JOURNAL OF RESEARCH of the National Bureau of Standards—B. Mathematical Sciences
Vol. 78B, No. 3, July–September 1974

Computational Experience With an Algorithm for Finding the k Shortest Paths in a Network*

Douglas R. Shier

Institute for Basic Standards, National Bureau of Standards, Washington, D.C. 20234

(July 12, 1974)

A particular computer implementation of the Double-Sweep method for calculating the k shortest paths in a network is described. Results are presented for a series of computational experiments performed on rectangular grid networks.

Key words: Double-Sweep method; graph; k shortest paths; network; network algorithms; shortest path.

1. Introduction

A common task which arises in analyzing a system of interconnected elements or *network* is that of calculating *shortest paths*—i.e., routes through the system whose total length or cost is as small as possible. Such calculations occur quite naturally in the context of transportation and communication networks. In applications such as these, it is sometimes desirable to have knowledge of the k *shortest paths* in contrast to simply a shortest path. For example, the knowledge of good alternative routes (as opposed to just the shortest one) can be used by transportation planners to model more realistically the flow of vehicular traffic on a road network. Or, as a second example, the routing of messages through a communications network when some routes are temporarily obstructed can be based on the best alternative routes which are available.

Several algorithms have been traditionally employed in order to determine the k shortest paths between specified nodes of a network (such paths may in fact contain repeated nodes). An excellent survey of these algorithms is provided by the review article of Dreyfus [1][1]. More recently, several new methods for performing such calculations have been proposed [2, 4]. These methods are based on a fairly strong analogy which exists between the solution of network path problems and traditional techniques for solving ordinary linear equations. On the basis of preliminary theoretical and computational evidence, one of these (the Double-Sweep method) emerged [4] as a reasonably effective procedure for calculating k shortest paths between a given node and all other nodes in a network.[2] The purpose of this report is to describe a particular implementation of the Double-Sweep method in FORTRAN V[3] and to present a body of computational results for a practically important class of networks (namely, those with a rectangular grid topology).

2. The Double-Sweep Method

Suppose that $G = (\mathcal{N}, \mathcal{A})$ is a finite directed *network* in which the real number l_{ij} denotes the length of *arc* $(i, j) \in \mathcal{A}$ joining *nodes* $i, j \in \mathcal{N}$. Node i of the arc (i, j) is said to be *incident to*

AMS Subject Classification: 05035.

*This work was done while the author was a National Academy of Sciences—National Research Council Postdoctoral Research Associate at the National Bureau of Standards, Washington, D.C. 20234.

[1] Figures in brackets indicate the literature references at the end of this paper.

[2] The algorithms presented in [2] are appropriate for calculating k shortest paths between all pairs of nodes; because of storage limitations, the application of these algorithms is limited to networks having at most a few hundred nodes.

[3] FORTRAN V is UNIVAC's augmented version of standard FORTRAN IV.

node j, while node j is said to be *incident from* node i. A *path* from node i to node j is an ordered sequence of arcs $[(i, i_1), (i_1, i_2), \ldots, (i_{m-1}, j)]$ in the network; a path is termed *elementary* if all nodes appearing along the path are distinct. A *circuit* is a path whose *starting node* i and *ending node* j coincide. The *length* of a path is defined to be the arithmetic sum of the arc lengths l_{ij} along the path.

The problem under investigation here is that of determining, among *all* paths extending between two specified nodes, those paths having the smallest, the second smallest, . . . and the kth smallest length. It is emphasized that these k *shortest paths* (that is, paths whose lengths are shortest, second shortest, . . . or kth shortest) are not required to be elementary. The method to be discussed here will allow the calculation of the k shortest paths from a given source node to all other nodes in the network. It is assumed that

(1) All circuits in the network have positive length, and
(2) The network contains no self-loops: that is, circuits of the form $[(i, i)]$.

The *Double-Sweep* method calculates the k shortest path lengths from a particular source node to all n nodes of the given network by means of alternating forward and backward passes. A precise statement of the method, together with a proof of its validity, can be found in [4]. Basically, the method begins with an initial guess as to the k shortest path lengths and successively improves the current guess to obtain an even better guess. During the forward pass, the current path lengths to each node $j = 1, \ldots, n$ are modified by using those nodes $i < j$ which are incident to node j. If the sum of a current path length to some node i and the arc length l_{ij} provides a shorter path length to node j than any which is currently known, then the corresponding path lengths to node j are updated to give an improved estimate of the k shortest path lengths. A similar procedure is followed during the backward pass, except that now only nodes $i > j$ are considered with $j = n, \ldots, 1$.

The alternating forward and backward passes are continued until no improvement in the path lengths to any node can be made. Convergence in this sense will always be achieved in a finite number of steps—in fact, in at most $nk+1$ forward and $nk+1$ backward passes. When convergence does obtain, the k shortest path lengths to each node j in the network will have been found. From such path length information, the actual paths corresponding to any of the k shortest path lengths can be determined by a backward path tracing procedure.

In essence, this path tracing procedure is based on the following fact. Namely, if a tth shortest path π of length l from node i to node j passes through node r, then the subpath of π extending from node i to node r is a qth shortest path for some q, $1 \leq q \leq t$. This fact can be used to determine the penultimate node r on a tth shortest path of known length l from node i to node j. Indeed, any such node r can be found by forming the quantity $l - l_{rj}$ for all nodes r incident to node j and determining if this quantity appears as a qth shortest path length ($q \leq t$) for node r. If so, then there is a tth shortest path of length l whose final arc is (r, j); otherwise, no such path exists. This idea is repeatedly used, in the manner of a backtrack program, to produce all paths from i to j with the length l, and ultimately all the k shortest paths from node i to node j.

It should be pointed out that, while the Double-Sweep method will work perfectly well if circuits of zero length are present, the preceding path tracing procedure may encounter some difficulties if such zero length circuits exist. The essential difficulty is that there can then be an infinite number of paths having the same length, so the generation of *all* of these paths is clearly impossible. Accordingly, the simple backtracking system described above could possibly cycle indefinitely unless certain precautions are taken. For the sake of retaining simplicity, then, the (quite reasonable) assumption has been made that all circuits have strictly positive length.

3. Program Implementation

The calculation of k shortest paths from a particular source node can be accomplished through the use of the four subprograms listed in section 5. The subroutine INPUT allows the description

of the given network to be entered, the subroutines DSWP and XMULT are used to calculate the k shortest path lengths, and the subroutine TRACE enables the actual tracing out of the k shortest paths. Certain details of the specific implementation used will be discussed in this section.

The first issue concerns the choice of the starting guess with which to initiate the Double-Sweep method, as several choices are possible. It has proved convenient to assign, as the initial approximation, k "infinite" path lengths to each node (the value used for infinity was $INF = 99999999$), save for the source node NS which receives the k-vector of path lengths $(0, INF, \ldots, INF)$. At any step of the process, the k-vector associated with each node will contain the k shortest path lengths found so far from node NS to that node. Moreover, these k path lengths are always distinct (apart from infinite values) and are always arranged in strictly increasing order. Such an ordering allows the following two computationally important observations to be made.

(1) If the value INF is encountered in some component of a k-vector, then all subsequent components of the k-vector also contain INF values. Therefore, when updating the k-vector for node j during a forward or backward pass, the k-vector for a node i incident to j need only be scanned as far as the first occurrence of an INF value since an infinite value cannot possibly yield an improved path length for node j.

(2) If IXV, the sum of some current path length in the k-vector for node i and the arc length l_{ij}, is greater than or equal to the maximum element of the k-vector for node j, then no improvement in the latter k-vector by use of the former can possibly be made. Therefore, it is appropriate to keep track of the current maximum element MAX of the k-vector for node j. If IXV is less than MAX then it is possible for an improvement to be made, as long as the value IXV does not already occur in the k-vector for node j (only *distinct* path lengths are retained).

