최적분해법에 의한 최단경로계산

論 文
30~5~2

Shortest Paths Calculation by Optimal Decomposition

李 章 揆*
(Jang - Gyu Lee)

Abstract

The problem of finding shortest paths between every pair of points in a network is solved employing an optimal network decomposition in which the network is decomposed into a number of subnetworks minimizing the number of cut-set between them while each subnetwork is constrained by a size limit. Shortest path computations are performed on individual subnetworks, and the solutions are recomposed to obtain the solution of the original network. The method when applied to large scale networks significantly reduces core requirement and computation time. This is demonstrated by developing a computer program based on the method and applying it to 30-vertex, 160-vertex, and 273-vertex networks.

1. Introduction

A path in a network refers to a successive chain of links connecting a point called origin and another point called destination. The shortest path between the pair is defined, when each link in the network is associated with cost, as the path in which the sum of the costs of the links is a minimum. In this paper, finding shortest paths between every pair of points in a network is treated.

The shortest paths calculation is required in routing problems of transportation system, data communication network, large-scale integrated circuit design, and operations research. A good example is the routing problem associated with the demand actuated transportation system (or dial-a-ride system) [1]. The demand actuated transportation system is a personalized door-to-door public transportation service in which a fleet of small vehicles serve the customers without fixed schedules or routes. As a custome request arrives, a vehicle is assigned to pick u and deliver the customer to its desired destina tion although not necessarily by a direct path While moving on its provisional route, the vehicl makes detours to pick up and drop off othe passengers as long as all customers are picke up and delivered within reasonable time period The driver of the vehicle proceeds on a stop-by stop basis knowing only his next stop at an time provided by a computer which stores th complete, tentatively planned tour of each vehicl at all times. In order to plan the tour, the com puter has to find shortest paths repeatedly. Thi is typically done by retreating necessary element from a shortest paths matrix which is prepare off-line and contains all the shortest paths infor mation of the serving area network. In thi network, street corners and street segments ar represented by points and links, respectively The cost of a link represents the travelin distance or the traveling time; the former implic a fixed cost and the latter a time-varying cost.

Another application of the shortest paths cal culation can be found in the routing problem

* 正會員 : The Analytic Sciences Corporation Technical Staff.
接受日字 : 1981年 4月 2日

of data communication networks. Data communication networks of various kinds are currently in existence or in the process of being set up. Examples include large-scale computer networks (e.g., the ARPA network, the French Cyclades network), and multipurpose data networks set up or proposed by airline reservation systems, medical data networks, banking networks, educational networks, corporate communication networks, and information service networks [2]. The basic routing problem here is that of establishing a best continuous path, usually incorporating several links in a network, between any pair of source and destination points, along which messages are to be sent. In general, the routing is centrally determined by a supervisory program located in an interdata supervisory computer in order to route messages with minimum average time delay or response time. Typically, a shortest path algorithm is used to determine the appropriate path from source to destination over which messages are sent for any particular user. The path in newly selected each time a user comes into the network, while other users maintain their current connections. In this sense, the route selection process is similar to that used in the demand actuated transportation system; the critical difference in data communication networks is that the costs of the links may vary frequently, requiring frequent shortest path computations.

The shortest paths problem has been an active research topic, as indicated in the survey papers [3, 4], and resulting numerous algorithms. In earlier algorithms, an exhaustive search method is used in which all possible paths between a pair are examined to find the shortest one. Algorithms by Dantzig [5] and Floyd [6] are of this type, both of which require exactly same number of computations though the search orders are different. Advanced algorithms are proposed by Tabourier [7] and Rosenthal [8] in which a sorting technique is introduced to the exhaustive search, resulting in a significant saving in the number of computations.

In spite of the significant saving, the above algorithms are limited by the network size that can be handled, because shortest paths calculation requires an operation of two N^2 matrices in core for a network of size N. For example, solving a 500 point (or vertex) network for shortest paths requires more than 500K words or two megabytes of core storage if one word uses four bytes. This core problem of large-scale networks has necessitated decomposition methods whose fundamental idea is that the large-scale network is broken into a number of subnetworks of smaller size. These subnetworks are then treated independently for solving shortest paths, and the solutions are recomposed to obtain the solution of the original network. Algorithms by Mills [9], Land and Stairs [10], and Hu [11] are in this category. The algorithms require much less core storage than those without decomposition and offers a faster solution time than the Dantzig's or Floyd's algorithms. Yet, the Mills algorithm [9] requests a restricted and difficult way of decomposing a network, and others [10, 11] use an arbitrary way of decomposition; all [9, 10, 11] employ methods similar to the Dantzig's or Floyd's algorithm to solve the individual subnetworks.

