696567

MEMORANDUM
RM-5433-1-PR
SEPTEMBER 1963

AN APPRAISAL OF SOME SHORTEST-PATH ALGORITHMS

S. E. Dreyfus

DDC
RECEIVED
SEP 26 1963
C

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation

MEMORANDUM
RM-5433-1-PR
SEPTEMBER 1968

AN APPRAISAL OF SOME SHORTEST-PATH ALGORITHMS

S. E. Dreyfus

This research is supported by the United States Air Force under Project RAND—Contract No. F44620-67-C-0045—monitored by the Directorate of Operational Requirements and Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of the United States Air Force.

DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

The RAND Corporation
1700 MAIN ST • SANTA MONICA • CALIFORNIA • 90406

This Rand Memorandum is presented as a competent treatment of the subject, worthy of publication. The Rand Corporation vouches for the quality of the research, without necessarily endorsing the opinions and conclusions of the authors.

Published by The RAND Corporation

PREFACE

This Memorandum treats five different problems involving the determination of the shortest path through a discrete network. Previous important results are reviewed, and misleading procedures are identified and (in some cases) modified. Conclusions are drawn and recommendations are made concerning efficient algorithms.

This work forms a part of RAND's continuing interest in problems and techniques in optimization theory.

The author, a member of the Industrial Engineering and Operations Research faculty at the University of California, Berkeley, is a consultant to the Computer Sciences Department of The RAND Corporation.

SUMMARY

This Memorandum treats five discrete shortest-path problems: 1) determining the shortest path between two specified nodes of a network; 2) determining the shortest paths between all pairs of nodes of a network; 3) determining the second, third, etc. shortest path; 4) determining of the fastest path through a network with travel times depending on the departure time; and 5) finding the shortest path between specified endpoints that passes through specified intermediate nodes. Existing good algorithms are identified while some others are modified to yield efficient procedures. Also certain misrepresentations and errors in the literature are demonstrated.

CONTENTS

I. INTRODUCTION

In the never-ending search for good algorithms for various discrete shortest-path problems, some authors have apparently overlooked or failed to appreciate previous results. Consequently, certain recently reported procedures are inferior to older ones. Also, occasionally, inefficient algorithms are adapted to new, generalized problems where more appropriate modifiable methods already exist. Finally, the literature contains some erroneous procedures. This Memorandum briefly considers various versions of discrete-path problems in the light of known results and some original ideas.

Our observations are, of course, by no means definitive or final. However, it is hoped that our somewhat skeptical survey of current literature will put the interested reader on guard and perhaps save him, or his digital computer, considerable time and trouble. Since our objective is more to alert than to resolve conclusively, this Memorandum is informal and, at times, cryptic. We hope that even our most laconic remarks will prove enlightening for any reader deeply involved with the particular procedure.

Corresponding to almost any shortest-path algorithm, some special network structure exists for which the algorithm

is efficient. Consequently, to give meaning to our conclusions, we shall generally restrict our attention to problems in which every pair of nodes is connected by an arc (perhaps of infinite length), and shall develop bounds on the number of computational steps. One procedure is considered significantly superior to another when these bounds differ by a multiplicative factor involving N, the number of nodes. When the resulting formulas differ by only a multiplicative constant, the user's choice of algorithm should be determined by problem structure, computer configuration, and programming language.

II. THE SHORTEST PATH BETWEEN A SPECIFIED PAIR OF NODES

Given a set of N nodes, numbered arbitrarily from 1 to N, and the NxN matrix D, not necessarily symmetric, whose element d_{ij} represents the length of the directed arc connecting node i to node j, find the path of shortest length connecting node 1 and node N. Assume initially that $d_{ii} = 0$ and $d_{ij} \geq 0$. If no arc is directed from node i to node j, then $d_{ij} = \infty$; or, for purposes of digital computation, d_{ij} is taken large.

The computationally most efficient procedure was described first by Dijkstra [1] in 1959, and in 1960 by Whiting and Hillier.* The algorithm assigns tentative labels, which are upper bounds on the shortest distance from node 1, to all nodes; after the fundamental iterative step described below is repeated exactly once for each node, the tentative node labels are all permanent and represent shortest distances.

Initially, label node 1 with the permanent value zero, and tentatively label all other nodes infinity. Then, one by one, compare each node label except that at node 1 with the sum of the label of node 1 (i.e., 0) and the direct

*See Ref. 2, last paragraph beginning on p. 39.

distance from node 1 to the node in question. The smaller of the two numbers is the new tentative label.

Next, determine the smallest of the N-1 tentative labels and declare it permanent. Suppose that node k is the one permanently labelled. Then, one at a time, compare each of the N-2 remaining tentative node labels to the sum of (a) the label just assigned permanently to node k and (b) the direct distance from node k to the node under consideration. The smaller of the two numbers becomes the tentative label. Determine the minimum of the N-2 tentative labels, declare it permanent, and make it the basis of another modification of the remaining tentative labels of the type described above. When, after at most N-1 executions of the fundamental iterative step, node N is permanently labelled, the procedure terminates. (If the shortest paths from node 1 to all other nodes are desired, the fundamental iterative step must be executed exactly N-1 times.)