The use of these two observations allows for a substantial reduction in the amount of computational effort required to update the current path lengths, as compared to the use of some general sorting routine to find the k smallest elements in a list. Since such updating comprises the major computational requirement of the Double-Sweep method, it has proved advantageous to keep the components of each k-vector in strictly increasing order. These updating steps, corresponding to appropriate "matrix multiplications" as defined in [4], are performed by the subroutine XMULT, called as required by the subroutine DSWP.

Together, the subprograms DSWP and XMULT enable the calculation of the K shortest path lengths from any given source node NS. In some applications, these path lengths may be all the information that is required by the user. In others, the actual paths joining various pairs of nodes are also needed. To accomplish this latter task, the subroutine TRACE is used to produce all paths from node NS to node NF having any of the K shortest path lengths from NS to NF. As presently implemented, paths containing up to 5000 arcs will be generated; an error message will indicate when this condition is not fulfilled.

The subroutine INPUT allows a description of the given network to be read in from external unit number 5 together with values for relevant parameters. The network description is achieved by specifying for each arc of the network a record containing its starting node, its ending node and its length. The records are assumed to be sorted in increasing order by arc ending node; moreover, the nodes are assumed to be numbered consecutively from 1 to N. The totality of such network description records is followed by a blank record and then a sequence of parameter specification records. Each block of parameter specification records consists of a path calculation record followed by any number, possibly zero, of path tracing records. The path calculation record gives values for K, NS and $IMAX$. The parameter $IMAX$ is the maximum number of Double-Sweep iterations [4] allowed before a mandatory termination of DSWP is imposed; for most cases, a value of $IMAX = 100$ should suffice. Each path tracing record gives values for NF and $PMAX$, which allow the user to trace out actual paths from node NS to node NF if required; at most $PMAX$ such paths will be determined. Several (but at least one) parameter specification blocks can be accommodated so that various values for K, NS and NF can be explored in the given network. The final record of the input stream is required to be blank.

[4] Each forward or backward pass constitutes an iteration.

The form of input acceptable to the program can be concisely specified by the following *BNF* formula system [3].

⟨input⟩ ::= ⟨network part⟩ ⟨blank record⟩ ⟨parameter part⟩ ⟨blank record⟩
⟨network part⟩ ::= ⟨arc record⟩|⟨network part⟩ ⟨arc record⟩
⟨parameter part⟩ ::= ⟨parameter block⟩|⟨parameter part⟩ ⟨parameter block⟩
⟨parameter block⟩ ::= ⟨path calcn record⟩|⟨parameter block⟩ ⟨path trace record⟩

Definitions for the various records mentioned above are provided below; the columns of a record which are not explicitly mentioned are assumed to be blank.

⟨blank record⟩:
⟨arc record⟩: Cols. 1–10 = Arc Starting Node
Cols. 11–20 = Arc Ending Node
Cols. 21–30 = Arc Length
⟨path calcn record⟩: Cols. 1–5 = K
Cols. 6–10 = NS
Cols. 11–15 = $IMAX$
⟨path trace record⟩: Cols. 1–5 = NF
Cols. 6–10 = $PMAX$

The values given to K, NS, $IMAX$, NF, and $PMAX$ are all specified by the user. In addition, the following ranges are assumed for the indicated variables.

N = number of nodes in the network:	$2 \leq N \leq 1000$.
MU = number of arcs (I, J) with $I < J$:	$0 \leq MU \leq 5000$.
ML = number of arcs (I, J) with $I > J$:	$0 \leq ML \leq 5000$.
K = number of distinct path lengths required:[5]	$2 \leq K \leq 20$.

Thus, at most $K = 20$ shortest path lengths from a given node in a network with up to 1000 nodes and 10,000 arcs can be handled by the currently programmed version of the algorithm. The approximate storage requirements for the four subroutines comprising the package are given in table 1; the storage requirements appropriate for arbitrary values of N, K, and $M = MU + ML$ are given in table 2. If there is insufficient storage available on a particular computer system to retain all the subroutines in core simultaneously, it is possible to break up the execution sequence by performing first the path length calculation (DSWP/XMULT) and next the path tracing calculation (TRACE). This is a feasible strategy because essentially two different network representations are used in these two distinct calculation phases.

TABLE 1. *Approximate number of storage locations required by various subprograms*

Subprogram	Storage for common blocks used				Program instruction storage	Total required storage[a]
	BLK1	BLK2	BLK3	BLK4		
INPUT	22003	2		21001	268	43274
DSWP	22003	2	20000		267	42272
XMULT		2	20000		214	20216
TRACE		2	20000	21001	15310	56313
ALL	22003	2	20000	21001	16059	79065

[a] Exclusive of system routines (requiring approximately 5700 locations for the UNIVAC 1108, EXEC 8 operating system used).

[5] Only minor program changes are necessary to allow the case $K = 1$.

TABLE 2. *Approximate number of storage locations required by common blocks and subprograms in terms of* N, K *and* M = MU + ML

Storage for common blocks			Total storage for subprograms[a]
BLK1	$2N+2M+3$	INPUT	$3N+4M+274$
BLK2	2	DSWP	$NK+2N+2M+272$
BLK3	NK	XMULT	$NK+216$
BLK4	$N+2M+1$	TRACE	$NK+N+2M+15313$
		ALL	$NK+3N+4M+16065$

[a] Exclusive of system routines (requiring approximately 5700 locations for the UNIVAC 1108, EXEC 8 operating system used).

TABLE 3. *Listing of sample card input*

2		1	65
5		1	38
1		2	78
3		2	82
6		2	78
2		3	79
4		3	96
7		3	55
3		4	87
8		4	12
1		5	24
6		5	9
9		5	18
2		6	45
5		6	48
7		6	22
10		6	23
3		7	8
6		7	41
8		7	44
11		7	58
4		8	29
7		8	47
12		8	51
5		9	74
10		9	88
6		10	69
9		10	40
11		10	48
7		11	32
10		11	35
12		11	93
8		12	14
11		12	30
(blank)			
5	12	30	
1	20		
(blank)			

TABLE 4. *Listing of sample printed output*

$K=5$ shortest path lengths from node 12

To node					
1	164	205	211	220	221
2	195	232	236	241	242
3	150	159	191	200	206
4	63	104	128	145	154
5	126	167	173	182	183
6	117	158	164	173	174
7	95	136	151	158	160
8	51	92	116	133	142
9	200	229	241	247	256
10	141	175	186	206	216
11	93	127	158	168	176
12	0	65	106	123	130

Number of iterations required for convergence = 6

The K shortest paths from node 12 to node 1

Path	Length	Node sequence							
1	164	12	8	7	6	5	1		
2	205	12	8	4	8	7	6	5	1
3	211	12	11	10	6	5	1		
4	220	12	11	7	6	5	1		
5	221	12	8	7	6	5	6	5	1

The output produced from execution of the subroutine DSWP takes the form of a tabular listing of the K shortest path lengths from node NS to all nodes in the network. In addition, the number of Double-Sweep iterations required for convergence of the algorithm is printed. Each time the subroutine TRACE is called, the required K shortest paths (together with their respective lengths) are printed out as the appropriate sequences of nodes leading from NS to NF. All printing is assumed to be done through external unit number 6.

Documented listings of the four subprograms INPUT, DSWP, XMULT and TRACE are given in section 5. Further details about this implementation of the Double-Sweep method, with a path tracing facility, can be obtained by referring to these listings.

Finally, we describe some sample input for the k shortest path package as well as the resulting output. The particular network chosen for illustrative purposes is shown in figure 1. For this net-

work of 12 nodes and 34 arcs, it is required to find the five shortest path lengths from node 12 to all nodes and the actual paths between node 12 and node 1. Accordingly, $K = 5$, $NS = 12$, and $NF = 1$; the values chosen for the other two input parameters are $IMAX = 30$, $PMAX = 20$. Table 3 displays the form of card input required for this particular problem. Note that the network description cards are listed in order of increasing arc ending node number. Table 4 displays the resulting output from the computer program.

