \{(u, v) | \; u, v \in V\} \; and \; \{cost(u, v) \in \mathbb{R}\}$ such that $\{ cost(u, v) \geq 0\}$ and $cost(u, v) \neq cost(v, u)$

Output: The minimum travel cost array for each vertex and the minimum lower bound for the cost to visit each vertex in V only once.

```
1.  Class MinTravel{InCost, From_Node, Node_ID, To_Node , OutCost}
2.  For each vertex  i ∈ V
3.      MinTravel[i,3] ← i
4.      MinTravel[i,2] ← u of minimum cost of  E' = {(u, i)|u ∈ V}
5.      MinTravel[i,1] ← minimum cost of  E' = {(u, i)|u ∈ V}
6.      MinTravel[i,4] ← v of minimum cost of  E'' = {(i, v)|v ∈ V}
7.      MinTravel[i,5] ← minimum cost of  E'' = {(i, v)|v ∈ V}
8.     if (MinTravel[i,2] ==  MinTravel[i,4] ) then Prevent(i)
9.  TSPLowerBound()
```

Procedure Prevent(i)

1. InCost2 ← $second$ min $cost\ E' = \{(u2, i) | u2 \in V \&\& u2 \neq MinTravel[i, 2]\}$
2. $u2 \leftarrow u2\ of\ second$ min $cost\ E' = \{(u2, i) | u2 \in V \&\& u2 \neq MinTravel[i, 2]\}$
3. OutCost2 ← $second$ min $cost\ E'' = \{(i, v2) | v2 \in V \&\& v2 \neq MinTravel[i, 4]\}$
4. $v2 \leftarrow v2\ of\ second$ min $cost\ E'' = \{(i, v2) | v2 \in V \&\& v2 \neq MinTravel[i, 4]\}$
5. C1 = MinTravel[i,1] + OutCost2
6. C2 = InCost2 + MinTravel[i,5]
7. C3 = InCost2 + OutCost2
8. C = min[C1, C2, C3]
9. Case (1): C = C1
10. MinTravel[i,4] ← v2
11. MinTravel[i,5] ← OutCost2
12. Case (2): C = C2
13. MinTravel[i,2] ← u2
14. MinTravel[i,1] ← InCost2
15. Case (3): C = C3
16. MinTravel[i,2] ← u2
17. MinTravel[i,1] ← InCost2
18. MinTravel[i,4] ← v2
19. MinTravel[i,5] ← OutCost2

Procedure TSPLowerBound

1. for i← 1 to n
2. Sum_in = Sum_in + MinTravel [i,1]
3. Sum_out = Sum_out + MinTravel [i,5]
4. If (Sum_in > Sum_out)
5. then TSPLowerBound = Sum_in
6. else TSPLowerBound = Sum_out

B. Main Results

A C++ program was implemented for this algorithm to test its validity among some TSP problems with (best) known solutions. The TSPLIB website provides sample TSP problems with best known solutions (http://www.math.uwaterloo.ca/tsp/problem/outlinks.html , May 2020) in order to test the validity of proposed solutions for this interesting problem. This program tested three problems which are (br17, ry48p, ft53) from TSPLIB, and a fourth problem is defined at (http://www.math.uwaterloo.ca/tsp/college/).

Table 4. Applications of algorithm

Problem Name	Size (n)	Best known solution	Incident Cost	Outgoing Cost	Lower Bound
br17	17	39	0	24	24
ry48p	48	14422	12987	11964	12987
ft53	53	6905	3580	3989	3989
College 647	647	47,149,705	25,615,500	42,777,207	42,777,207

As shown in Table 4, all the four tested problems had the computed lower bound less than best-known-solution. This test prove practically the validity of the algorithm.

C. Algorithm Classification

The Minimum Travel Cost Algorithm for the Traveling Salesman Problem has the following characteristics:

1) It divides the problem into two separate subproblems: incident side and outgoing side, solving each one separately.
2) The algorithm is recursive, it computes the minimum incident and outgoing cost for each node.
3) It has iterative part, in which for each node the minimum cost is computed for the whole problem.
4) It memorizes the output for each step for further use.

From the above characteristics, the algorithm can be classified as Dynamic Programming (DP) algorithm (Bertsekas 2005).

D. Asymptotic Analysis of the Algorithm

The main function finds the minimum travel cost for each node in the TSP. The highest complexity is for the selection of the minimum (incident/outgoing) cost which runs (*n-1*) time for each node, making it (*n(n-1)*) for each side, as shown in Table 5. The *Prevent(i)* function is executed only when both incident and outgoing nodes are the same. This can never happen in best condition and can appear at each node in worst condition.