This paper presents a decomposition method which is different from the other decomposition methods in the following aspects:

- Decompose a network in an optimal way to reduce the number of shortest path computations
- Employ a sorting technique in solving individual subnetworks for shortest paths.

This method uses less core storage and its solution time for large-scale networks is faster than any other algorithm discussed in this paper.

The above claim is demonstrated in the following sections. Beginning from the definitions of the related terminologies, some preliminary algorithms including Dantzig's, Rosenthal's, and an optimal network decomposition algorithm are given. A detailed description and proof of the proposed algorithm are presented, followed by exercise results of the three algorithms-- Dantzig's, Rosenthal's, and the proposed--for a number of networks of various sizes.

2. Definitions

The underlying mathematical concept with the shortest paths problem is a network which is defined as a set of points called vertices and a set of directed lines called arcs. Each arc joins two vertices with a specified direction, and it can be denoted by

$$a_{ij}=[v_i, v_j]$$

Such an arc a_{ij} is said to be incident from the vertex v_i to the vertex v_j. Formally, a network G is defined as a pair

$$G=(V, A)$$

where V represents the vertex set and A the arc set. Further, a cost is defined for each arc as an associated real number with it.

A path is a chain of similarly directed arcs where the two subsequent arcs in the path share a common vertex; the corresponding sequence of vertices is another way of specifying the path. The total cost of a path is the sum of costs of its constituent arcs. The shortest path between two vertices is the path whose total cost is less than any other path between them in a given network. The shortest paths problem considered in this paper is a problem of finding shortest paths between all pairs of vertices from the network.

To formally describe the problem, it is convenient to define the cost, minimum cost, and successor matrices associated with a given network $G=(V, A)$. The cost matrix

$$C=[c_{ij}]$$

is the matrix whose ij^{th} element c_{ij} equals the cost of the arc connecting the vertex v_i to the vertex v_j if v_i is incident to v_j; otherwise, c_{ij} equals infinity. The minimum cost matrix

$$D=[d_{ij}]$$

is the matrix whose ij^{th} element d_{ij} represents the total cost of the shortest path from v_i to v_j. The successor matrix

$$S=[s_{ij}]$$

is the matrix whose ij^{th} element s_{ij} is the vertex number which is adjacent to v_i in the shortest path from v_i to v_j. Using these representations, the shortest paths problem is to calculate D and S from a given C. Upon calculation of D and S, for any pair of vertices, the minimum cost of the shortest path is obtained from D and the route is obtained from S.

A subnetwork $G_i=(V_i, A_i)$ of a network $G=(V, A)$ is defined as a network formed by a subset of vertices $V_i \subset V$ together with a set of all arcs of A that join any two vertices of V_i. In the network $G=(V, A)$, a set of vertices V_c $(V_c \subset V)$ can be found whose removal with all the incident arcs from G generates completely separated subnetworks, and it is called a cut-set. A process of finding the cut-set in G is called network decomposition.

3. Preliminary Algorithms

Before the development of the proposed algorithm is introduced, the Dantzig's and Rosenthal's algorithms and an optimal network decomposition algorithm are outlined in this section. These algorithms are utilized later by the proposed algorithm.

DANTZIG'S ALGORITHM

The Dantzig algorithm finds all shortest paths in a network, and it is summarized as follows: [5]

Step 1. Initialize the minimum cost matrix $[d_{ij}]=[c_{ij}]$ and the successor matrix $[s_{ij}]=[j]$, where $[j]$ represents a matrix in which all the elements of j^{th} column are j.*

Step 2. For $n=3, 4, \cdots, N$, do Steps 3 and 4. N is the total number of vertices in the network.

Step 3. For $j=1, 2, \cdots, n-1$ and for $i=1, 2, \cdots, n-1$, where $i \neq j$,

(A) $d_{in}=\min[d_{in}, d_{ij}+d_{jn}]$

$s_{in}=s_{ij}$ if d_{in} is replaced by $d_{ij}+d_{jn}$.

(B) $d_{ni}=\min[d_{ni}, d_{nj}+d_{ji}]$

$s_{ni}=s_{nj}$ if d_{ni} is replaced by $d_{nj}+d_{ji}$.