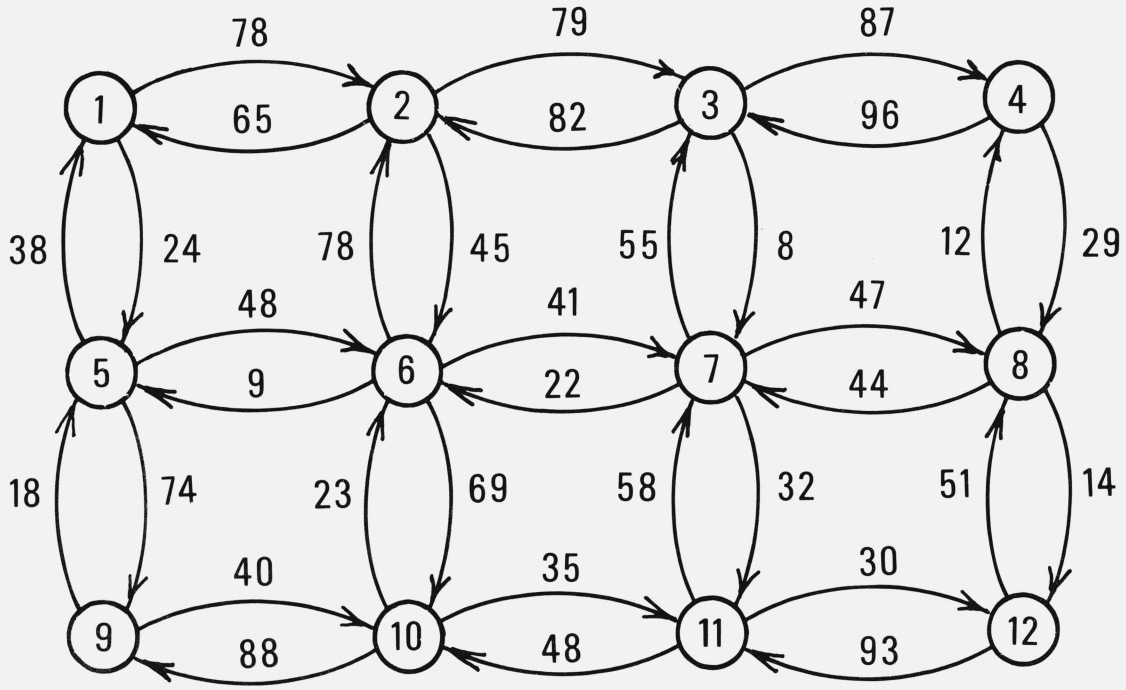

FIGURE 1. *An illustrative network.*

4. Computational Results

The Double-Sweep method has been previously shown [4] to possess certain theoretical and computational advantages over other algorithms for finding the k shortest paths from a given node. The current implementation of this method, embodied in the four subroutines discussed earlier, has undergone testing for a variety of different sample networks. This section will present in particular the results of a number of computational experiments performed on rectangular grid networks. Such networks were chosen for detailed investigation since grid-like networks arise quite frequently in representing, for example, a city street system.

A general $P \times Q$ grid network consists of $N = PQ$ nodes (which are assumed to be numbered consecutively from left to right and top to bottom) and $4PQ - 2(P + Q)$ arcs. The integral lengths of each arc are generated from a uniform distribution over the interval 1 to *RANGE*. (See fig. 1 for an illustrative 3×4 grid network.) The currently implemented version of the Double-Sweep method was used to obtain the K shortest path lengths from node NS to all nodes of various grid networks; no tracing of paths was considered in these computational experiments. The response characteristics of elapsed computation time on a UNIVAC 1108 were studied as different parameters of the problem were varied.

First, a series of four 24×24 grid networks $G_1 - G_4$ were created by drawing four different random samples of arc lengths from the uniform distribution on 1 to 100. Figures 2–4 show how for three distinct source nodes the variation of computation time with K is affected by the particular arc length sample G_i chosen. Since the arc length sample chosen does have a noticeable effect on computation time, it was decided that subsequently all results would be averaged over four arc length samples. Figure 5 shows how the *average* computation time varies, quite nearly quadratically, with K for three distinct source nodes. In fact, table 5 displays the second-degree polynomials which give the best (least squares) fit to the curves of figure 5 over the range $2 \leq K \leq 20$. From the values obtained for the residual standard deviations, these second-degree polynomials produce an excellent fit to the observations; essentially, then, the average computation time varies quadratically with K. Such behavior is not unexpected since the doubling of K, for instance, not only increases the computation per iteration (potentially twice as many sums IXV must now be formed for each node) but also tends to increase the number of iterations required for convergence (the equality of $2k$-vectors automatically ensures the equality of their first k components).

TABLE 5. *Best fit second-degree polynomials to figure 5 curves*

NS	Second-degree fitted polynomial	Residual standard deviation[a](s)
1	$0.8457 + 0.1616\,K + 0.0260\,K^2$	0.1369
300	$1.6111 + 0.0761\,K + 0.0425\,K^2$	0.2151
576	$0.7747 + 0.1603\,K + 0.0249\,K^2$	0.1121

[a] Residual standard deviation $= \sqrt{\frac{\Sigma\,(y_i - \hat{y}_i)^2}{n-3}}$.

