Exact Minimum Lower Bound Algorithm for Traveling Salesman Problem

Mohamed Eleiche
GeoTiba Systems
mohamed.eleiche@gmail.com

Abstract

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.

Keywords

Traveling Salesman Problem; Big O notation; exact algorithm

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. 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), it is mathematically presented as a full graph with (n) nodes.

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. Data Structure

The TSP is composed of (n) nodes (V) and (n^2) edges (E). A weighted complete directed graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, and $E = \{(u,v) | u,v \in V\}\ and\ \{cost(u,v) \in \mathbb{R}\}$ (Bondy and Murty 1976).

2.1 Input

The input data, which is the cost array, is stored in array with the format [from-node, to-node, cost]. From-node and to-node have integer type, while cost type is real. The size of the input array is ($3\ n^2$) and its structure is shown in Table 1. The input array is sorted by from-node then to-node in order to fast seek for minimum cost for each node from both direction.

Table 1 Input data structure

From-Node	To-Node	Cost
1	*1*	∞
…	…	…
1	*n*	…
2	*1*	…
2	*2*	∞

…	…	…
n	*n*	∞

2.2 Minimum Travel Array

The Minimum-Travel-Array is the main output of this algorithm. It is (*5 n*) in size. The first and second columns are for incident cost, and its incident node. Third is the ID of the node. This column is sorted in ascending order for all the nodes ID. Then, the fourth and fifth columns are the outgoing node, and its outgoing cost (Eleiche, Markus 2010).

The algorithm is based on creating the Minimum Travel Array for the problem. It has the same structure as the following Table 2.

Table 2 Minimum Travel Array

Incident Cost	**Incident Node (*u*)**	**Node ID**	**Outgoing Node (*v*)**	**Outgoing Cost**
…	…	*1*	…	…
InCost	*u*	*i*	*v*	*OutCost*
…	…	*n*	…	…

3. Main Idea

The idea of this algorithm 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*). The outgoing side is to depart from the node (*i*) to another node (*v*), as shown in Figure 1.

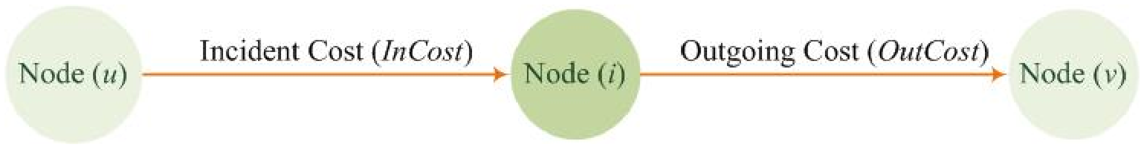

Figure 1 Incident and outgoing nodes

For each subproblem, the minimum cost is determined for each node (*i*). From the incident side, the node (*u*) with the minimum cost (*InCost*) to arrive to node (*i*) is determined and stored inside the main array. From the outgoing side, the node (*v*) with the minimum cost (*OutCost*) to depart from node (*i*) is determined and stored inside the main array.

$$InCost = E(u,i)\ such\ that\ E(u,i) = \min\ E([1,n],i)$$

$$OutCost = E(i,v)\ such\ that\ E(i,v) = \min \mathrm{E}(i,[1,n])$$

$$\min Travel\ Cost\ for\ node\ (i) = minCost = InCost + \ OutCost$$

After the Minimum Travel Array is computed and completed, each node is analyzed separately to ensure if the node (*i*) has the same node as incident and outgoing node. This is not allowed by the definition of TSP, which states that each node is visited only once (Eleiche 2015).

In case node(*u*) = node(*v*) for node(*i*), the second cost is computed from each side and the minimum travel cost for the node is computed, to prevent same node to be in both sides for node (*i*).

3.1 Prevent same node

In case where node (*u*) = node (*v*), same node has the minimum cost to arrive to node (*i*) and to depart from it, another computation is required. The second node (*u2*), from incident side, is selected such has it has second minimum cost (*InCost2*) to arrive to node (*i*), and node (*v2*) from outgoing side which has the second minimum cost (*OutCost2*) to depart from node (*i*), as shown in Figure 2. It is worth to note that although node (*u*) = node (*v*) = node (*a*), however, *InCost* ≠ *OutCost*, as the problem is not symmetrical.

