JKAU: Eng. Sci., **vol. 4**, pp. 117-122 (1412 A.H./1992 A.D.)

An Algorithm for Solving the Traveling Salesman Problem

M. HAMED
College of Arts and Science,
Bahrain University, Isa Town, Bahrain

ABSTRACT. The main objective of the paper is to present an algorithm for finding a solution to the traveling salesman problem. The solution found by the algorithm being an optimal one or not, depends on the values of the elements of the cost matrix. The algorithm is described and its time complexity is calculated and compared to other algorithms in the literature. It is shown that the proposed algorithm is efficient as it finds the solution in shorter time if compared to other algorithms.

1. Introduction

The ordinary traveling salesman problem (TSP) is formulated in the literature[1] as: "Finding the tour with minimum cost for passing through each of n cities exactly once, starting from and ending at an arbitrary city". This problem is classified as being NP-complete[2]. Actually, there are several variations of the TSP, *e.g.*, time-dependent TSP[3,4], and stochastic TSP[5]. Each variation lead to a rather different problem. Our main concern here is the ordinary TSP.

Exhaustive search for an optimal solution by trying all possible permutations is a method which is not referred to any specific author. The time of such an approach is $O(n!)$ since we must consider $(n-1)!$ different permutations, and each permutation takes $O(n)$ time to evaluate. Another algorithm due to Held and Karp[6] gives the optimal solution in time $O(n^2\ 2^n)$. This algorithm uses dynamic programming techniques. Other algorithms in the literature do not give an optimal solution, but rather they give a "good" solution based on heuristics. Examples of such heuristics are found in Kruskal[7] and Lin & Kernighan[8]. When comparing different al-

gorithms of solving the problem, it is necessarry to consider only those ones that yield an optimal solution. The time complexity of the proposed algorithm is $O[2^{n/2} + m \log_2 m]$.

The idea of the algorithm depends on isolating the needed elements of the cost matrix, sorting them in ascending order, then synthesizing the tour in such a way to include minimum values and exclude maximum values of the cost matrix.

It should be noted that the solution found by the algorithm may or may not be an optimal one. This is dependent upon the values of the cost matrix. The algorithm checks whether the given solution is seen to be an optimal one or there might be a possibility to be a near-optimal solution. This is realized through the satisfaction of some relationship between some elements of the cost matrix.

In section 2 of the paper, the proposed algorithm is described. Also the time complexity of the algorithm is calculated and compared to other algorithms in the literature.

2. The Proposed Algorithm

2.1. General Principles

The principles according to which the algorithm is designed are :

1 – As the cost matrix is symmetric, then only the elements of the upper triangle are needed.

2 – The minimum cost tour is synthesized basically from the first $n/2$ low-value cost elements which incorporate k nodes, plus some additional elements that incorporate r remaining nodes. Permutations are tried for different possible positions of the basic and remaining nodes.

3 – As the choice of the start city of the tour is immaterial, then the number of permutations can be reduced as several tours may have the same cost elements but differ only in the start point.

4 – The solution is exactly an optimal one if $S1 < S2$.

2.2 Description of the Algorithm

The input to the algorithm is an $n \times n$ cost matrix C. As the cost matrix is symmetric ($c_{ij} = c_{ji}$) and diagonal elements (c_{ii}) are zeros, then the number of unrepeated non-zero elements (upper triangle) is $m = [n \times (n - 1)/2]$; call them effective elements. It should be noted that each element of the matrix C is an arc whose head and tail nodes are cities (row and column).

The first step of the algorithm is to sort the effective elements in ascending order and put them in vector A. The corresponding head and tail nodes are put in vectors S and E respectively. Then, a check is made to see whether the solution is an optimal one. The next step is to synthesize the tour with minimum cost by including the first $[n/2]$ elements of vector A and excluding the last $[n/2]$ elements. The maximum number of tours to be synthesized is $NMAX = 2^{[n/2]}$.

The algorithm works as follows :

1 – Build vector A from the upper triangle elements of the cost matrix C (without diagonal elements). While building vector A, the other two vectors S and E are filled with start and end cities of the corresponding cost elements respectively. Thus element S_i represents the start city of the cost element A_i, while element E_i represents the end city of A_i.

2 – Sort the elements of vector A in ascending order. This can be done using Quicksort[9]. Note that vectors S and E have to suffer from corresponding rearrangements.

3 – Check whether the given solution is seen to be an optimal one or there is a possibility to be a near-optimal solution.

4 – Use the first $n/2$ elements of S and E to build the first part of the tour with minimum cost. The main idea here is to group the cities' numbers of S and E into clusters. A cluster is a set of city numbers that can be considered as a part of the tour. The arcs resulting from considering each two adjacent cities in the cluster are important in finding the tour with minimum cost. A cluster is built according to the following procedure :

a – Put the first element of both S and E into the first two elements of R.

b – Use i to point to the next element of S and E. Assume that k is the number of elements in R. Search for either E_i or S_i to be equal to R_j ($j = 1, 2, \ldots, k$) if yes, then the element of S_i (or E_i) would be added to R in the proper position (of an existing cluster); otherwise add both elements S_i and E_i at the rear end of R (incrementing k) to make a new cluster. Note that if element j of R was used before in one of the clusters as indicated by the corresponding element j of vector U, then it cannot be used again in a new cluster.