The optimal paths can easily be reconstructed if an optimal policy table (in this case, a table indicating the node from which each permanently labelled node was labelled) is recorded. Alternatively, no policy table need be constructed, since it can always be determined from the final

node labels by ascertaining which nodes have labels that differ by exactly the length of the connecting arc.*

The proof of the validity of the method is inductive, with the key step as follows. Suppose that, at a particular stage, the nodes are divided into two mutually exclusive and collectively exhaustive sets--Set 1 contains the permanently labelled nodes and Set 2 the temporarily labelled ones. The node labels of Set 1 are correct minimum distances from the source node. The node labels of Set 2 are the shortest distances from the source that can be attained by a path in which all except the terminal node belong to Set 1. Then the minimum-label node of Set 2--call it node i--can be transferred to Set 1, because if a shorter path to node i existed it would have to contain a first node that is currently in Set 2. However, that node must be farther away from the source, since its label exceeds that of node i. The subsequent use of node i to reduce the labels of adjacent nodes belonging to Set 2 restores to Set 2 the property assumed above.

This algorithm requires $\frac{N(N-1)}{2}$ additions and $N(N-1)$ comparisons to solve the problem--totaled over all steps,

---

*M. Bellmore points out, in a private communication, that if arcs of length 0 are permitted, special care must be taken to prevent possible cycling during the phase of the calculation used to deduce the path from the value table.

$\frac{N(N-1)}{2}$ additions and comparisons are necessary to compute tentative node labels and $\frac{N(N-1)}{2}$ comparisons are necessary to find the minimum label at each step. All steps are naturally and easily programmed except that of distinguishing which nodes are permanently and which tentatively labelled. Some computational experimentation indicates that an efficient method of distinguishing is to attach to each node an index number that changes from, say, 0 to 1 when a node label becomes permanent. When branching out from a just-permanently-labelled node, the index of the destination node is consulted as each outgoing arc is considered. If the index is zero, the temporary label of the destination is reduced, if appropriate. At the same time, memory cells designating the smallest temporary label encountered thus far during the branching and the associated destination node are modified, if appropriate. This programming device requires $(N-1)^2$ comparisons to consult the indices. Hence, a total of about $N^2/2$ additions and $2N^2$ comparisons are necessary.

Pollack and Wiebenson* describe and credit to Minty† the first (and hence of some historical interest) systematic, easily programmed, permanent-label-setting precursor

*See Ref. 3, p. 225.

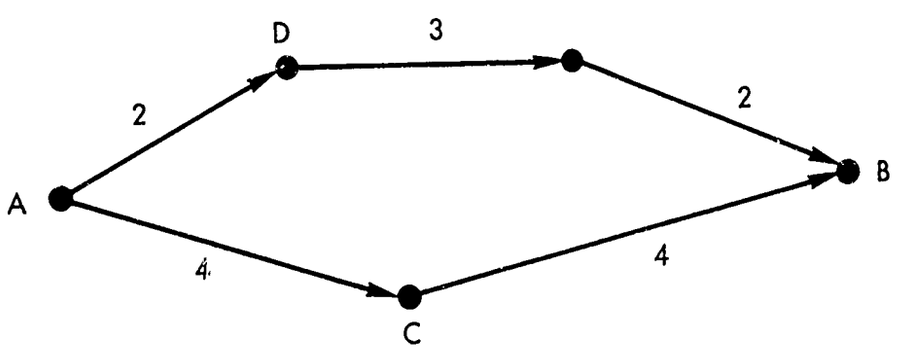
†Pollack and Wiebenson cite, without date, a private communication by Minty.

of the above method. That algorithm is probably due originally to Ford and Fulkerson [4], who developed it for a more general flow problem, of which the shortest-path problem is a special case requiring fastest flow of one item. The method requires approximately $N^3/6$ additions and comparisons for solving the shortest-path problem, and hence is not recommended.

The Minty-Ford-Fulkerson procedure can be accelerated--but not enough to compete in general with the Dijkstra algorithm--by use of a modification reported by Whiting and Hillier (the first method of Ref. 2) and Dantzig [5]. In their method, derived independently of each other and of the Minty and Ford-Fulkerson [4] works, outgoing arcs from each node are listed from shortest to longest. This obviates a search for the shortest arc out of each permanently labelled node at each step, and reduces the number of additions and comparisons from $\frac{N^3}{6}$ to $\frac{N^2}{2}$. While this number suggests that the method improves the Dijkstra procedure, it is misleading for two reasons. First, ordering the data as required by the method necessitates approximately $N^2 \log_2 N$ additional comparisons. Second, the method must delete arcs from lists as they are used; and also, each time a node is permanently labelled, the method must delete, from

all lists, arcs that lead into that node. This data modification requires about $3N^2$ comparisons, plus elaborate programming of list-processing procedures. Therefore, except perhaps for sparse networks with far fewer than N^2 arcs, the method is not recommended.

Some authors have proposed simultaneously fanning out from both endpoints as a means of reducing computation. Berge and Ghouila-Houri [6] falsely assert that when, for the first time, some node is permanently labelled in both fans, the optimal path results and goes through that node. Dantzig [7] is vague about his stopping procedure. In the problem depicted below, if the Dijkstra scheme is first used to permanently label the node nearest A reachable from A, then the node nearest B from which B can be reached, then the second closest to A, etc., the node C is permanently labelled both "out from A" and "into B" after two applications of the procedure at each end. Yet ACB is <u>not</u> the shortest path.