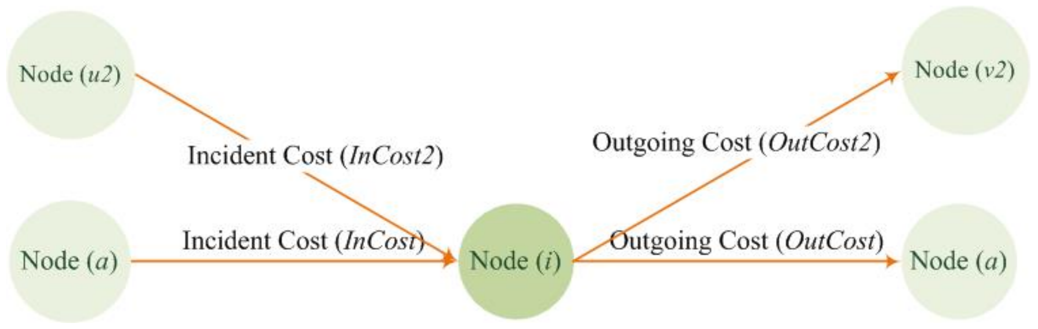
Node (*u2*)
Incident Cost (*InCost2*)
Outgoing Cost (*OutCost2*)
Node (*v2*)
Node (*a*)
Incident Cost (*InCost*)
Node (*i*)
Outgoing Cost (*OutCost*)
Node (*a*)

Figure 2 Second minimum cost for Node (i)

$$InCost2 = E(u2,i)\ such\ that\ E(u2,i) = \min\ E([1,n],i)\ \&\&\ E(u2,i) \geq\ E(u,i)$$

$$OutCost2 = E(i,v2)\ such\ that\ E(i,v2) = \min\ \mathrm{E}(i,[1,n])\ \&\&\ E(v2,i) \geq\ E(v,i)$$

Let *minCost* = Minimum travel cost for same node case

$$minCost\ =\ min[(InCost + OutCost2),(InCost2 + OutCost),(InCost2 + OutCost2)]$$

Case (1): $minCost\ = (InCost + OutCost2)$

In this case, the Minimum-Travel-Array will be filled as follows *[InCost, u,i,v2,OutCost2*].

Case (2): $minCost\ = (InCost2 + OutCost)$

In this case, the Minimum-Travel-Array will be filled as follows [*InCost2, u2,i,v,OutCost*].

Case (3): $minCost\ = (InCost2 + OutCost2)$

In this case, the Minimum-Travel-Array will be filled as follows [*InCost2, u2,i,v2,OutCost2*].