Step 4. For $j=1, 2, \cdots, n-1$ and for $i=1, 2, \cdots, n$

* Throughout the algorithm, equal signs are used in the sense of computer language.

-1, where $i \neq j$,

$d_{ij}=\min[d_{ij},\ d_{in}+d_{nj}]$

$s_{ij}=s_{in}$ if d_{ij} is replaced by $d_{in}+d_{nj}$.

The idea of the algorithm is to perform an xhaustive search inductively starting from two ertex subnetwork and adding one vertex at a ime in order to find shortest paths in the sub-ietwork. When the subnetwork covers all the ertices in the original network, the solution is inal. To explain the algorithm further, let G_k lenote a network obtained from G by deleting ertices v_{k+1}, $v_{k+2}, \cdots, v_N$ and all incident arcs, .e., G_k is a subnetwork of G composed of verti-es v_1, $v_2, \cdots, v_k$. When the main loop for n is ompleted, $[d_{ij}]$ and $[s_{ij}]$ are the minimum cost nd successor matrices of G_n. In computing the $d_{ij}]$, for instance in Step 3 (A), the element d_{in} nust be either c_{in} or else for some j a shortest ath d_{ij} in G_{n-1} followed by d_{jn}. The (B) step is he same in the reverse direction. Once all d_{in} nd d_{ni} are found, d_{ij} for G_n should be either the ame as G_{n-1} or else $d_{in}+d_{nj}$. Whenever a new ath is introduced for $[d_{ij}]$, $[s_{ij}]$ must be updated ccordingly.

ROSENTHAL'S ALGORITHM

The Rosenthal algorithm [6] is an extension f Dantzig's algorithm, and Step 3 is modified s follows:

Step 3. For $j=1,2,\cdots,n-1$ and for $i=1,2,\cdots,n-1$, where $i \neq j$,

3.1 (A) Sort $\{c_{in} : i<n\}$ and denote the result as $c_{i_1 n} \leq c_{i_2 n} \leq \cdots \leq c_{i_{n-1} n}$.

3.2 (A) Set $d_{i_1 n}=c_{i_1 n}$ and initialize a linear list $S_A=\{i_1\}$.

3.3 For $k=2,3,\cdots,n-1$,

3.3.1 (A) $d_{i_k n}=\min_{j \in S_A}[c_{i_k n},\ d_{i_k j}+c_{jn}]$

3.3.2 (A) $s_{i_k n}=s_{i_k j}$ if $d_{i_k n}$ is replaced by $d_{i_k j}+c_{jn}$.

3.3.3 (A) If $d_{i_k n}=c_{i_k n}$, set $S_A=S_A \cup \{i_k\}$.

The otner steps remain unchanged, and, for simplicity, (B) operations which are just reverse direction of (A) are not shown. The idea of this algorithm is to sort newly introduced arcs and to save redundant computations. In the algorithm, S_A contains all i_j which must be considered when finding the shortest path $d_{i_k n}$. If in 3.3.3 (A) $d_{ijn} \leq c_{ijn}$, it is not needed to go from the vertex i_j directly to n, and hence this arc may be ignored in all future computations. Computation time savings over Dantzig's algorithm is rather significant for large-scale sparse networks, as demonstrated in a later section.

4. Optimal Network Decomposition

An algorithm based on the optimal network decomposition theorem of Ref. [12] is presented in this subsection. The algorithm decomposes a network into a number of smaller subnetworks with a minimum number of cut-set and a size limit for each subnetwork.

Step 1. Given the total number of vertices N in a network and a size constraint K for each subnetwork, calculate the minimal number of subnetworks given by

$$\bar{m}=\left[\frac{N-1}{K}\right]$$

where $\left[\frac{N-1}{K}\right]$ indicates the closest integer no smaller than $\frac{N-1}{K}$

Step 2. For m subnetworks, do Steps 3~6. Initially $m=\bar{m}$.

Step 3. Choose m initial vertices, one for each subnetwork.

Step 4. Form a boundary vertex set for each subnetwork such that the removal of the set isolates the subnetwork from the rest of the whole network. Any vertex contained in more than two boundary vertex sets becomes a member of cut-set and it is removed from the list of boundary vertices.

Step 5. Select a vertex from each boundary vertex set and add it to the corresponding subnetwork such that the number of vertices of the resulting boundary vertex set decreases maximum or increases minimum.