When the correct stopping procedure given cryptically by Nicholson [8] and explained clearly by Murchland [9] is used, the least upper bound on the computation required by a two-ended procedure exceeds that for the one-ended Dijkstra algorithm, contrary to assertions by Berge and Ghouila-Houri [6]. This happens because, as more nodes become permanently labelled in the Dijkstra procedure, a decreasing number of additions and comparisons are necessary to modify all tentative labels. As a result, determining the first $\frac{N}{2}$ permanent labels requires just over three-fifths the work of the complete solution. If the Nicholson stopping condition is satisfied long before $\frac{N}{2}$ nodes have been permanently labelled from each terminus, a savings may accrue; but, in a case where nearly all N nodes must be permanently labelled from either one end or the other, the two-ended procedure will prove inefficient.

All these methods require all elements of D to be non-negative. A related problem assumes that some d_{ij} are negative, but the sum of the d_{ij} around any loop is positive. Such data arise when arc numbers represent costs, and some arcs are profitable. (The problem has no solution if negative loops exist. Should negative loops exist but be excluded from admissible paths, no known algorithm is satisfactory.)

We present, first, a well-known algorithm that either solves the problem, with negative d_{ij}, in at most N^3 additions and comparisons, or detects the existence of a negative cycle in that number of steps; second, a recent improvement that halves the number of calculations; and third, a different procedure that is competitive. While we know of no better procedures, the disparity in methods and bounds indicates the probability of further improvements.

The basic procedure has been proposed, originally for problems with $d_{ij} \geq 0$, by Ford [10], Moore [11], Bellman [12], and undoubtedly others. This procedure repeatedly updates all node labels. For the initial condition given in Eq. (1), the node label $f_i^{(k)}$ represents the length of the shortest path that connects node 1 and node i and that contains k+1 or fewer arcs. Unlike the permanent-label-setting procedures recommended when all d_{ij} are non-negative, in this procedure no node labels are considered final until all are. The fundamental recursion is

$$f_i^{(k+1)} = \min_j [d_{ji} + f_j^{(k)}] \qquad (1)$$

$$f_i^{(0)} = d_{1i} \, .$$

For the case $d_{ij} \geq 0$, convergence occurs whenever $f_i^{(k)} = f_i^{(k+1)}$ for all i, or after N-2 iterations if the former situation occurs no sooner (since no shortest path contains more than N-1 arcs). If (N-2) iterations are required and if each pair of nodes is connected by an arc, then $(N-2)(N-1)^2$ additions and comparisons take place.

This method is inefficient for a positive-distance problem if two or more iterations of Eq. (1) are needed; and, unless all N-2 iterations are required, as many iterations will be necessary as the number of arcs in the shortest path from node 1 to node j, where node j is the node whose shortest path has the greatest number of arcs.

When this procedure is applied to a problem with some negative d_{ij}, either convergence will occur for $k \leq N-1$, indicating no negative cycles exist and the solution is optimal; or a change in some f_i will occur on the $(N-1)^{st}$ iteration, indicating the existence of a negative loop.

Yen [13] has recently reduced the computation by a factor of two. He suggests the recursion, for k=1,2,...,

$$f_i^{(2k-1)} = \min \left[\min_{j<i} \left(d_{ji} + f_j^{(2k-1)} \right) , f_i^{(2k-2)} \right] ,$$

$$i=1,\ldots,N,$$

$$f_i^{(2k)} = \min \left[\min_{j>i} \left(d_{ji} + f_j^{(2k)} \right) , f_i^{(2k-1)} \right] , \qquad (2)$$

$$i=N,\ldots,1,$$

with initial condition $f_i^{(0)} = d_{1i}$. Using new values of f as soon as they are determined, and processing the nodes alternately forwards and backwards, produces bounds on convergence the same as above; yet, minimizing over only nodes previously treated* necessitates only half as many additions and comparisons per iteration.

A scheme of Dantzig, Blattner, and Rao [14] is novel and, with a slight modification, efficient. It can detect a negative loop much more quickly than the above procedures, if such a loop exists; it is as fast, if no such loop exists. Unfortunately, the procedure is difficult to explain, prove, or program.

---

*Note that in Eq. (1) the minimization is over all j while in Eq. (2) it is over only either $j < i$ or $j > i$.

Suppose that, at iteration k+1, node numbers $f_i^{(k)}$ have been assigned to nodes 1 through k, each representing the length of the shortest path to node i that may pass through intermediate nodes 1 through k, but no others. The length of the shortest path to node k+1, using only nodes 1 through k+1, is found by

$$f_{k+1}^{(k+1)} = \min_{j} \left[f_j^{(k)} + d_{j,k+1} \right] . \qquad (3)$$

$$j=1,\ldots,k$$

Then the node numbers $f_j^{(k)}$, j=1,...,k, are reduced if, by introducing node k+1 into a path, a shorter distance results. This calculation involves a sub-iteration that first finds the node whose distance can be most reduced by introducing node k+1 into the path, then the node with the second-largest reduction, etc. This reduction is accomplished in a manner analogous to the Dijkstra procedure [1].* At iteration k+1, Eq. (3) involves a negligible k additions and comparisons. The updating, if carried out as in Ref. 1, requires at most $\frac{k^2}{2}$ comparisons with a flag indicating nodes

*Reference 14 gives a less efficient method, involving list processing and reordering.

already reduced, and $\frac{k^2}{2}$ calculations involving two additions and a comparison.* Hence, for N nodes, the method appears to involve at most $\frac{N^3}{3}$ additions and comparisons--slightly bettering the upper bound for the Yen [13] algorithm. Since, as indicated in the introduction, such improvements as reducing the number of computational steps from $\frac{N^3}{2}$ to $\frac{N^3}{3}$ are easily negated by computer hardware or programming language idiosyncracies, both of the latter two methods should be seriously considered for application.

