MANAGEMENT SCIENCE
Vol. 17, No. 11, July, 1971
Printed in U.S.A.

FINDING THE K SHORTEST LOOPLESS PATHS IN A NETWORK*

JIN Y. YEN†

University of Santa Clara

This paper presents an algorithm for finding the K loopless paths that have the shortest lengths from one node to another node in a network. The significance of the new algorithm is that its computational upper bound increases only linearly with the value of K. Consequently, in general, the new algorithm is extremely efficient as compared with the algorithms proposed by Bock, Kantner, and Haynes [2], Pollack [7], [8], Clarke, Krikorian, and Rausan [3], Sakarovitch [9] and others. This paper first reviews the algorithms presently available for finding the K shortest loopless paths in terms of the computational effort and memory addresses they require. This is followed by the presentation of the new algorithm and its justification. Finally, the efficiency of the new algorithm is examined and compared with that of other algorithms.

1. Introduction

There are two types of K-shortest-paths network problems. The first is to find K paths from the origin to the sink that have the shortest lengths, in which *loops are allowed*. The available algorithms for solving this type of problem are proposed by Hoffman and Pavley [6], Bellman and Kalaba [1], Sakarovitch [9] and others. The second type of problem is to find K paths from the origin to the sink that have the shortest lengths, in which *no loops are allowed*. The available algorithms for solving this type of problem are proposed by Bock, Kantner, and Haynes [2], Pollack [7], Clarke, Krikorian, and Rausan [3], Sakarovitch [9] and others.

The purpose of this paper is to present a new algorithm for solving the second type of problem. The significance of the new algorithm is that the computational upper bound of the algorithm increases linearly with the value of K. Consequently, in general, the new algorithm is much more efficient than the other available algorithms.

In this paper, first the efficiencies of the available K-shortest-loopless-paths algorithms are examined in terms of the computational effort and memory addresses they require. This is followed by a presentation of the new algorithm with its justification. Finally, the efficiency of the new algorithm is examined and compared to that of the other algorithms.

2. Review of Available Algorithms

There are several algorithms presently available for solving a K-shortest-loopless-paths problem in an N-node network. The following is a brief review of these algorithms and their efficiencies in terms of the computational effort and memory addresses they require. Note that in the following, unless specified otherwise, "a network" means a network without negative loops, "the K shortest paths" means K loopless paths from the origin to the sink that have the shortest lengths, and "the Kth shortest path" means the last of "the K shortest paths."

(1) In [2] Bock, Kantner, and Haynes introduce an enumeration procedure for finding the K shortest paths in a network. Their procedure enumerates all possible paths from the origin to the sink, then sorts from these the K paths that have the shortest lengths.

* Received November 1969; revised October 1970, December 1970.

† The author is very grateful to Professor C. West Churchman, Stuart E. Dreyfus, and the author's wife Wendy for their discussions and moral support. The comments of the anonymous referees are also greatly appreciated.

This algorithm has two main disadvantages. One is that the algorithm requires very large numbers of computations and memory addresses. The other is that the algorithm requires as much effort to solve a problem in which K is small as to solve a problem in which K is large. Therefore unless K is extremely large, e.g., $K = (N - 1)!$, the use of this algorithm cannot be recommended.

(2) In [7] Pollack introduces a procedure for finding the Kth shortest path in a network. To find the Kth shortest path this procedure first obtains $K - 1$ shortest paths. Then the distance of each arc in each of the 1st, 2nd, $\cdots$, $(K - 1)$st shortest paths is set, in turn, to infinity. The shortest-path problem is solved for each such case. The best of these resulting shortest paths is the desired Kth shortest path.

Pollack's algorithm can be considered the most applicable among the algorithms reviewed in this section because it has the lowest computational upper bound when K is not large (say, 5 or more, depending on the value of N). Unfortunately, the number of computations required by this procedure increases *exponentially* with the value of K—e.g., if the first $K - 1$ shortest paths each contains m arcs, this algorithm has to solve m^{K-1} shortest-path problems in order to find the Kth shortest path. Therefore, unless K is small, Pollack's algorithm is computationally overburdening.

(3) In [3] Clarke, Krikorian, and Rausan introduce a branch-and-bound procedure for finding the K shortest paths in a network. Their procedure first finds the shortest path, then finds the K shortest paths from all paths that "branch" out from the shortest path.