4. Minimum bound Algorithm for TSP

Input: A weighted complete directed graph *G* = (*V*, *E*), where *V* = {1, 2, …, 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.  u2 ← u of second minimum cost of  $E' = \{(u,i)|u \in V\}$
  2.  InCost2  ← second minimum cost of  $E' = \{(u2,i)|u \in V\}$
  3.  v2  ← v of second minimum cost of  $E'' = \{(i,v)|v \in V\}$
  4.  OutCost2 ← second minimum cost of  $E'' = \{(i,v2)|v \in V\}$
  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
```

5. Proof

The minimum incident cost (*Incost*) is the least cost to arrive to node (*i*), and (*OutCost*) is the least cost to depart from it. It is not possible to travel through node (*i*) with a cost less than $minCost = InCost + OutCost$.

The Minimum-Travel-Array presents minimum exact cost to travel each node, and by summation of costs from both sides, exact minimum bound for the TSP is computed.

Consider that : (C = a), then:

$C = a = a + b - b \qquad (where\ \{(a,b) \geq 0\ and\ a \geq b\ \}$
$C = a = (a - b) + b \qquad \{(a - b) \geq 0\ \}$, then

$C \geq b$

In the previous example, (C) is the required cost for the minimum cycle and it is unknown, and it equals to the quantity (a). The quantity (b) is the minimum travel cost for each vertex and it is a known quantity. It is evident that both $\{(a,b) > 0 \; and \; a \geq b\}$ by addind and sudtracting (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 Minimum-Travel-Array in Table 2 represents the tour of least cost, then:

$$C_{TSP} = \sum_{1}^{n} C_{TSP-in} = \sum_{1}^{n} C_{TSP-out}$$

$$C_{TSP} = \sum_{1}^{n} C_{TSP-in} = \sum_{1}^{n} (C_{TSP-in} - InCost) + \sum_{1}^{n} InCost$$

$$C_{TSP} = \sum_{1}^{n} C_{TSP-out} = \sum_{1}^{n} (C_{TSP-out} - OutCost) + \sum_{1}^{n} OutCost$$

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

$C_{TSP} \geq \sum_{1}^{n} InCost \geq 0$ ($known \; quantity \; computed \; by \; algorithm$)

$C_{TSP} \geq \sum_{1}^{n} OutCost \geq 0$ ($known \; quantity \; computed \; by \; algorithm$)

$$\text{TSPLowerBound} = \begin{cases} \sum_{1}^{n} InCost, & \sum_{1}^{n} InCost > \sum_{1}^{n} OutCost \\ \sum_{1}^{n} OutCost, & \sum_{1}^{n} OutCost \geq \sum_{1}^{n} InCost \end{cases}$$

Where

C_{TSP} $is\ the\ cost\ of\ the\ required\ mimimum\ \ cycle\ (unknown)$

C_{TSP-in} $is\ the\ cost\ to\ arrive\ to\ the\ vertex(i) from\ incident\ directionfrom\ the\ mimimum\ \ cycle$

$C_{TSP-out}$ $is\ the\ cost\ to\ leave\ the\ vertex(i) from\ outgoing\ direction\ from\ the\ mimimum\ \ cycle$

$InCost\ \ is\ the\ mimimum\ \ cost\ to\ arrive\ to\ the\ vertex\ (i)\ from\ incident\ direction$

$OutCost\ \ is\ the\ mimimum\ \ cost\ to\ depart\ from\ the\ vertex(i) from\ outgoing\ direction$

C_{TSP} $is\ unknown$

C_{TSP-in} , $C_{TSP-out}$ $are\ unknown\ for\ each\ vertex(i)$

$InCost\ \ \&\ OutCost\ are\ known\ for\ each\ vertex(i)$

Then,

$C_{TSP} \geq \sum_1^n InCost$ and $C_{TSP} \geq \sum_1^n OutCost$

The minimum bound for the general case of TSP is the higher from sum of incident and outgoing cost.

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.

6. Asymptotic Analysis and 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 memoizes the output for each step for further use.

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

6.1 Main Function asymptotic analysis

The main function finds the minimum cost for each side for the TSP. The highest cost is for the minimum cost which runs (*n*) time for each node, making it (n^2) for each side, as shown in Table 3. If the input array of data is not sorted, then it will cost (n^3). 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 3 Asymptotic analysis of main function

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

6.2 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 has cost of (*n-1*), as shown in Table 4.

Table 4 Asymptotic analysis for Prevent(i) function

Each Node	Function	Complexity
Yes	$u2 \leftarrow u$ of second minimum cost of $E' = \{(u,i) \mid u \in V\}$	*1*
Yes	$InCost2 \leftarrow$ second minimum cost of $E' = \{(u2,i) \mid u \in V\}$	*n-1*
Yes	$v2 \leftarrow v$ of second minimum cost of $E'' = \{(i,v) \mid v \in V\}$	*1*
Yes	$OutCost2 \leftarrow$ second minimum cost of $E'' = \{(i,v2) \mid v \in V\}$	*n-1*

6.4 TSPLowerBound() asymptotic analysis

This function is simple and compute the total value for the minimum-travel-cost as shown in Table 5, by direct addition with cost of (*n*).

Table 5 Asymptotic analysis for TSPLowerBound() function

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

6.5 Algorithm asymptotic analysis

The algorithm has an upper bound of $O(n^2)$ before arriving to *Prevent(i)* function. This function can be executed (n) times in worst condition and never executed in best condition. It has (*n-1*) complexity for each node separately, leading to complexity of ($n(n-1)$) same $O(n^2)$ too.

7. Application

A C++ program was implemented for this algorithm to test its validity among some TSP problems with (best) known solutions. The TSPLIB website (http://www.math.uwaterloo.ca/tsp/problem/outlinks.html, December 2015) provides sample TSP problems with best known solutions in order to test the validity of proposed solutions for this interesting problem. This research tested three problems which are (br17, ry48p, ft53) as shown in Table 6. The fourth problem is defined at (http://www.math.uwaterloo.ca/tsp/college/).

Table 6 Applications of algorithm

Problem Name	**Size (n)**	**Best known solution**	**Incident Cost**	**Outgoing Cost**	**Minimum Cost**
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 6, all the four tested problems had the computed lower bound less than best-known-solution. This test prove practically the validity of the algorithm.

8. Conclusion

This article presented the Minimum-Travel-Cost Algorithm for computing an exact lower bound for the general case of Traveling-Salesman-Problem. It computes the minimum cost to arrive to each node and depart from it. Then, it compute the total cost to arrive to all nodes, and depart from all nodes. The highest from arrival and departure costs is the lower bound for the problem. The mathematical proof for the algorithm was presented. The algorithm is classified as dynamic programming algorithm with complexity of $O(n^2)$. The algorithm was implemented into a program and tested among well known cases for existing problems, and the results were consistent and validate the algorithm. It does not solve the Traveling-Salesman-Problem, however it provides an exact and deterministic lower bound for the general case.

References

Applegate, David , Robert Bixby, Vasek Chvátal , and William Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006.

Bertsekas, Dimitri . *Dynamic Programming and Optimal Control.* Athena : Athena Scientific, 2005.

Bondy, J, and U Murty. *Graph Theory with Applications.* London: Macmillan, 1976.

Eleiche, Mohamed. "Applying Minimum Travel Cost Approach on 43–Nodes Travelling Salesman Problem." *Pure and Applied Mathematics Journal* 4, no. 1 (2015): 9-23.

Eleiche, Mohamed, and Bela Markus. "Applying Minimum Travel Cost Approach On 17–Nodes Travelling Salesman Problem." *Geomatikai Közlemények* XIII, no. 2 (2010): 15-22.

Jungnickel , D. " Graphs, Networks, and Algorithms." *Springer*, 2008: 433-472.

Kavitha, Telikepalli , Kurt Mehlhorn, Dimitrios Michail, and Katarzyna Paluch. "An O(m2n) Algorithm for Minimum Cycle Basis of Graphs." *Algorithmica*, 2008: 333–349.