In concluding our treatment of the problem of finding the shortest path between a specified source node and all other nodes of a network, let us briefly summarize the results of some computational experiments of Hitchner [15] involving various specially structured problems, all with the restriction that $d_{ij} \geq 0$. Hitchner compares one method that successively reduces node labels by considering neighboring nodes (in the spirit of the algorithm described by Eq. (1)) with four variations on the permanent-label-setting Dijkstra procedure. Three of the latter keep lists to avoid repeated minimization over the set of all non-permanently labelled nodes. He concludes that, for problems with only

*The sub-iteration terminates when no further reduction is found.

four arcs emanating from each node (such as in "ideal" city maps), the procedure based on Eq. (1) and one special list-keeping version of the Dijkstra method are superior (because they depend more on the number of arcs than nodes). For problems with 25 percent or more of the N(N-1) possible arcs present, the Dijkstra procedure (with no sophisticated list-processing adornments) out-performed all competitors.

III. THE SHORTEST PATHS BETWEEN ALL PAIRS OF NODES OF A NETWORK

Two somewhat different, but equally elegant and efficient, algorithms are recommended. One was published without comment as an obscure nine-line ALGOL algorithm in 1962 by Floyd [16], based on a procedure by Warshall [17], and was rediscovered and appropriately extolled in 1965 by Murchland [18]. The other was produced in 1966 by Dantzig [19]. Since both require exactly the same number of calculations--N(N-1)(N-2) additions and comparisons for the case $d_{ij} \geq 0$--are easily proved and programmed, and culminate a steady progression of successive improvements ([20], [12], [21], [22]; actually Ref. 16 precedes the inferior algorithm of Ref. 22), there is good reason to believe that they are definitive. While the reader should consult the primary sources cited above, we briefly describe the algorithms here.

The Floyd procedure [16] builds optimal paths by inserting nodes, when appropriate, into more direct paths. Starting with the NxN matrix D of direct distances, N matrices are constructed sequentially. The k^{th} such matrix can be interpreted as giving the lengths of the shortest allowable paths between all node pairs (i,j), where

only paths with intermediate nodes belonging to the set of nodes 1 through k are allowed. The $(k+1)^{st}$ matrix is constructed from the k^{th} by using the formula

$$d_{ij}^{(k+1)} = \min \left[d_{ij}^{(k)}, d_{i,k+1}^{(k)} + d_{k+1,j}^{(k)} \right], \qquad (4)$$

$$d_{ij}^{(0)} = d_{ij} .$$

Here, k, which is initially zero, is incremented by 1 after i and j have ranged over the values 1,...,N; and k = N-1 at termination.

To appreciate the rationale of the procedure, suppose the shortest path from node 8 to node 5 is 8 - 3 - 7 - 1 - 9 - 5. Iteration 1 will replace d_{79} by $d_{71} + d_{19}$; iteration 3 will replace the current value of d_{87} (which may or may not be the original value) by $d_{83} + d_{37}$ (the optimal value); iteration 7 will replace the current d_{89} by $d_{87} + d_{79}$, where these numbers are the optimal values as computed above; and iteration 9 will obtain for d_{85} the sum of d_{89} and d_{95}, when d_{89} is as computed at iteration 7. Hence, the correct shortest distance is obtained. (The above justification is, of course, not a proof.)

A minor modification, in case some d_{ij} are negative, detects negative loops, if any exist, and otherwise yields correct results. An additional advantage of this procedure is that N-1 additions and comparisons are easily circumvented whenever an element $d^{(k)}_{i,k+1}$ equals infinity in Eq. (4) (i.e., no path, with only nodes 1 through k as intermediate nodes, connects nodes i and k+1).* As in the case of the particular initial-terminal pair problem, an optimal policy table (matrix) associating with the initial-terminal node pair (i,j) the next node along the best path from i to j can be developed during the computation, or can be deduced from the final shortest-distance matrix.

Dantzig's scheme [19] generates successive matrices of increasing size. The k^{th} iteration produces a kxk matrix whose elements are the lengths of the shortest paths connecting nodes i and j, i=1,...,k, j=1,...,k, in which only nodes 1 through k may be intermediate nodes. Given the kxk matrix $D^{(k)}$ with elements $d^{(k)}_{ij}$ as defined above, compute $D^{(k+1)}$ as follows:

---

*The additional computation introduced in order to test for the presence of infinities adds about 5 percent to the computing time. For a sample problem with 10 nodes and 34 arcs and, hence, 56 infinities initially, the test yielded a 5 percent net improvement over no test. The amount of net improvement or degradation depends on the actual network configuration as well as the number of non-existent arcs.

1) Compute $d_{i,k+1}^{(k+1)}$ for i=1,...,k by

$$d_{i,k+1}^{(k+1)} = \min_{1\le j\le k} \left[d_{ij}^{(k)} + d_{j,k+1} \right] ,$$

and similarly compute $d_{k+1,i}^{(k+1)}$.

2) Compute $d_{ij}^{(k+1)}$ for i=1,...,k, j=1,...,k by

$$d_{ij}^{(k+1)} = \min \left[d_{ij}^{(k)}, d_{i,k+1}^{(k+1)} + d_{k+1,j}^{(k+1)} \right] .$$

That the $d_{ij}^{(k)}$ yielded by the above steps are as previously defined is obvious after a little thought. (Reference 19 gives a proof.) If all distances are positive, $d_{ii}^{(k)} = 0$ for all i and k. If not, $d_{ii}^{(k)}$ can be computed easily; and if any $d_{ii}^{(k)}$ is negative, a negative loop exists.