The efficiency of this algorithm depends on the particular network. If it so happens that the second shortest path "branches immediately" from the first shortest path, the third shortest path "branches immediately" from the second shortest path, etc., this procedure can determine all K shortest paths very quickly. However, in general, this algorithm is likely to require a tremendous number of computations and memory addresses. Therefore the efficiency of this procedure is difficult to determine.

(4) In [9] Sakarovitch introduces a K-shortest-path algorithm. His algorithm first finds H, $H \geqq K$, shortest paths that may contain loops by a procedure similar to (but less efficient than) Hoffman and Pavley's algorithm [6]. Then the H paths are scanned for the K shortest paths that contain no loops.

The efficiency of this algorithm depends on the particular network. If it so happens that the H shortest paths obtained by Hoffman and Pavley's algorithm are loopless, Sakarovitch's algorithm can determine the K-shortest-loopless paths very quickly. However, it is difficult to specify a computational upper bound for this algorithm—e.g., if all the distances of the arcs in a network are extremely large except that the distances of the two direct arcs going back and forth between the origin and the sink are extremely small, the H shortest paths found by Hoffman and Pavley's algorithm consist of only the loops that go around the origin and the sink; consequently, it is difficult to specify a computational bound within which the K shortest loopless paths must be determined.

3. Notation and Definitions

In an N-node network, let

(i), $i = 1, 2, \cdots, N$, be the nodes where (1) is the origin and (N) is the sink;

(1)—(i)—$\cdots$—(j), $i \neq j \neq \cdots \neq 1$, be the path from (1) to (j), passing through (i), $\cdots$;

$d_{ij} \gtreqless 0$, $i \neq j$, be the distance of the direct arc from (i) to (j)—if this arc exists, d_{ij} is a finite number, otherwise, d_{ij} is considered equal to infinity;

$A^k = (1)$—(2^k)—(3^k)— $\cdots$ —$(Q_k{}^k) - (N)$, $k = 1, 2, \cdots, K$, be the kth shortest

path from (1) to (N) where (2^k), (3^k), $\cdots$, ($Q_k{}^k$) are respectively the 2nd, 3rd, $\cdots$, Q_kth node of the kth shortest path;

$A_i{}^k$, $i = 1, 2, \cdots, Q_k$, be a set of "deviations from A^{k-1} at (i)"—a "deviation from A^{k-1} at (i)" is the shortest of the paths that coincide with A^{k-1} from (1) to the ith node on the path and then deviate to a node that is different from any of the ($i+1$)st nodes of those A^j, $j = 1, 2, \cdots, k-1$, that have the same paths from (1) to the ith node as does A^{k-1}; and finally reaches (N) by a shortest subpath without passing any node that is already included in the first part of the path. Note that the $A_i{}^k$ is loopless and contains the same node no more than once;

$R_i{}^k$ be the *root* of $A_i{}^k$—the root of $A_i{}^k$ is the subpath of $A_i{}^k$ that coincides with A^{k-1}, i.e., (1)—(2^k)—$\cdots$—(i^k) in $A_i{}^k$;

$S_i{}^k$ be the *spur* of $A_i{}^k$—the spur of $A_i{}^k$ is the last part of $A_i{}^k$ that has only one node coinciding with A^{k-1}, i.e., (i^k)—$\cdots$—(N) in $A_i{}^k$.

4. The New Algorithm and Its Justification

The new algorithm that finds the K shortest path is as follows:

Iteration 1. *To determine* A^1.

Determine A^1 by an efficient shortest-path algorithm—by Yen's algorithm [12] if $d_{ij} \geqq 0$; by Yen's algorithm [11] if $d_{ij} \gtreqless 0$. (*Remark.* Yen's algorithm [12] is a newly developed algorithm which finds the lengths of all shortest paths from a fixed node to all other nodes in an N-node nonnegative-distance network. This new procedure requires only $\frac{1}{2}N^2$ additions and N^2 comparisons—which is less than the number of operations required by other available algorithms. See [12] for details.)

Note that when there are negative loops in the network (which is detected by Yen's algorithm [11]), this K-shortest-path algorithm has to be terminated. This is because there is no satisfactory algorithm presently available for finding the shortest-loopless paths in a network with negative loops, consequently, this K-shortest-paths algorithm is no longer applicable for solving the problem.