Table 5. Asymptotic analysis of main function

Whole Problem	Each Node	Function	Complexity
Yes	Yes	MinTravel[i,3] ← i	*n*
Yes	Yes	MinTravel[i,2] ← u of minimum cost of $E' = \{(u,i)\|u \in V\}$	*n*
Yes	Yes	MinTravel[i,1] ← minimum cost of $E' = \{(u,i)\|u \in V\}$	*n(n-1)*
Yes	Yes	MinTravel[i,4] ← v of minimum cost of $E'' = \{(i,v)\|v \in V\}$	*n*
Yes	Yes	MinTravel[i,5] ← minimum cost of $E'' = \{(i,v)\|v \in V\}$	*n(n-1)*
Yes	Yes	if (MinTravel[i,2] == MinTravel[i,4])	*n*
May be	May be	Prevent(i)	*Worst condition = (n* (n-2))* *Best Condition = 0*
no	no	TSPLowerBound()	*n*

1) *Prevent(i) asymptotic analysis:* This function computes the second minimum cost for each node (Kavitha, et al. 2008). Similarly to main function, the minimum second cost is preformed (*n-2*) times, as shown in Table 6**Table 6**.

Table 6. Asymptotic analysis for Prevent(i) function

Each Node	Function	Complexity
Yes	$\text{InCost2} \leftarrow second \min cost\ E' = \{(\text{u2}, \text{i}) \mid \text{u2} \in \text{V \&\& u2} \neq \text{MinTravel[i, 2]}\}$	*n-2*
Yes	$u2 \leftarrow u2\ of\ second \min cost\ E' = \{(\text{u2}, \text{i}) \mid \text{u2} \in \text{V \&\& u2} \neq \text{MinTravel[i, 2]}\}$	*1*
Yes	$\text{OutCost2} \leftarrow second \min cost\ E'' = \{(\text{i}, \text{v2}) \mid \text{v2} \in \text{V \&\& v2} \neq \text{MinTravel[i, 4]}\}$	*n-2*
Yes	$v2 \leftarrow v2\ of\ second \min cost\ E'' = \{(\text{i}, \text{v2}) \mid \text{v2} \in \text{V \&\& v2} \neq \text{MinTravel[i, 4]}\}$	*1*

2) *TSPLowerBound() asymptotic analysis:* This function is simple and compute the total value for the minimum-travel-cost as shown in Table 7**Table 7**, by direct addition with cost of (*n*).

Table 7. Asymptotic analysis for TSPLowerBound() function

Each Node	Function	Complexity
Yes	$\text{Sum\_in} = \text{Sum\_in} + \text{MinTravel [i,1]}$	*n*
Yes	$\text{Sum\_out} = \text{Sum\_out} + \text{MinTravel [i,5]}$	*n*

3) *The asymptotic analysis of the algorithm:* The algorithm highest complexity functions are the selection of minimum cost with complexity of *O(n(n-1))* and *Prevent(i)* with complexity of *O(n(n-2))*. Then, the algorithm has an overall bound complexity of $O(n^2)$.

5- Conclusion

This research presented the Minimum-Travel-Cost Algorithm for computing an exact lower bound for the general case of Traveling-Salesman-Problem (TSP). Although the TSP does not have yet an exact algorithm to determine its optimal path, this algorithm can help on identifying a minimal threshold that the exact unknown cost will exceed.

It selected the minimum cost to arrive to each node and depart from it. Then, it computed the sum of the total incident and departure costs. The highest value from both sum is a lower bound for the problem. The mathematical proof for the algorithm was presented. The algorithm was implemented into a program and tested among well known cases for existing problems, and the results were consistent. The algorithm can be classified as dynamic programming algorithm with complexity of $O(n^2)$. It does not solve the Traveling-Salesman-Problem, however it provides an exact and deterministic lower bound for its general case.

References

[1] Applegate, David , Robert Bixby, Vasek Chvátal , and William Cook. The Traveling Salesman Problem: A Computational Study. Princeton University Press, 2006.
[2] Bertsekas, Dimitri . Dynamic Programming and Optimal Control. Athena : Athena Scientific, 2005.
[3] Bondy, J, and U Murty. Graph Theory with Applications. London: Macmillan, 1976.
[4] Eleiche, Mohamed. "Applying Minimum Travel Cost Approach on 43–Nodes Traveling Salesman Problem." Pure and Applied Mathematics Journal 4, no. 1 (2015): 9-23.
[5] Eleiche, Mohamed, and Bela Markus. "Applying Minimum Travel Cost Approach On 17–Nodes Traveling Salesman Problem." Geomatikai Közlemények XIII, no. 2 (2010): 15-22.
[6] Jungnickel, D. " Graphs, Networks, and Algorithms." Springer, 2008: 433-472.
[7] Kavitha, Telikepalli , Kurt Mehlhorn, Dimitrios Michail, and Katarzyna Paluch. "An O(m2n) Algorithm for Minimum Cycle Basis of Graphs." Algorithmica, 2008: 333–349.