The Dantzig algorithm seemingly can not exploit non-existent arcs in a manner similar to Floyd's.

If the above algorithms, requiring N(N-1)(N-2) additions and comparisons, are indeed as efficient as possible, then the most efficient particular-pair algorithm must require at least (N-1)(N-2) such calculations (assuming, as was the case in Sec. II, that such procedures must--at least in the worst case--generate best paths from the

initial node to all other nodes). The best algorithm of Sec. II, that which required no elaborate data-preparation or list keeping, involves $N^2/2$ additions and $2N^2$ comparisons. Assuming additions and comparisons use equal amounts of computation time, theory gives, as a reasonable upper-bound estimate of potential future improvements for the particular pair-of-endpoints problem, a reduction in computation time of about 20 percent. Actual computational experiments indicate that computing optimal paths between all pairs of nodes by N applications of the Dijkstra method requires roughly one and one-half-times the time consumed by either algorithm specifically solving the all-pairs problem. Indexing operations account for the difference between theory and practice. Viewed from the perspective of a combinatorialist, the well is nearly dry. (Such may not be the case for the $\frac{N^3}{3}$ and $\frac{N^3}{2}$ procedures recommended above for the problem with negative distances.)

IV. DETERMINATION OF THE SECOND-SHORTEST PATH

It is occasionally desirable to know the value of the second- (and third-, etc.) shortest path through a network. For example, suppose that some complex quantitative (or even qualitative) feature characterizes paths, and the shortest path possessing this additional attribute is sought. By ignoring the special aspect in question and ordering paths from shortest to longest, the best path with the additional feature can sometimes be determined efficiently.

The analysis below initially considers the problem of determining the second-best path between a specified initial node 1 and a specified destination, N. Then it draws conclusions for more general problems. Two paths that do not visit precisely the same nodes in the same order are considered different. The discussion ignores ties by assuming, for simplicity, that all paths have different values. A path with a loop is considered an admissible path, and indeed such a path can be second best, even for problems with all $d_{ij} > 0$. Even node N may be visited twice along the second-best path.

The earliest good algorithm known to this author was proposed by Hoffman and Pavley [23]. A deviation from the

the shortest path was defined to be a path that coincides with the shortest path from its origin up to some node j on the path (j may be the origin or the terminal node), then deviates directly to some node k not the next node of the shortest path, and finally proceeds from k to the fixed terminal node via the shortest path from k. Reference 23 shows the second-shortest path between specified initial and terminal nodes to be a deviation from the shortest path.

To solve the problem posed above, first the shortest paths from all initial nodes to the specified destination are determined by means of any efficient algorithm. Then, all deviations from the shortest path between the specified origin and terminus are determined, evaluated, and compared, and the best noted. If the average node has M outgoing links, and the average shortest path contains K arcs, an average problem is solved in approximately MK additions and comparisons beyond those required for solution of the shortest-path problem.

Suppose second-shortest paths from all nodes to the specified terminal node N are sought. Then we propose the following modification of the Hoffman-Pavley method [23]. After solving the shortest-path problem, determine

v_N, the length of the second-shortest path from N to N (it may be infinity), by considering all deviations at N. Then, for each node k whose shortest path to N contains only one arc, compare (a) the length of the shortest path deviating at k and (b) $d_{kN} + v_N$. The minimum of these two quantities is v_k, the length of the second-shortest path from that node. Then, consider all nodes j that are two arcs from N via the shortest path. For each, compare (a) the length of the shortest path deviating at j and (b) the length of the arc d_{ji} that is the first arc of the shortest path from j to N plus v_i, the previously-determined length of the second-shortest path from i to N. The minimum value is v_j. Repeat this iterative process until all nodes are labelled. Note that the iteration is performed on an index representing the number of arcs in the shortest path from each node. This procedure requires about MN additions and comparisons.

Reference 24, published subsequently to the above method, gives a seemingly different procedure. Define u_i as the length of the shortest path from node i to a specified terminal node N, and v_i as the length of the second-shortest path. Define $\min_k (x_1,\ldots,x_n)$ as the k^{th}-smallest value of the quantities x_i. Then, according to Ref. 24, v_j is characterized by the equation

$$v_i = \min \begin{bmatrix} \min_2 \; (d_{ij} + u_j) \\ j \neq i \\ \\ \min_1 \; (d_{ij} + v_j) \\ j \neq i \end{bmatrix} , \qquad i=1,\ldots,N-1$$

$$v_N = \min_i{}_1 \left[d_{Ni} + u_i \right] . \tag{5}$$

The term $\min_2(d_{ij} + u_j)$ determines the value of the best path originating at node i and deviating from the shortest path at that node i. The term $\min_1(d_{ij} + v_j)$ evaluates the best path consisting of any first arc, plus the second-best continuation. The originators of the method apparently did not notice that if the minimizing node in the $\min_1$ operation is not k (the next node of the already-known shortest path from i) but some other node p, then $d_{ip} + u_p$ (an admissible solution to the $\min_2$ expression) is less than $d_{ip} + v_p$ (since $u_p < v_p$). Since the term $\min_1(d_{ij} + v_j)$ can yield the overall minimum in Eq. (5) only if $j = k$, where k is the node after i on the shortest path from i, the $\min_1$ term in Eq. (5) can be replaced by merely $d_{ik} + v_k$.