However, when there are no negative loops in the network, we should have obtained at least one path that has the shortest length. If we have K or more such paths, we are done. If we have less than K and more than one paths, we assign any arbitrary one of these paths to be A^1 and store it in List A (the list of k-shortest paths); the rest of these paths are stored in List B (the list of candidates for ($k+1$)st shortest paths). Otherwise, if we have only one such path, it is A^1 which is to be stored in List A.

Iteration k ($k = 2, 3, \cdots K$). *To determine* A^k.

In order to find A^k, the shortest paths $A^1, A^2, \cdots, A^{k-1}$ must have been previously determined. A^k is then found as follows:

I. For each of $i = 1, 2, \cdots, Q_{k-1}$, do the following:

(a) Check if the subpath consisting of the first i nodes of A^{k-1} in sequence coincide with the subpath consisting of the first i nodes of A^j in sequence for $j = 1, 2, \cdots, k-1$. If so, set $d_{iq} = \infty$—where (q) is the ($i+1$)st node of A^j; otherwise, make no changes. Then go to Step (b).

Note that d_{iq}'s are set to ∞ for computations in iteration k only. They should be replaced by their original values before iteration $k+1$ starts.

(b) Apply a shortest-path algorithm to find the shortest path from (i) to (N), allowing it to pass through those nodes that are not yet included in the path. Note that the subpath from (1) to (i) is $R_i{}^k$, the root of $A_i{}^k$; and the subpath from (i) to (N) is

S_i^k, the spur of A_i^k. Note also if there are more than one subpaths from (i) to (N) that have the minimum length, take any arbitrary one of them and denote it by S_i^k.

(c) Find A_i^k by joining R_i^k and S_i^k. Then add A_i^k to List B.

Note that it is necessary to store only the $K - k + 1$ shortest A_i^k's in List B.

II. Find from List B the path(s) that have the minimum length.

If the path(s) found plus the path(s) already in List A exceed K, we are done. Otherwise, denote this path (or an arbitrary one, if there are more than one such paths) by A^k and move it from List B to List A—leaving alone the rest of the paths in List B. Then go to iteration $k + 1$.

Note that the above algorithm is developed from an obvious fact [3], [5] that A^k is a deviation from A^j, $j = 1, 2, \cdots, k - 1$. More precisely, A^k must coincide with A^j, $j = 1, 2, \cdots, k - 1$, for the first $m \geqq 1$ node(s) then deviates to a different node and finally arrives at the sink without passing each node more than once. Therefore to obtain A^k it is only necessary to look for all shortest deviations from the A^j's, then scan from these deviations the one that has the shortest length.

As shown in §4, in iteration k, Step I(a) of the algorithm sets d_{iq}'s equal to ∞ to force A^{k-1} to deviate at each node on the path—without allowing the deviations to take any path that have a length shorter than A^{k-1}. This is followed by Steps I(a) and I(c) which find the shortest deviations of A^{k-1} that are different from A^j, $j = 1, 2, \cdots, k - 1$. Finally in Step II the A^k is selected from all possible candidates in List B. Therefore the A^j, $j = 1, 2, \cdots, K$, thus obtained by the iterative procedure are the K shortest-loopless paths from the origin to the sink.

5. The Efficiency of the New Algorithm

The efficiency of an algorithm can be represented by the number of operations and the number of memory addresses required by the algorithm to solve the problem. The major operations and the memory addresses required by the new algorithm are as follows.

Number of operations

As shown above, in iteration k, the algorithm requires approximately the following major operations:

Step I(a). qKN, $0 < q \leqq 1$, comparisons, which is negligible as compared to operations required in Step I(b).

Step I(b). A. When $d_{ij} \geqq 0$: $\frac{1}{2}(N-1)^2 + \frac{1}{2}(N-2)^2 + \cdots \doteq \frac{1}{6} \cdot qN^3$ additions, and $(N-1)^2 + (N-2)^2 + \cdots \doteq \frac{1}{3} \cdot qN^3$ comparisons.

B. When $d_{ij} \gtreqless 0$: $\frac{1}{4}(N-1)^3 + \frac{1}{4}(N-2)^3 + \cdots \doteq \frac{1}{16} \cdot qN^4$ additions and comparisons.

Steps I(c) *and* II. Negligible number of operations as compared with Steps I(a) and I(b).

Therefore up to iteration K, the algorithm requires approximately a total of $\frac{1}{6} \cdot qKN^3$ additions and $\frac{1}{3} \cdot qKN^3$ comparisons, if $d_{ij} \geqq 0$; and $\frac{1}{16} \cdot qKN^4$ additions and $\frac{1}{16} \cdot qKN^4$ comparions, if $d_{ij} \gtreqless 0$. These totals are very small when compared with the number of operations required by other algorithms.