Step 6. Repeat Steps 4~5 until every vertex is assigned to a subnetwork or cut-set.

Step 7. Repeat Steps 3~6 to find the best solution. If a solution not violating the constraint is found, stop. Otherwise, increase m by 1 and go to Step 2.

5. Algorithm

Shortest paths calculation by optimal decomposition involves: 1) optimally decomposing a network into a number of subnetworks; 2) solving shortest paths for each subnetwork in consideration with its interconnectivity to others; and 3) calculating shortest paths between vertices of different subnetworks. The first task is taken care of by the optimal network decomposition. For the other two tasks, two theorems are first presented. Theorem 1 describes a sufficient condition that warrants the shortest paths solution of a subnetwork calculated independent of other subnetworks. Theorem 2 shows a methodology to calculate the shortest paths between vertices of different subnetworks provided that shortest paths of each subnetwork are known.

Theorem 1: Given $G=(V, A)$ and a subnetwork $G_i=(V_i, A_i)$ of G, define $\overline{G}_i=(V_i, \overline{A}_i)$ such that $V_i \cup V_i=V$ and $A_i \cup \overline{A}_i=A$, and $V_i \wedge V_i=V_x$, i.e., V_x is a cut-set between G_i and $\overline{G}_i$, and $\overline{G}_i$ represents the complement of G_i only overlapped by the cut-set. Then, shortest paths between any two vertices in G_i, whether all intermediate vertices of the shortest paths are in G_i or not, can be obtained by considering only the subnetwork G_i if conditional shortest paths between all pairs of V_x in $\overline{G}_i$ are known.

Proof: Consider a shortest path calculation between a pair v_p and v_q in G_i. If the shortest path lies entirely in G_i, it is sufficient to carry out the calculation considering only G_i.

Assume then that there are many subpaths in the shortest path which contain vertices of $\overline{G}_i$. This is shown symbolically in Fig. 1. Since both the starting vertex v_p and the terminal vertex v_q are in G_j, and V_x is a cut-set whose removal together with the incident arcs separates G_i from $\overline{G}_i$, any subpath that contains vertices of $\overline{G}_i$ must begin and end with vertices in V_x. In

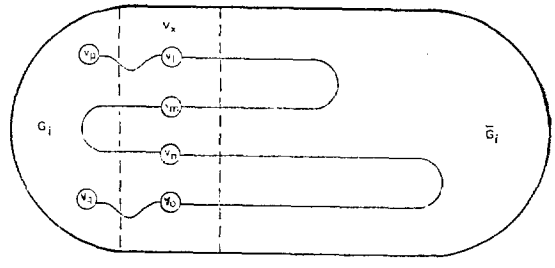

Fig. 1. Shortest Path Computation in a Subnetwork

Fig. 1, two such subpaths can be identified, and, by the assumption, their minimum costs and shortest routes are known. Treating the subpaths as additional arcs in G_i having the minimum costs, all the information needed to compute the shortest path between v_p and v_q are bound in G_i. The above reasoning holds for any vertex pair of G_i and any number of the subpaths. Thus, the shortest path between any vertex pair is obtainable considering only G_i as long as shortest paths between all pairs of V_x in $\overline{G}_i$ are known. *Q.E.D.*

Theorem 2: Let $G_i=(V_i, A_i)$ and $G_j=(V_j, A_j)$ be subnetworks of $G=(V, A)$, joined by a cut-set V_x, i.e., $V_i \wedge V_j=V_x$. Then, for $v_i \varepsilon V_i$, $v_j \varepsilon V_j$, and $v_i, v_j \varepsilon V_x$, the following holds,

$$d_{ij}=\min_{v_k \varepsilon V_x}(d_{ik}+d_{kj})$$

and

$$d_{ji}=\min_{v_k \varepsilon V_x}(d_{jk}+d_{ki})$$

where d_{ij} represents the shortest path from the vertex v_i to v_j, etc.

Proof: A path from a vertex of G_i to another vertex of G_j must pass at least one vertex in V_x because of the definition of cut-set. In computing the shortest path between these two vertices, if a minimum is taken over all vertices in V_x, then it is the minimum. Q.E.D.

Based on the optimal network decomposition algorithm and the above two theorems, the algorithm calculating shortest paths by optimal decomposition is now given.

Step 1. Decompose a given network using the optimal network decomposition algori

thm, and obtain the decomposed cost and initial successor matrices.