After this reduction by a factor of two in required computation, we can calculate the method's approximate

computational requirements. Bellman and Kalaba [24] recommend solution of Eq. (5) by an iterative procedure, where v_i is superscripted on the left by (k + 1) and v_j on the right by (k). Defining M and N as above and L as the average number of iterations until convergence of the iterative solution of Eq. (5), the method requires an average of NML additions and comparisons. L is less than N-1 and may be as large as the number of arcs in the shortest path containing the most arcs. However, after replacing the $\min_l$ term in Eq. (5) by $d_{ik} + v_k$ as discussed above, solution can be greatly accelerated by using a one-pass scheme first labelling nodes one-arc-by-shortest-path from N, then two-arcs-by-shortest-path, etc. This reduces the Bellman-Kalaba [24] procedure to precisely the modified Hoffman-Pavley [23] algorithm recommended above.

In summary, if only the second-shortest path connecting a <u>particular</u> pair of nodes is desired, the method of Ref. 23 is clearly best since it requires MK calculations compared to MN for our improved version of the method of Ref. 24, which must solve the all-initial-nodes problem in order to resolve the particular-initial-node case. If a problem involving a fixed terminal node and all possible initial nodes is posed, the methods as modified in this

Memorandum are equivalent. These conclusions contradict those of Pollack's survey paper [25].

For determination of the third-shortest paths from all initial nodes to N, we recommend the following generalization of the above procedure. If w_i represents the length of the third-shortest path from i, then

$$w_i = \min \begin{bmatrix} d_{ik} + w_k \\ \min_{\substack{2 \\ j \neq i}} [d_{ij} + u_j] \end{bmatrix}, \tag{6}$$

if a single node k follows i along both the first- and second-shortest paths. If k is the node following i on the shortest path, and m is the node following i on the second-shortest, then

$$w_i = \min \begin{bmatrix} d_{ik} + v_k \\ d_{im} + v_m \\ \min_{\substack{3 \\ j \neq i}} [d_{ij} + u_j] \end{bmatrix}. \tag{7}$$

Once the functions u_i and v_i have been determined, the function w_i can be computed node-by-node by first computing

w at nodes that are one arc distant from N via the shortest path, then two arcs, etc. This one-pass procedure generalizes to the n^{th} best-path problem, because the same function appears on both the left and right in the appropriate equation of the type of Eq. (6) above only if the p^{th}-best paths from i for p=1,...,n-1 all go to the same second node.

While we assumed above, in order to avoid complicating the explanations, that no two paths have the same length, all of these methods can be generalized--at the expense of a little additional bookkeeping--to include the treatment of possible ties. The problem of ties can sometimes be avoided by slightly perturbing the data.

Reference 26 examines an entirely different procedure, the efficiency of which is difficult to determine.

Clarke, Krikorian, and Rausen [27] treat this problem under the additional restriction that only loopless paths are admissible. Their branch-and-bound algorithm involves generating, listing, and processing all paths with certain properties. The number of such paths is not easy to bound. Certainly much of the elegance of the above algorithms is lost. It is unclear how the procedure of Ref. 27 generally compares with that of merely producing next-best paths (possibly containing loops) by the above algorithms until obtaining the desired number of loopless ones.

It is not simple to modify Eq. (5) correctly so as to exclude paths with loops.* While Pollack [25] clearly recognizes the problem, his subsequent scheme encounters circularity, since determination of the k^{th}-best loopless path may depend on the length of certain other $k+p$, $p \geq 1$, best paths, and conversely. Another scheme of Pollack [29] is clearly correct, but computation increases rapidly with k and the method can be recommended for only very small k.

---

*Reference 25, p. 555, explains why. The considerations it raises render incorrect the algorithm of Elmaghraby, Ref. 28, Sec. 6.3.

V. TIME-DEPENDENT LENGTHS OF ARCS

At least one paper [30] has studied the problem of finding the fastest path between cities where the time of travel between city i and city j depends on the time of departure from city i. When t is the time of departure from city i for city j, let $d_{ij}(t)$ denote the travel time. (If travel schedules are such that a delay before departure decreases the time of arrival, $d_{ij}(t)$ represents the elapsed time between time t and the earliest possible time of arrival.) This model has applications in the areas of transportation planning and communication routing.

Cooke and Halsey [30] define $f_i(t)$ as the minimum time of travel to N, starting at city i at time t, and establish the formula

$$f_i(t) = \min_{j \neq i} \left[d_{ij}(t) + f_j(t + d_{ij}(t)) \right] \qquad (8)$$

$$f_N(t) = 0 \ .$$

Assuming all the $d_{ij}(t)$ are defined at, and take on, positive integer values, an iterative procedure is given for finding the quickest paths from all cities to city N, starting at city i at time 0. Defining T to be the maximum

taken over all i of $d_{iN}(0)$ (a smaller T can be determined, at some inconvenience, if this number is infinite), and assuming all cities are connected at all times (perhaps by links taking infinite time), the procedure requires at most $N^2 T^2$ additions and comparisons.