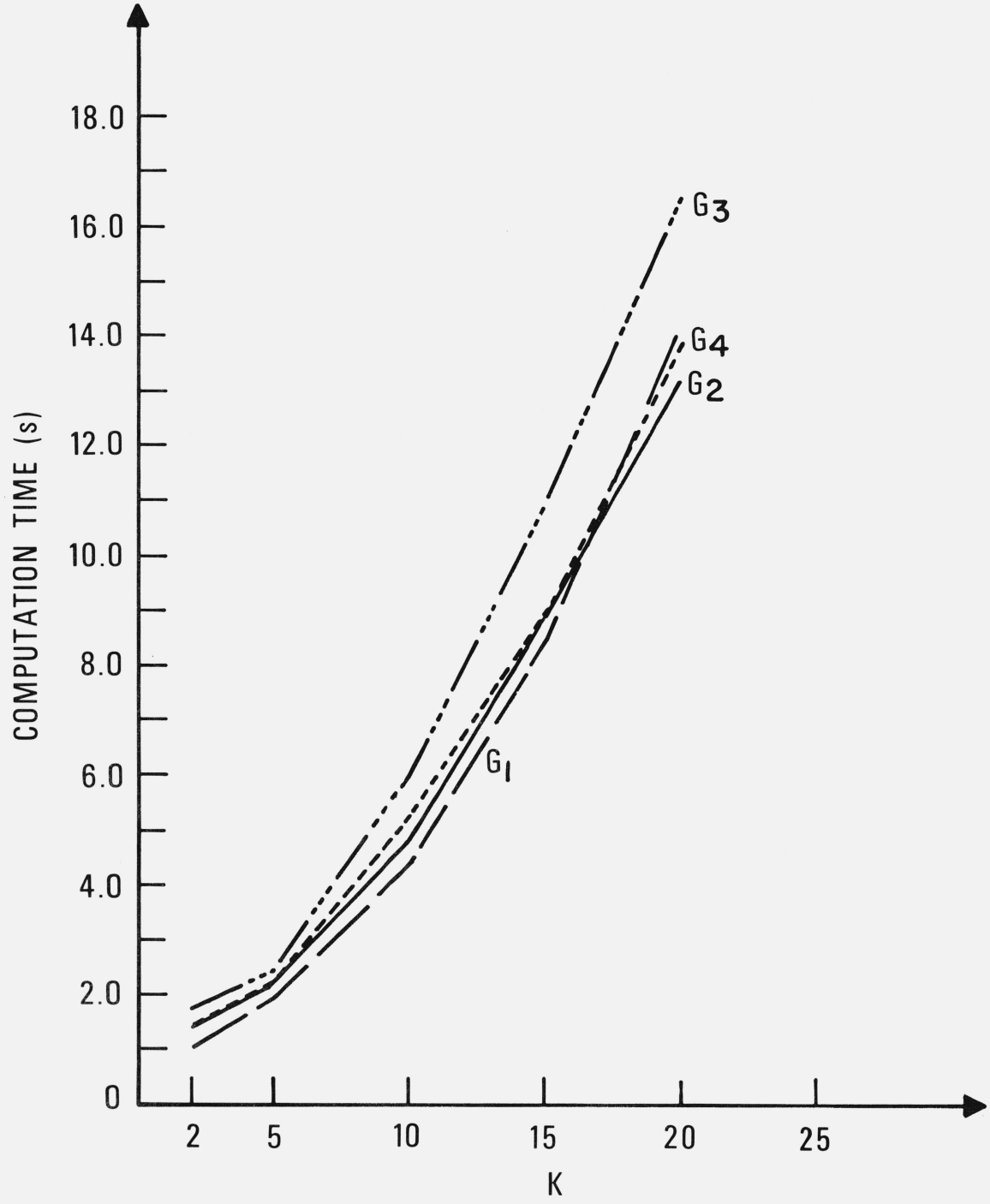

FIGURE 2. *Computation times for four 24 × 24 grids ("corner" source node 1,* RANGE = *100).*

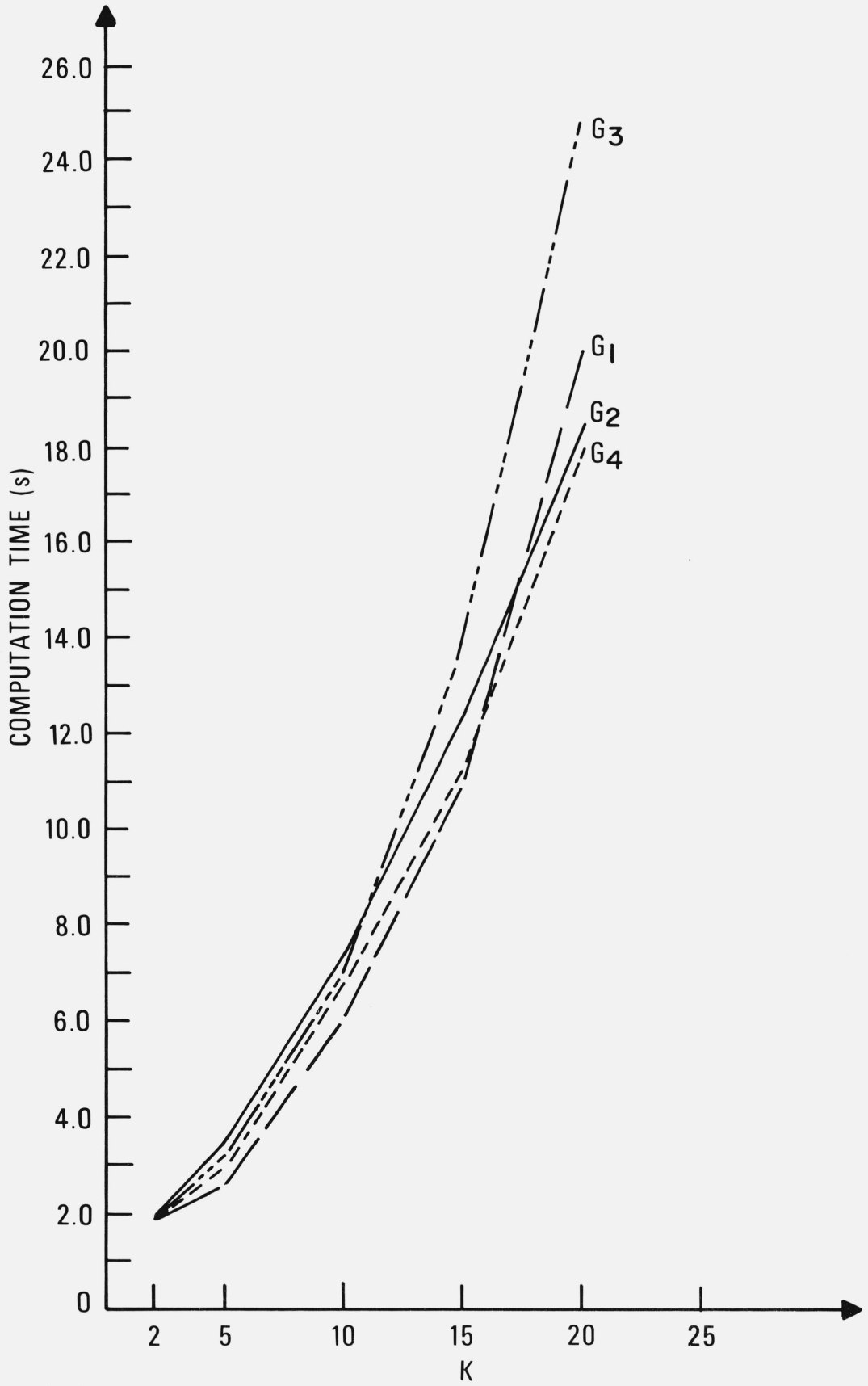

FIGURE 3. *Computation times for four 24 × 24 grids ("central" source node 300,* RANGE = *100).*