Step 2. Compute the shortest paths between all vertex pairs of cut-set considering all possible paths in the network.

Step 3. Compute the shortest paths for each subnetwork.

Step 4. Compute the shortest paths between all pairs of a vertex in a subnetwork and a vertex in another subnetwork.

Step 1 decomposes the network into a number of subnetworks minimizing the number of cut-set while each subnetwork has a limited size. In the decomposed network, any path from a vertex in a subnetwork to a vertex in another subnetwork must include at least one intermediate vertex in the cut-set. This does not exclude a possibility that the shortest path from a vertex in a subnetwork to another vertex in the same subnetwork passes through vertices in any number of other subnetworks as long as there is at least one cut-set vertex on a path between two vertices in different subnetworks.

The arrangement of the cost matrix according to the decomposition forms the bordered block-diagonal matrix shown in Fig. 2.

In Fig. 2, C_{kk} contains all the arcs in the subnetwork k except the ones incident with the cut-set. C_{kx} contains the arcs of subnetwork k incident to the cut-set, while C_{xk} contains those incident from the cut-set. Only shaded submatrices contain nontrivial elements and they constitute the bordered block-diagonal form. The corresponding successor matrix is handled similarly. For convenience, in the following descriptions of the algorithm, the successor matrix is not explicitly treated. However, it can be inferred from the shortest path algorithms presented in the previous section without much difficulty.

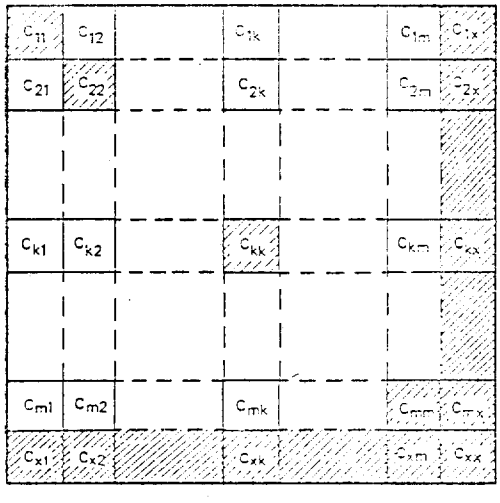

Fig. 2. Cost Matrix in Bordered Block-Diagonal Form

Shortest paths between all vertex pairs of the cut-set are obtained by repeated shortest paths computations beginning subnetwork 1 through subnetwork m. First perform shortest paths computation for subnetwork 1 starting from the following cost matrix

$$C_1=\begin{bmatrix} C_{11} & C_{1x} \\ C_{x1} & C_{xx} \end{bmatrix}$$

and resulting in

$$\tilde{D}_1=\begin{bmatrix} \tilde{D}_{11} & \tilde{D}_{1x} \\ \tilde{D}_{x1} & \tilde{D}_{xx}{}^1 \end{bmatrix}$$

where the tildes indicate incomplete minimum cost matrices obtained by considering only subnetwork 1. $\tilde{D}_{xx}{}^1$ is an incomplete minimum cost matrix of the cut-set. In computing the matrices, Rosenthal's algorithm is used.

Next, the cost matrix for subnetwork 2 is initialized as

$$C_2=\begin{bmatrix} C_{22} & C_{2x} \\ C_{x2} & \tilde{D}_{xx}{}^1 \end{bmatrix}$$

resulting in

$$\tilde{D}_2=\begin{bmatrix} \tilde{D}_{22} & \tilde{D}_{2x} \\ \tilde{D}_{x2} & \tilde{D}_{xx}{}^2 \end{bmatrix}$$

It should be noted, that for this subnetwork, $\tilde{D}_{xx}{}^1$ is used in the cost matrix instead of C_{xx}. $\tilde{D}_{xx}{}^2$ represents the incomplete minimum cost matrix of the cut-set taking into account subnetwork 1 and subnetwork 2. This procedure is continued. At subnetwork m, the initial cost matrix is given by

$$C_m=\begin{bmatrix} C_{mm} & C_{mx} \\ C_{xm} & \tilde{D}_{xx}{}^{m-1} \end{bmatrix}$$

and the resulted matrix is

$$\tilde{D}_m=\begin{bmatrix} \tilde{D}_{mm} & \tilde{D}_{mx} \\ \tilde{D}_{xm} & \tilde{D}_{xx}{}^m \end{bmatrix}$$

This completes Step 2, and $\tilde{D}_{xx}{}^m$ is now the complete minimum cost matrix of the cut-set.