Number of memory addresses

The algorithm requires approximately $N^2 + KN$ addresses to store the d_{ij}'s, List A, List B, and some negligible number of intermediate data.

TABLE 1.

Comparisons of the efficiencies of K-shortest-paths algorithms

Algorithm	Type of Network	Approximate Number of Necessary			Approximate Upper Bound of Necessary Operations	Ratio of Other Algo's Upper Bound to Yen's Upper Bound
		Additions	Comparisons	Memory Addresses		
Yen's	$d_{ij} \geqq 0$	$\frac{1}{6}qKN^3$	$\frac{1}{3}qKN^3$	$N^2 + KN$	$\frac{1}{2}KN^3$	1
	$d_{ij} \gtrless 0$	$\frac{1}{16}qKN^4$	$\frac{1}{16}qKN^4$	$N^2 + KN$	$\frac{1}{8}KN^4$	1
Pollack's	$d_{ij} \geqq 0$	$\frac{1}{2}qN^{K+1}$	qN^{K+1}	$N^2 + KN$	$\frac{3}{2}N^{K+1}$	$3N^{K-2}/K$
	$d_{ij} \gtrless 0$	$\frac{1}{4}qN^{K+2}$	$\frac{1}{4}qN^{K+2}$	$N^2 + KN$	$\frac{1}{2}N^{K+2}$	$4N^{K-2}/K$
Bock, Kantner and Haynes'	$d_{ij} \gtrless 0$	$\sum_{i=1}^{N-2} \binom{N-2}{i} \cdot i!$	$\sum_{i=1}^{N-2} (i!) \cdot \log_2 (i!)$	$N^2 + KN$	$\sum_{i=1}^{N-2} [\binom{N-2}{i} + \log_2 (i!)] \cdot i!$	Very large
Clarke, Krikorian and and Rausan's	$d_{ij} \gtrless 0$	Difficult to specify				
Sakarovitch's	$d_{ij} \gtrless 0$	Difficult to specify				

Note. $0 < q \leqq 1$.

The following is a table of comparisons of the efficiency of the new algorithm with the other algorithms available for solving the problem.

References

1. Bellman, R. and Kalaba, R. "On kth Best Policies," *J. of SIAM*, Vol. 8, No. 4 (December 1960), pp. 582–588.
2. Bock, F., Kantner, H. and Haynes, J., *An Algorithm (The rh Best Path Algorithm) for Finding and Ranking Paths Through a Network*, Research Report, Armour Research Foundation, Chicago, Illinois, November 15, 1957.
3. Clarke, S., Krikorian, A. and Rausan, J., "Computing the N Best Loopless Paths in a Network," *J. of SIAM*, Vol. 11, No. 4 (December 1963), pp. 1096–1102.
4. Dijkstra, E. W., "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, Vol. 1 (1959), pp. 269–271.
5. Dreyfus, S. E., "An Appraisal of Some Shortest-Path Algorithms," *Opns. Res.*, Vol. 17, No. 3 (May-June 1969), pp. 395–412.
6. Hoffman, W. and Pavley, R., "A Method for the Solution of the Nth Best Problem," *J. of ACM*, Vol. 6, No. 4 (October 1959), pp. 506–514.
7. Pollack, M., "The kth Best Route Through a Network," *Opns. Res.*, Vol. 9, No. 4 (1961), pp. 578.
8. ——, "Solutions of the k Best Route Through a Network—A Review," *J. of Math. Anal. and Appl.*, Vol. 13, No. 3 (August–December 1961), pp. 547–559.
9. Sakarovitch, M., *The k Shortest Routes and the k Shortest Chains in a Graph*, Opns. Res. Center, University of California, Berkeley, Report ORC-32, October 1966.
10. Yen, J. Y., "An Algorithm for Finding Shortest Routes from All Source Nodes to a Given Destination in General Networks," *Quart. of Applied Math.*, Vol. 27, No. 4 (January 1970), pp. 526–530.
11. ——, *A Shortest Path Algorithm*, Ph.D. dissertation, University of California, Berkeley, 1970.
12. ——, "Finding All Shortest Paths from a Fixed Node in Non-Negative-Distance Networks," submitted to *J. of ACM*.