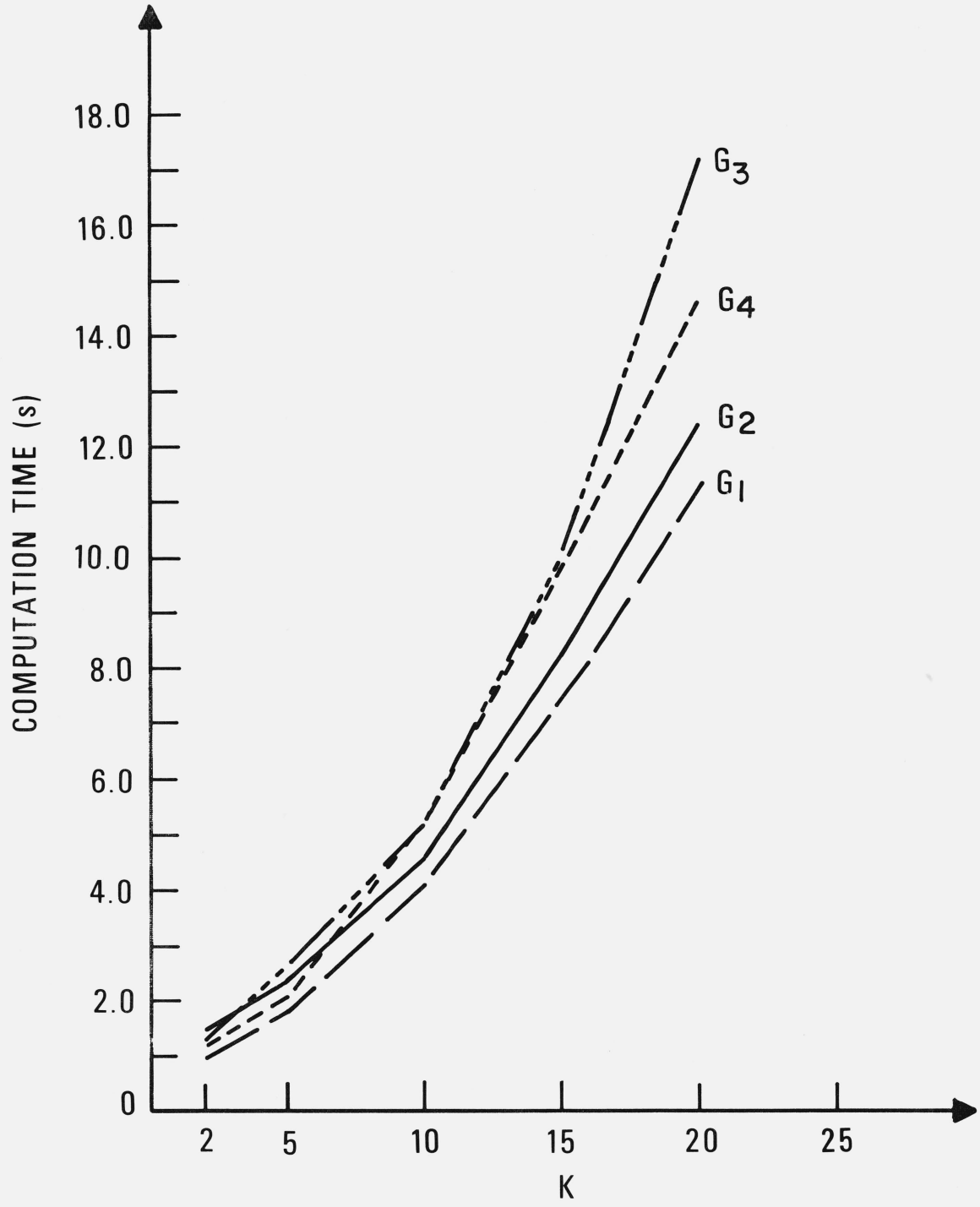

FIGURE 4. *Computation times for four 24×24 grids ("corner" source node 576,* RANGE = *100).*

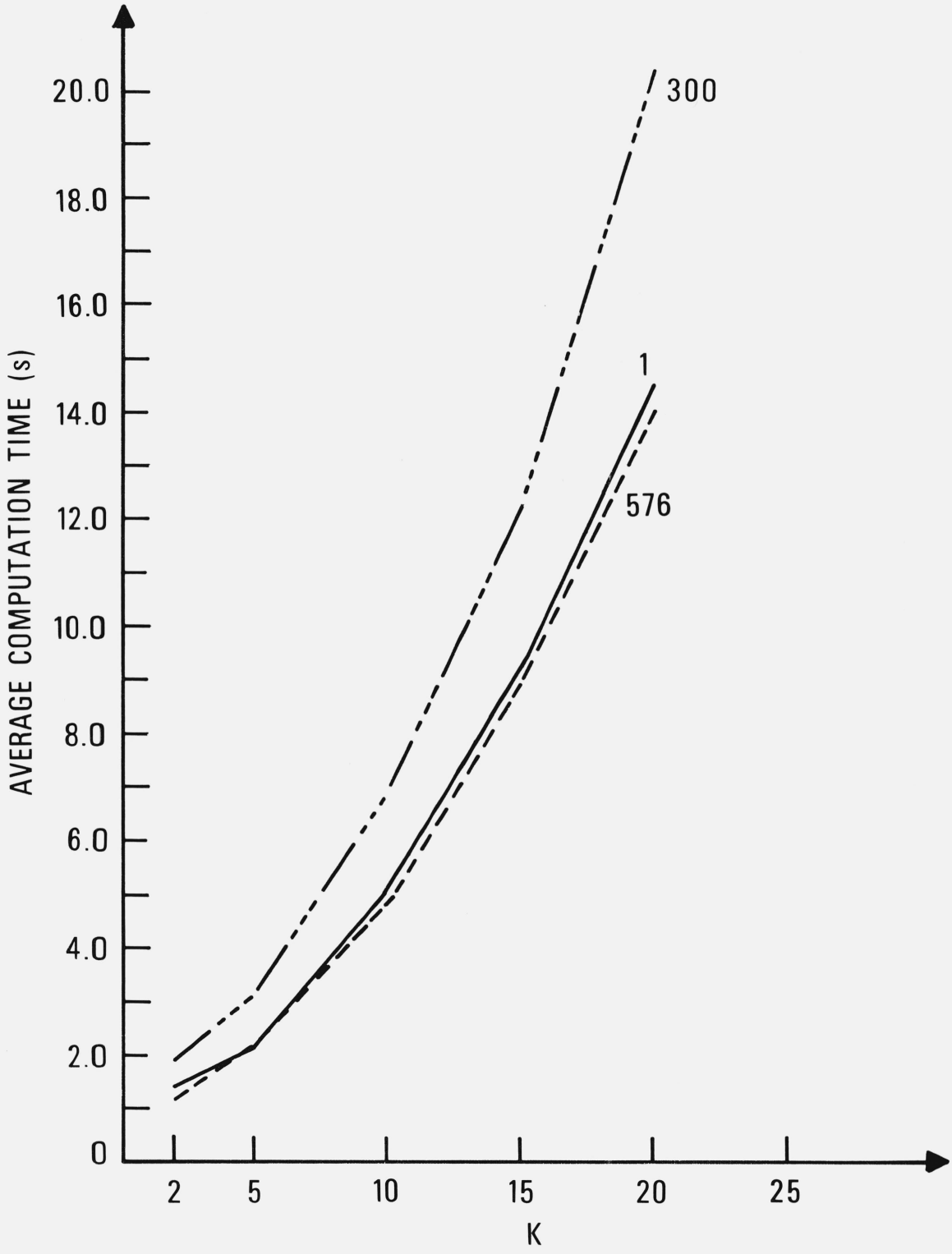

FIGURE 5. *Average computation time for source nodes 1, 300, 576* (RANGE = *100*).

In addition, figure 5 shows that the average computation time for a centrally located source node ($NS = 300$) is consistently higher than for the boundary nodes $NS = 1$ and $NS = 576$ (located at the upper left and lower right corners of the grid network). One possible explanation for this observed difference between central and corner nodes is based on the concept of a *nonretrograde* path in a grid network: that is, a path whose tracing only involves two orthogonal directions (e.g., north and east). A nonretrograde path from node 1 only involves the south and east directions, while a nonretrograde path from node N only involves the north and west directions. On the other hand, a nonretrograde path from a centrally located node can use any pair of orthogonal directions. It has been observed in practice that for grid networks with randomly generated arc lengths a large percentage of the calculated k shortest paths are nonretrograde paths, while the remaining such paths tend to be quite nearly nonretrograde paths. Because the nodes are numbered from left to right and top to bottom, a nonretrograde path from node 1 will have its length correctly assigned in a single forward pass of the Double-Sweep method. Similarly, a nonretrograde path from node N will have its length correctly assigned in a single backward pass. However, a nonretrograde north/east or south/west path from a centrally located node will generally require several forward and backward passes in order for its length to be correctly determined. For example, in a north/east path each horizontal segment will require a separate forward pass and each vertical segment will require a separate backward pass. Since the processing of nonretrograde (as well as nearly nonretrograde) paths from a centrally located node requires more effort than that for a corner node, it seems reasonable that the calculation of k shortest path lengths for the former should involve more computational effort as compared to the latter type of node.

Another series of grid networks all containing 576 nodes but of differing "shapes" (different ratios of P to Q) were created for test purposes. With $NS = 1$ and $RANGE = 100$, the dependence of average computation time was explored as a function of network shape for different values of K. Figure 6 displays the relation between average computation time and $\log(P/Q)$ for 24×24, 32×18, 64×9, and 144×4 grid networks. It is apparent that a "square-like" network requires less computation than an "elongated" network, despite the fact that for a given number of nodes a square-like network contains more arcs than an elongated one. A similar relationship between computation time and shape has been observed in 24×24, 18×32, 9×64, and 4×144 grid networks.

In another sequence of test networks with $N = 576$, $NS = 1$ and $K = 5$, the effect of *RANGE* on average computation time was investigated. Four different values of *RANGE* (1, 10, 100, 1000) were chosen and the average computation time has been plotted as a function of range for 24×24 and 144×4 grid networks (see fig. 7). It is clear that the greater the variability of the arc lengths, the greater the computation time required. As seems reasonable, for larger values of *RANGE*, the effect is not as radical as for smaller values of *RANGE*.