This problem can be solved by the method of Dijkstra [1] discussed above (pp. 3-6) just as efficiently as can the problem where the times (or distances) are not time-dependent. Also, the restriction to integer-valued times can be dropped and any real-valued times can be treated. Define the tentative node (city) label f_i to be an upper bound on the earliest time of arrival at node i, and permanent labels to be earliest possible (optimal) times-of-arrival. First, permanently label node i_o (the initial node) zero and all other nodes infinity. Next, tentatively label all nodes j with the minimum of the current node label f_j and the sum of f_{i_o} and $d_{i_o j}(f_{i_o})$. Then, find the minimum, non-permanent node label, say f_k, and declare it permanent. (f_k is the earliest possible time of arrival at node k, leaving node i_o at time 0.) Node k is then used to try to reduce the labels at all tentatively labelled cities, by comparing $f_k + d_{kj}(f_k)$ to the current label, and the minimum new temporary label

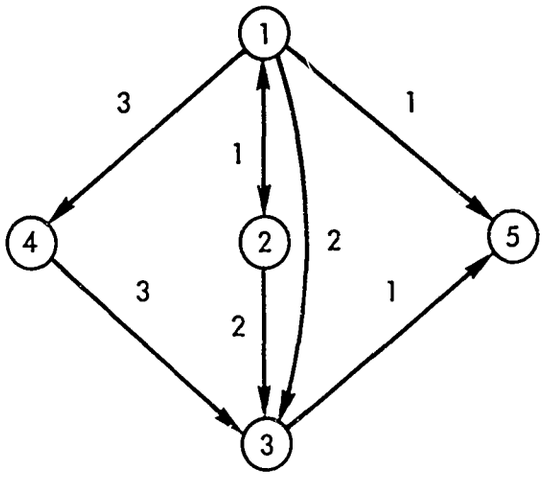
is made permanent, etc. After at most N^2 comparisons and $\frac{N^2}{2}$ additions, city N is labelled and, leaving i_o at time 0, the quickest paths to all nodes, including N, are determined. As is the case for the closely related Dijkstra procedure [1], about $N^2/2$ additions and $2N^2$ comparisons are required when implementing a method of distinguishing temporarily from permanently labelled nodes. If quickest paths from all cities to N are desired, the algorithm must be repeated N-1 times; but, even then, the procedure compares favorably, in both computation and required assumptions, with the Cooke-Halsey algorithm [30].

VI. SHORTEST PATHS VISITING SPECIFIED NODES

Given a set of N nodes and distances $d_{ij} \geq 0$, suppose we desire to find the shortest path between nodes 1 and N that passes through the k-1 nodes 2, 3,...,k ≤ N-1, called "specified nodes." A simple, but completely erroneous, solution of this problem was reported by Saksena and Kumar [31] Noting this, we wish to give a solution method.

The fallacy in Ref. 31 is the assertion (subject to a proviso to follow) that the shortest path from a specified node i to N passing through at least p of the specified nodes enroute is composed of the shortest path from i to some specified node j, followed by the shortest path from j to N passing through at least p-r-1 specified nodes, where r is the number of specified nodes that lie on the shortest unrestricted path from i to j. Saksena and Kumar [31] incorrectly assert that this is true, provided--should specified nodes occurring on the shortest path from i to j also lie on the continuation path from j to N and therefore be counted twice--that at least p distinct specified nodes lie on the path. Should some candidate path corresponding to some j violate the duplication-of-nodes proviso above, that possibility is inadmissible and the possibility

of going initially from i to j by shortest path is dropped from consideration. The procedure fails to note that, in this case, some less short continuation from j passing through at least p-r-1 specified nodes, and avoiding duplication of nodes, may yield a better path than the best remaining path satisfying the conditions described above. For example, in the network shown below, with all nodes considered specified, suppose we seek the best path from 1 to 5 passing through at least two intermediate nodes.



The best path from 1 to 4 has length 3 and happens to pass through no nodes enroute, and the best continuation from 4 to 5 passing through at least one node has length 4; hence, this possibility has length 7. The best path

from 1 to 3 has length 2 and happens to pass through no nodes enroute, and the best continuation from 3 to 5 passing through one node has length infinity (no such continuation from 3 exists). The best path from 1 to 2 has length 1 and happens to pass through no nodes enroute, and the best continuation from 2 to 5 passing through one node enroute has length 2 (it returns to node 1), yielding a sum of 3. However, it is inadmissible as a path through two intermediate specified nodes because node 1 is counted twice. The answer, by the Saksena-Kumar algorithm [31], would then be 7, the best of the other alternatives. Yet, the path 1-2-3-5 has length 4 and is admissible. This is an example of a best first portion and a second-best continuation being optimal. (Or, the same path can alternatively be viewed as the second-best path from 1 to 3 followed by the best continuation.) No simple modification of the referenced method seems to handle this kind of situation.

Assuming paths with loops are admissible, the problem can be correctly solved as follows. First solve the shortest-path problem for the N-node network for all pairs of initial and final nodes. Let d'_{ij} represent the length of the shortest path from node i to j. Then, solve the

(k + 1)-city "traveling-salesman" problem for the shortest path from 1 to N passing through nodes 2, 3,...,k, where the distance from node i to j is d'_{ij}. Reference 32 discusses methods of solution. While no easy solutions exist for the traveling-salesman problem, the specified-city problem can certainly be no easier than the traveling-salesman problem of dimension k + 1, since if k = N-1 it is the traveling-salesman problem.

VII. CONCLUSION

We have referenced, evaluated, and occasionally modified various algorithms for computationally solving certain shortest-path problems. By collecting contributions from several disciplines, we hope to re-orient and revitalize the sometimes rather incestuous research of each. The author would appreciate notification of overlooked or new significant research in this area.