The above resultant matrix actually becomes the minimum cost matrix of subnetwork m, following Theorem 1, since $\tilde{D}_{xx}^{m-1}$ represents the conditional shortest paths of the cut-set taking into account subnetworks 1 through $m-1$. The shortest path computation for the subnetwork m results in the solution of the subnetwork. Therefore, the above matrix may be written as

$$D_m=\begin{bmatrix} D_{mm} & D_{mx} \\ D_{xm} & D_{xx} \end{bmatrix}$$

representing the minimum cost matrix of the subnetwork m.

Shortest paths for the rest of $m-1$ subnetworks can be computed by employing D_{xx} as the conditional shortest paths of the cut-set. This may be done in any order, and again Rosenthal's algorithm is used. At the end of Step 3, the shortest paths for the shaded blocks in Fig. 2 are all found.

The rest of the blocks in the figure are taken care of in Step 4. The procedure described in Theorem 2 is employed to perform the step. For example, an element of D_{ij} is computed from the equation

$$d_{ij}=\min_{v_k \epsilon V_x}\{d_{ik}+d_{kj}\}$$

where d_{ik} and d_{kj} are the elements of D_i and D_j, respectively.

6. Exercise

The three shortest path algorithms described in the previous two sections are coded as computer programs, and they are exercised for 30-, 160-, and 273-vertex networks. The latter two are partial networks of the Haddonfield, New Jersey street network in which street corners and street segments represent vertices and arcs, respectively, and the distances represent costs of the arcs.

The Dantzig algorithm, though the formulation is different, requires exactly the same number of computer operations as Floyd's which is the most widely known by virtue of its simplicity. Therefore, the results of the Dantzig algorithm can represent those of the Floyd's algorithm as well. In Rosenthal's algorithm, the heapsort method [13] which is the most commonly used sorting method is employed for sorting. For the proposed algorithm, the 30-vertex network is decomposed into two subnetworks decomposed into two subnetworks; the 160-vertex network into two and three subnetworks; and the 273-vertex network into two through five subnetworks. The exercise results are shown in Tables 1 and 2.

The programs are written in Fortran and they are executed on the IBM 370/4341 machine. Two variables are chosen to compare the performances of the algorithms. The core requirement (Table 1) is the data block storage requirement in core to execute the program; the core storage required to compile the program source file is not included. The CPU time (Table 2) indicates the computer time required, to carry out the shortest path computation in the algorithm; the computer time spent for the data input and output are not included. Consequently, time spent for network decomposition in the proposed algorithm is not included, since the decomposition is considered part of data preparation. Once a network is decomposed, this step is not required unless the network configuration is changed. In general, network decomposition is performed once off-line, and then shortest paths for the network are repeatedly computed for various cost functions.

Table 1 shows that the Rosenthal algorithm requires slightly more core storage than Dantzig's algorithm. This is due to differences in core space used for sorting. The proposed algorithm consistently uses less core storage than the other two algorithms. In the 273-vertex network case, the proposed algorithm uses only one-third as much storage space as the Rosenthal and Dantzig algorithms. CPU time savings for the proposed algorithm are less apparent. For small size networks, Rosenthal's algorithm is quite effective; it improves the solution time by about four times over Dantzig's algorithm. The advantage of the proposed algorithm is clearly revealed as the network size increases and the network is divided

Table 1. Core Requirements for Shortest Path Algorithms

Network	Dantzig's Algorithm (K Bytes)	Rosenthal's Algorithm (K Bytes)	Proposed Algorithm (K Bytes)			
			2-Subnetwork	3-Subnetwork	4-Subnetwork	5-Subnetwork
273-Vertex Network	598.50	602.86	317.24	253.90	236.36	201.26
160-Vertex Network	206.16	208.72	117.74	100.86		
30-Vertex Network	7.52	8.00	6.29			

Table 2. CPU Time for Shorest Path Algorithms

Network	Dantzig's Algorithm (Seconds)	Rosenthal's Algorithm (Seconds)	Proposed Algorithm (Seconds)			
			2-Subnetwork	3-Subnetwork	4-Subnetwork	5-Subnetwork
273-Vertex Network	1,013.72	295.58	237.20	148.60	130.74	97.12
160-Vertex Network	200.67	42.61	43.82	30.18		
30-Vertex Network	1.06	0.31	0.50			

into more subnetworks. As shown in Table 2, the CPU time for the proposed algorithm can be as little as one-tenth that for the Dantzig algorithm.