Finally, a sequence of square $P \times P$ grid networks was studied with $NS = 1$, $K = 5$ and $RANGE = 100$ in order to assess the effect of the number of nodes on average computation time. The parameter P was allowed to assume the values $P = 10$, 15, 20, 24 and 30 (producing networks with between 100 and 900 nodes). The influence of this variation in P is seen in figure 8. Quite surprisingly, in the range $P = 15$ to $P = 30$ the increase with P is demonstrably linear, with an R^2 value of 0.988 when a linear regression is performed. Recall that, by contrast, average computation time appears to be quadratically dependent on the parameter K.

In sum, then, several series of computational experiments have been performed in order to evaluate the effect of various parameters on computation time, averaged over four different samples from the distribution of arc lengths. The results obtained by varying these parameters can be helpful guidelines when planning computations to investigate changes to a particular network.

In all of the above experiments, the numbering of nodes was taken to be the usual left-to-right, top-to-bottom numbering described earlier. Since each pass of the Double-Sweep method processes the nodes in the fixed order $1, \ldots, N$ (for a forward pass) or the fixed order $N, \ldots, 1$ (for a backward pass), the actual numbering of nodes used can potentially affect the convergence characteristics of this method. Accordingly, a 20×20 grid network with specific randomly generated arc

lengths in the range 1 to 100 was investigated for various node numbering schemes. To begin, 10 different randomly generated permutations were used to renumber the nodes of this 20×20 grid network. The $K=5$ shortest path lengths were calculated from the source node located at the upper left corner of each of the 10 randomly renumbered networks, yielding a mean computation time of 2.488 s with a standard deviation of 0.220 s. By contrast, when the usual left-to-right, top-to-bottom numbering scheme was used, the same $K=5$ shortest path lengths were produced in only 1.213 s, approximately half the average time for randomly numbered networks.

Furthermore, when the nodes were renumbered consecutively from left to right along odd-numbered rows and right to left along even-numbered rows, the required computing time was 1.683 s. On the other hand, when the nodes were renumbered consecutively from top to bottom along odd-numbered columns and bottom to top along even-numbered columns, the required computing time was 1.504 s. The main conclusion to be drawn from such times is that the ordering of nodes can have a pronounced effect on the efficiency of the Double-Sweep method. In particular, a systematic numbering scheme is to be preferred to a random numbering of the nodes. This result accords well with our intuition, since in a systematic numbering scheme the effect of changing the k-vector associated with a given node will be passed on rather quickly to some neighboring nodes; there is certainly no guarantee that such will be the case for randomly assigned numbering schemes.

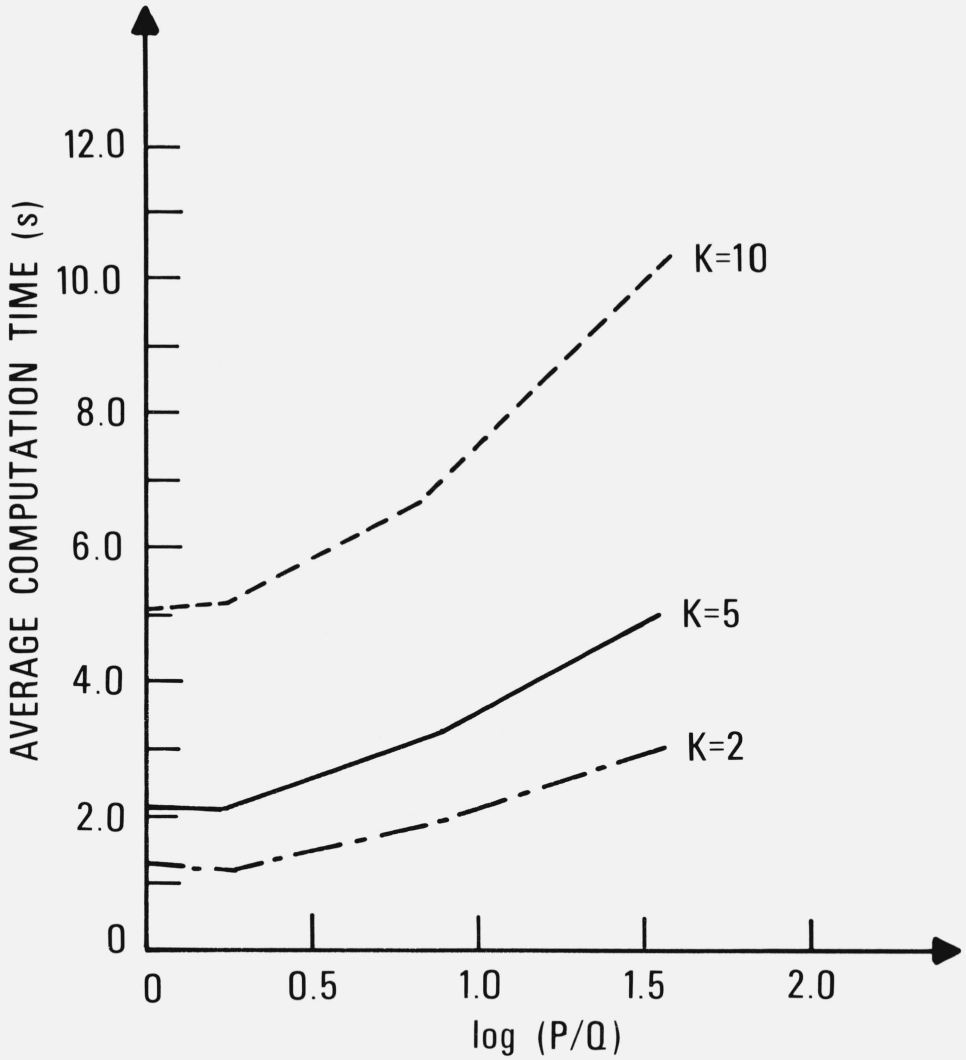

FIGURE 6. *Average computation time for* 24×24, 32×18, 64×9 *and* 144×4 *grids* (NS = *1*, RANGE = *100*, K = *2*, *5*, *10*).