5 – Synthesize the tour. The tour is synthesized by grouping different clusters, together with remaining nodes. As there are at most $[n/2] + 1$ alternative (clusters + remaining nodes), and each alternative has at most two different permutations, then the maximum number of permutations is $2^{[n/2]+1}$. Because of the fact that the tour may arbitrarily start at any city, we will find that the maximum number of permutations can be reduced to $2^{[n/2]+1}/2$ (*i.e.*, $2^{[n/2]}$) only (half of the permutations are repeated with different start city).

2.3 The Optimal Solution

The condition for an optimal solution is that $S1 < S2$.

Proof

– From the definition of the matrix C and vector A, the label of each node is mentioned in the set $NS \cup NE$ an equal number of times equal to

$$\frac{n \times (n-1)}{2} \times 2 \div n = (n-1) \text{ times.}$$

– The maximum number of times a node is mentioned in the excluded part of A ($R4$) is $n/2$ (number of elements in $R4$).

– As $(n-1)$ is $> n/2$, then a tour can be synthesized without using the excluded part of A ($R4$).

– As the set $R1$ contains minimum values, then the worst case for a tour including elements of $R1$ and does not include elements of $R4$ is that one consisting of elements in $R1 \cup R3$, giving a tour of cost $S1$.

– On the other hand, the best case for a tour that does not include elements of $R1$ is that tour consisting of elements in $R2$, giving a tour of cost $S2$.

– Thus if $S1 < S2$ then the worst case of a tour including elements in $R1$ is better than the best case of a tour where elements of $R1$ are excluded. This proves that the optimal solution is obtained when $S1 < S2$.

2.4 Time Complexity[5]

If n is odd, then: minimum value of $k = n - 1 - (n-1)/2 - 1$
& maximum value of $k = n - 1$

If n is even, then: minimum value of $k = n - (n/2) - 1$
& maximum value of $k = n$

remaining nodes = $\{r\} = \{n\} - \{k\}$

number of slots = t = number of clusters = k

Time Complexity of the proposed algorithm $2^{\lceil n/2 \rceil} + m \log_2 m$

Time Complexity of Exhaustive Search Algorithm = $O(n!)$

Time Complexity of Dynamic Programming Algorithm = $O(n^2 2^n)$

In order to illustrate the difference in time between the different algorithms, consider the case where $n = 20$:

The time of Exhaustive Search Algorithm is = $2.432902 * 10^{18}$

The time of Dynamic Programming Algorithm is = 41,943,040

The time of the proposed algorithm is = 4800

Symbols

A	vector of needed elements (dimension m).
m	number of needed elements [$= n(n-1)/2$].
S	vector of heads of arcs.
E	vector of tails of arcs.
R	a temporary vector for building the tour.
U	usage indicator vector.
t	number of clusters.
k	number of nodes corresponding to the first $(n-1)/2$ low-value elements of A.
r	remaining nodes ($= n - k$).
$R1$	the set of the first $n/2$ elements of vector A.
$R2$	the set of n elements following $R1$.

$R3$ the set of $n/2$ elements just preceding $R4$.
$R4$ the set of the last $n/2$ elements of A.
$S1$ sum of elements of $R1$ and $R3$.
$S2$ sum of elements of $R2$.
NS set of nodes mentioned in vector S.
NE set of nodes mentioned in vector E.

References

[1] **Aho, A.A., Hopcroft, J.E.** and **Ullman, J.D.**, *Data Structures and Algorithms*, Addison-Wesely, Reading, Mass., pp. 323-335 (1983).
[2] **Garey, M.R.** and **Johnson, D.S.**, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, pp. 206-218 (1979).
[3] **Hadley, G.**, *Nonlinear and Dynamic Programming*, Addison-Wesely, Reading MA (1964).
[4] **Picard, J.C.** and **Queyranne, M.**, The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling, *Oper. Res.*, **26:** 86-110 (1978).
[5] **Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.** and **Shmoys, D.B. (eds.)**, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley, pp. 31-36 (1985).
[6] **Held, M.** and **Karp, R.M.**, A dynamic programming approach to sequencing problems, *SIAM J. Appl. Math.* **10:** 196-210 (1962).
[7] **Kruskal, J.B. Jr.**, On the shortest spanning subtree of a graph and the traveling salesman problem, *Proc. AMS* 7 (1): 48-50 (1956).
[8] **Lin, S.** and **Kernighan, B.W.**, A heuristic algorithm for the traveling salesman problem, *Oper. Res.* **21:** 498-516 (1973).
[9] **Knuth, D.E.**, *The Art of Computer Programming*, vol. 3, Addison-Wesely, Reading MA, pp. 73-180 (1973).

خـــوارزمي لحـــل معضلـــة البائـــع المتجـــول

محمد مصطفى حامد

جامعة البحرين - مدينـة عيسى - البـــحرين .

المستخلص . يتلخص الهدف الرئيس من هذا البحث في تقديم حل خوارزمي لمعضلة البـائـع المتجول . ومن الجائز أن يكون الحل الذي يقدمه الخوارزم هو الحل الأمثل إذا توافرت بعض الشروط بين القيم التي تمثل تكاليف السفر بين المدن . ويحتوي هذا البحث على وصف كامل للخوارزم وحساب عنصر الوقت اللازم للتنفيذ . وبمقارنة وقت التنفيذ مع الخوارزميات المعروفة لحل هذه المعضلة ، فإننا نجد أن هذا الخوارزم يصل إلى الحل في زمن أقصر من نظرائـه .