7. Conclusion

A shortest path algorithm using optimal network decomposition is introduced in this paper. The algorithm first decomposes a network by minimizing the number of cut-set between subnetworks, while each subnetwork is limited by its size. Then, individual subnetworks are treated independently of others for shortest paths calculation. Savings in core requirement and CPU time of the algorithm results from the fact that it solves shortest paths for a series of smaller size networks instead of one large size network. The magnitude of the savings depends on the sizes of the individual subnetworks which, in turn depend on decomposition. The optimum criterion and the constraint of the optimal network decomposition work for the best result.

The proposed algorithm, when applied to large-scale networks, greatly reduces the core requirement and solution time in comparison with other algorithms. In Tables 1 and 2, the core requirements and solution times for the proposed algorithm are shown for a serial mode operation, i.e., all the minimum cost and successor matrices of subnetworks are stored in a core block and the shortest paths calculation is done for a time. If it is performed in a parallel mode, i.e., the matrices of each subnetwork are handled by a string of computed in parallel (e.g., using a multi-process microprocessor system), the core requirement and solution time will be further reduced by about 1/m when the network is decomposed into m subnetworks.

The comparatively small core requirement of the proposed algorithm results in a computer cost reduction since a smaller size computer can be used to compute shortest paths. Further, if a parallel processing technique using a number of micropocessors is adopted, the cost reduction will be very large. It is especially significant when a dedicated computer system is employed for a real-time calculation of shortest paths, as in the case of some dynamic systems. Fast solution time is also essential for those cases. For example, if

arc cost represents travel time in a transportation system, frequent shortest paths calculations are desirable as the traffic volume is changed affecting the travel time. In data communication network, frequent shortest paths calcualtions are required as users come in and out of system, thereby producing fluctuating data volume in each communication link. For these applications, the proposed algorithm can provide an economical way of solving shortest paths through its speed and reduction in computer cost.

References

1. Vogt, W.G., Mickle, M.H., Aldermeshian, H., Amir-Ebrahimi, K., Lee, J.G., Trygar, T.A., and Wright, G.L.; "Comparison of Variable and Fixed Routes for Demand Actuated Transportation Systems," Technical Report for UMTA Grant No. PA-06-0030, University of Pittsburgh, Department of Electrical Engineering, Pittsburgh, PA, August 1974.
2. Schwartz, M.; Computer-Communication Network Design and Analysis, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.
3. Dreyfus, S.E.; "An Appraisal of Some Shortest-Path Algorithms," Operations Research, Vol. 17, 1969, pp. 395~412.
4. Pierce, A.R.; "Bibliography on Algorithms for Shortest Path, Shortest Spanning Tree, and Related Circuit Routing Problems (1956~1974)," Networks, Vol. 5, 1975, pp. 129~149.
5. Dantzig, G.B.; "On the Shortest Route through a Network," Management Science, Vol. 6, 1960, pp. 187~190.
6. Floyd, R.W.; "Algorithm 97: Shortest Path," Comm. ACM, Vol. 5, 1962, p. 345.
7. Tabourier, Y.; "All Shortest Distances in a Graph. An Improvement to Dantzig's Inductive Algorithm," Discrete Mathematics, Vol. 4, 1973, pp. 83~87.
8. Rosenthal, A.; "On Finding Shortest Paths in Nonnegative Networks," Discrete Mathematics, Vol. 10, 1974, pp. 159~162.
9. Mills, G.; "A Decomposition Algorithm for the Shortest-Route Problem," Operations Research, Vol. 14, 1966, pp. 279~291.
10. Land, A.H., and Stairs, S.W.; "The Extension of the Cascade Algorithm to Large Graphs," Management Science, Vol. 14, No. 1, September 1967, pp. 29~33.
11. Hu, T.C.; "A Decomposition Algorithm for Shortest Paths in a Network," Operations Research, Vol. 16, No. 1, 1968, pp. 91~102.
12. Lee, J.G., Vogt, W. G., and Mickle, M.H., "Optimal Decomposition of Large-Scale Networks," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-9, No. 7, July 1979, pp. 369~375.
13. Knuth, D.E.; The Art of Computer Programming, Vol. 3, Addison-Wesley Pub. Co., Reading, Mass., 1973.