554-917 OL - 74 - 4

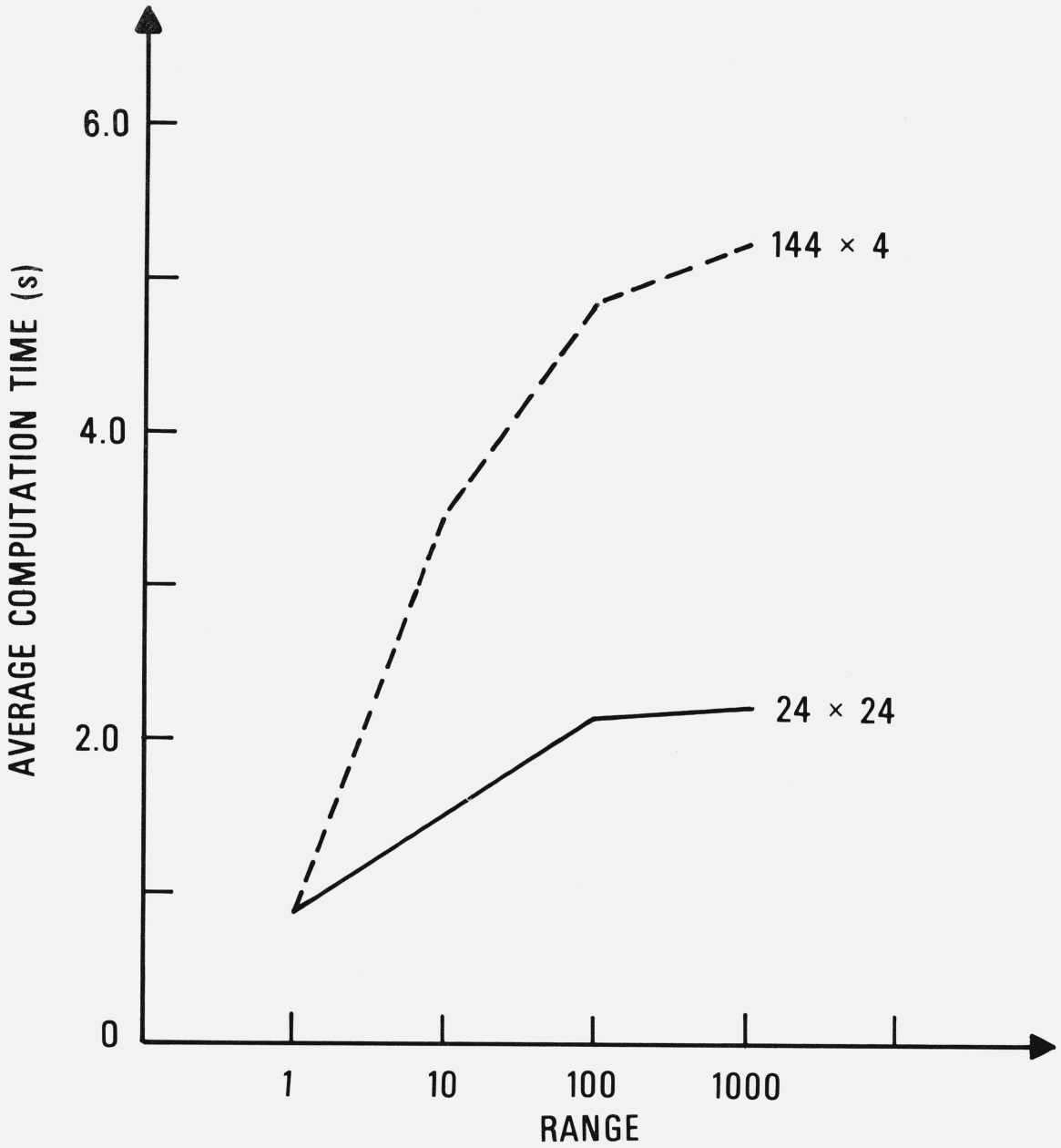

FIGURE 7. *Average computation time related to* RANGE *for* 24×24 *and* 144×4 *grids* (NS = *1*, K = *5*).

In concluding, some indication will be given of the actual number of iterations required for the Double-Sweep algorithm to converge; recall that each forward or backward pass constitutes an iteration. While the algorithm is guaranteed to converge in at most $2(NK+1)$ iterations [4], this theoretical upper bound usually exceeds the actual number of iterations required by a few orders of magnitude in our sample grid networks. For example, in all of the 24×24 grid networks generated with $RANGE = 100$ and $NS = 1$, 300, 576 (see figs.2–4), the number of iterations varied from a minimum of 9 (for $K = 2$) to a maximum of 25 (for $K = 20$). By contrast, the theoretical upper bounds for these two situations are 2306 and 23,042 iterations, respectively. In the case of $P \times Q$ grid networks with $N = 576$, $NS = 1$ and $RANGE = 100$ (see fig. 6), the number of iterations varied from a minimum of 9 (for a 32×18 grid with $K = 2$) to a maximum of 37 (for a 144×4 grid with $K = 10$); the theoretical upper bounds are 2306 and 11,522, respectively. For the 24×24 and 144×4 grid networks with $NS = 1$ and $K = 5$ (see fig. 7), the number of iterations varied from a minimum of

5 (for all 24×24 and 144×4 grids with $RANGE = 1$) to a maximum of 39 iterations (for a 144×4 grid with $RANGE = 1000$); the theoretical upper bound for this situation is 5762 iterations. Finally, even for the 900 node square grids with $NS = 1$, $K = 5$ and $RANGE = 100$ (see fig. 8), the number of iterations was at most 19, compared to the theoretical upper bound of 9002. In all these cases, then, the number of iterations required for convergence is really quite modest; in addition, the theoretical upper bounds are seen to give very little qualitative or quantitative information about the actual number of iterations needed.