REFERENCES

1. Dijkstra, E. W., "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, Vol. 1, 1959, pp. 269-271.

2. Whiting, P. D., and J. A. Hillier, "A Method for Finding the Shortest Route through a Road Network," Operational Research Quarterly, Vol. 11, Nos. 1/2, March/June 1960, pp. 37-40.

3. Pollack, M., and W. Wiebenson, "Solution of the Shortest-Route Problem--A Review," Op. Res., Vol. 8, No. 2, March-April 1960, pp. 224-230.

4. Ford, L. R., Jr., and D. R. Fulkerson, "Constructing Maximal Dynamic Flows from Static Flows," Op. Res., Vol. 6, No. 3, May-June 1958, pp. 419-433.

5. Dantzig, G. B., "On the Shortest Route through a Network," Management Science, Vol. 6, No. 2, January 1960, pp. 187-190. See also Ref. 7.

6. Berge, C., and A. Ghouila-Houri, Programming, Games and Transportation Networks, trans. by M. Merrington and C. Ramanujacharyulu, Methuen, London, 1965, p. 176.

7. Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963, pp. 363-366.

8. Nicholson, T.A.J., "Finding the Shortest Route Between Two Points in a Network," The Computer Journal, Vol. 9, No. 3, November 1966, pp. 275-280.

9. Murchland, J. D., The "Once-Through" Method of Finding all Shortest Distances in a Graph from a Single Origin, Transport Network Theory Unit, London Graduate School of Business Studies, Report LBS-TNT-56, August 1967.

10. Ford, L. R., Jr., Network Flow Theory, The RAND Corporation, P-923, August 1956.

11. Moore, E. F., "The Shortest Path through a Maze," Proc. Int. Symp. on the Theory of Switching, Part II, April 2-5, 1957, The Annals of the Computation Laboratory of Harvard University 30, Harvard University Press, 1959, pp. 285-292.

12. Bellman, R. E., "On a Routing Problem," Quart. Appl. Math., Vol. XVI, No. 1, April 1958, pp. 87-90.

13. Yen, J. Y., Matrix Algorithm for Solving All Shortest Routes from a Fixed Origin in the General Networks, Graduate School of Business Administration, University of California, Berkeley (unpublished manuscript).

14. Dantzig, G. B., W. O. Blattner, and M. R. Rao, All Shortest Routes from a Fixed Origin in a Graph, Operations Research House, Stanford University, Technical Report 66-2, November 1966.

15. Hitchner, L. E., A Comparative Investigation of the Computational Efficiency of Shortest Path Algorithms, Operations Research Center, University of California, Berkeley, Report ORC 68-17, July 1968.

16. Floyd, R. W., "Algorithm 97, Shortest Path," Comm. ACM, Vol. 5, No. 6, June 1962, p. 345.

17. Warshall, S., "A Theorem on Boolean Matrices," J. ACM, Vol. 9, 1962, pp. 11-12.

18. Murchland, J. D., A New Method for Finding all Elementary Paths in a Complete Directed Graph, Transport Network Theory Unit, London School of Economics, Report LSE-TNT-22, October 1965.

19. Dantzig, G. B., All Shortest Routes in a Graph, Operations Research House, Stanford University, Technical Report 66-3, November 1966.

20. Shimbal, A., "Structure in Communication Nets," Proc. of the Symposium on Information Networks, Polytechnic Institute of Brooklyn, April 12-14, 1954.

21. "Investigation of Model Techniques," Second Annual Report, July 1957-1958, Case Institute of Technology, Cleveland, Ohio, ASTIA Report AD211968.

22. Hu, T. C., "Revised Matrix Algorithms for Shortest Paths," SIAM J. on Appl. Math., Vol. 15, No. 1, January 1967, pp. 207-218.

23. Hoffman, W., and R. Pavley, "A Method for the Solution of the Nth Best Path Problem," J. ACM, Vol. 6, No. 4, October 1959, pp. 506-514.

24. Bellman, R., and R. Kalaba, "On kth Best Policies," J. SIAM, Vol. 8, No. 4, December 1960, pp. 582-588.

25. Pollack, M., "Solutions of the kth Best Route through a Network--A Review," J. Math. Anal. and Appl., Vol. 3, August-December 1961, pp. 547-559.

26. Sakarovitch, M., The k Shortest Routes and the k Shortest Chains in a Graph, Operations Research Center, University of California, Berkeley, Report ORC 66-32, October 1966.

27. Clarke, S., A. Krikorian, and J. Rausen, "Computing the N Best Loopless Paths in a Network," J. SIAM, Vol. 11, No. 4, December 1963, pp. 1096-1102.

28. Elmaghraby, S. E., The Shortest Route Problem: Survey and Extensions, Yale University, September 1966 (unpublished manuscript).

29. Pollack, M., "The kth Best Route through a Network," Op. Res., Vol. 9, No. 4, July-August 1961, pp. 578-580.

30. Cooke, K. L., and E. Halsey, "The Shortest Route through a Network with Time-Dependent Internodal Transit Times," J. Math. Anal. and Appl., Vol. 14, No. 3, June 1966, pp. 493-498.

31. Saksena, J. P., and S. Kumar, "The Routing Problem with 'K' Specified Nodes," Op. Res., Vol. 14, No. 5, September-October 1966, pp. 909-913.

32. Bellmore, M., and G. L. Nemhauser, "The Traveling Salesman Problem: A Survey," Op. Res., Vol. 16, No. 3, May-June 1968, pp. 538-558.