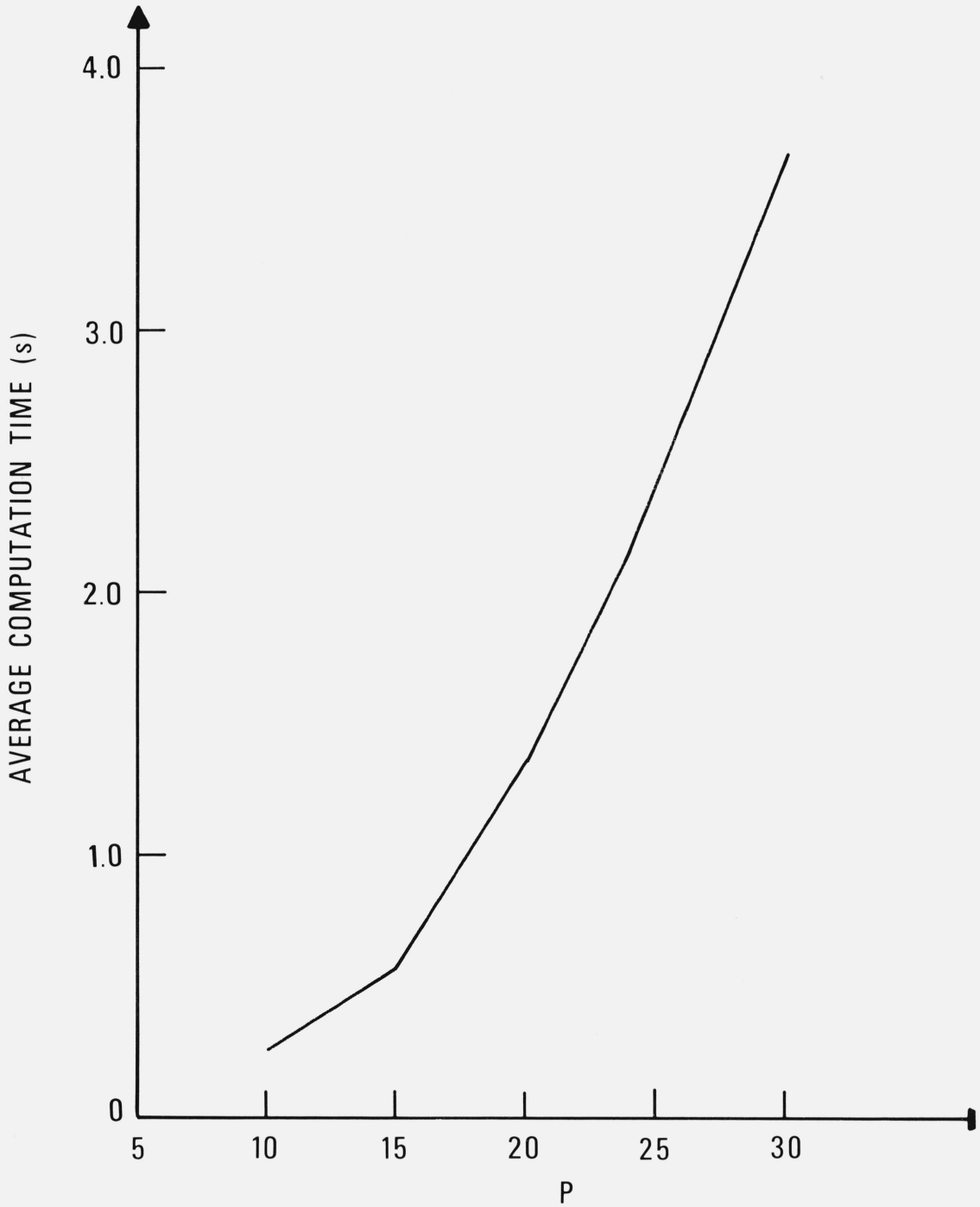

FIGURE 8. *Average computation time for five* P × P *grids* (NS = *1*, RANGE = *100*, K = *5*).

```
      SUBROUTINE INPUT
C
C
C  THIS SUBROUTINE INVOLVES THE FOLLOWING INPUT SEQUENCE.
C
C    (1) A DESCRIPTION OF THE NETWORK IS READ IN FROM UNIT 5 - THE
C        STARTING NODE, ENDING NODE AND LENGTH FOR EACH ARC.  THE
C        ARRAYS AND VARIABLES NEEDED BY SUBROUTINES DSWP, XMULT AND
C        TRACE ARE THEN CREATED.
C    (2) THE NEXT RECORD READ IN FROM UNIT 5 GIVES VALUES FOR K, NS
C        AND IMAX.  THE K SHORTEST (DISTINCT) PATH LENGTHS FROM NODE NS
C        TO ALL NODES OF THE NETWORK ARE THEN CALCULATED AND PRINTED
C        THROUGH A CALL OF DSWP.
C    (3) THE SUCCEEDING INPUT RECORDS READ IN FROM UNIT 5 INDICATE
C        VALUES FOR NF AND PMAX.  FOR EACH SUCH RECORD (THERE MAY BE
C        NONE) THE APPROPRIATE PATHS FROM NS TO NF ARE PRINTED THROUGH
C        A CALL OF TRACE.
C
C  SEVERAL SUCCESSIVE INPUT DATA SETS, CONSISTING OF A (2) RECORD
C  POSSIBLY FOLLOWED BY (3) RECORDS, CAN BE ACCOMMODATED.  A BLANK
C  RECORD SHOULD PRECEDE AND FOLLOW THE TOTALITY OF ALL SUCH (2), (3)
C  COMBINATIONS.  THE NETWORK MUST BE SORTED IN INCREASING ORDER BY
C  ARC ENDING NODE NUMBER.  MOREOVER IT IS ASSUMED THAT THE NODES
C  ARE NUMBERED SEQUENTIALLY FROM 1 TO N.  ALSO THE NETWORK SHOULD
C  CONTAIN NO SELF-LOOPS AND ALL CIRCUITS IN THE NETWORK ARE REQUIRED
C  TO HAVE POSITIVE LENGTH.
C
C  THE VARIABLES AND ARRAYS IN COMMON ARE
C
C     N      = THE NUMBER OF NODES IN THE NETWORK.
C     MU     = THE NUMBER OF ARCS (I,J) WITH I LESS THAN J.
C     ML     = THE NUMBER OF ARCS (I,J) WITH J LESS THAN I.
C     LLEN   = AN ARRAY WHOSE J-TH ENTRY IS THE NUMBER OF ARCS (I,J)
C              WITH J LESS THAN I.
C     LINC   = AN ARRAY CONTAINING THE NODES I INCIDENT TO NODE J WITH
C              I GREATER THAN J, LISTED IN ORDER OF INCREASING J.
C     LVAL   = AN ARRAY CONTAINING THE ARC LENGTH VALUES CORRESPONDING
C              TO ARCS IN LINC.
C     ULEN   = AN ARRAY WHOSE J-TH ENTRY IS THE NUMBER OF ARCS (I,J)
C              WITH I LESS THAN J.
C     UINC   = AN ARRAY CONTAINING THE NODES I INCIDENT TO NODE J WITH
C              I LESS THAN J, LISTED IN ORDER OF INCREASING J.
C     UVAL   = AN ARRAY CONTAINING THE ARC LENGTH VALUES CORRESPONDING
C              TO ARCS IN UINC.
C     INF    = A NUMBER LARGER THAN ANY PATH LENGTH.  A NONEXISTENT
C              PATH IS ASSIGNED THE LENGTH INF.
C     START  = AN ARRAY WHOSE J-TH ELEMENT INDICATES THE FIRST POSITION
C              ON INC WHERE NODES INCIDENT TO NODE J ARE LISTED.
C     INC    = AN ARRAY CONTAINING NODES I WHICH ARE INCIDENT TO NODE J,
C              LISTED IN ORDER OF INCREASING J.
C     VAL    = AN ARRAY CONTAINING THE ARC LENGTH VALUES CORRESPONDING
C              TO ARCS IN INC.
C
C  ADDITIONAL VARIABLES WHOSE VALUES MUST BE SPECIFIED BY THE USER ARE
```

```
C
C      K      = THE NUMBER OF DISTINCT PATH LENGTHS REQUIRED.
C      IMAX   = THE MAXIMUM NUMBER OF DOUBLE-SWEEP ITERATIONS ALLOWED.
C               FOR MOST CASES IMAX = 100 SHOULD SUFFICE.
C      NS,NF  = THE INITIAL AND FINAL NODES OF ALL K SHORTEST PATHS TO
C               BE GENERATED.
C      PMAX   = THE MAXIMUM NUMBER OF PATHS TO BE GENERATED BETWEEN
C               NODES NS AND NF.
C
C   THE FOLLOWING VARIABLES MUST LIE WITHIN THE (INCLUSIVE) RANGES.
C
C         VARIABLE    LOWER BOUND    UPPER BOUND
C
C             N             2             1000
C            MU             0             5000
C            ML             0             5000
C             K             2               20
C
C   THE FORMAT FOR THE INPUT NETWORK IS AS FOLLOWS.
C
C         COLS.       CONTENTS
C
C          1-10       ARC STARTING NODE
C         11-20       ARC ENDING NODE
C         21-30       ARC LENGTH
C
C   THE NETWORK DESCRIPTION IS FOLLOWED BY A BLANK RECORD.  SUCCEEDING
C   RECORDS GIVE VALUES FOR K, NS AND IMAX ACCORDING TO
C
C         COLS.       CONTENTS
C
C           1-5       K
C          6-10       NS
C         11-15       IMAX
C
C   EACH (K,NS,IMAX) RECORD CAN BE FOLLOWED BY ANY NUMBER OF RECORDS
C   (POSSIBLY NONE) GIVING VALUES FOR NF AND PMAX ACCORDING TO
C
C         COLS.       CONTENTS
C
C           1-5       NF
C          6-10       PMAX
C
C   THIS PATTERN OF (K,NS,IMAX) AND (NF,PMAX) RECORDS CAN BE REPEATED
C   AS OFTEN AS IS REQUIRED.  THE FINAL DATA RECORD OF THE INPUT DECK
C   IS BLANK.
C
C
      DIMENSION LLEN(1000),LINC(5000),LVAL(5000)
      INTEGER ULEN(1000),UINC(5000),UVAL(5000),START,VAL
      COMMON /BLK1/ N,MU,ML,LLEN,LINC,LVAL,ULEN,UINC,UVAL
      COMMON /BLK2/ INF,K
      COMMON /BLK4/ START(1001),INC(10000),VAL(10000)
C
C   INF IS DEFINED.
C
      INF=99999999
C
```

```
C  AS THE INPUT NETWORK IS READ IN, THE VARIABLES AND ARRAYS NEEDED
C  BY DSWP AND TRACE ARE CREATED.
C
      J=0
      MU=0
      ML=0
      NPREV=0
      N=0
    1 READ (5,800) NB,NA,LEN
  800 FORMAT(3I10)
      IF(NA.GT.N) N=NA
      IF(NB.GT.N) N=NB
      IF(NA.EQ.NPREV) GO TO 10
      IF(NA.EQ.NPREV+1) GO TO 3
      IF(NA.EQ.0) GO TO 30
      L1=NPREV+1
      L2=NA-1
      DO 2 L=L1,L2
      START(L)=0
      ULEN(L)=0
    2 LLEN(L)=0
    3 IF(J.EQ.0) GO TO 5
      ULEN(NPREV)=JU
      LLEN(NPREV)=JL
    5 START(NA)=J+1
      JU=0
      JL=0
      NPREV=NA
   10 J=J+1
      INC(J)=NB
      VAL(J)=LEN
      IF(NB.GT.NA) GO TO 20
      MU=MU+1
      UINC(MU)=NB
      UVAL(MU)=LEN
      JU=JU+1
      GO TO 1
   20 ML=ML+1
      LINC(ML)=NB
      LVAL(ML)=LEN
      JL=JL+1
      GO TO 1
   30 START(NPREV+1)=J+1
      ULEN(NPREV)=JU
      LLEN(NPREV)=JL
C
C  THE (K,NS,IMAX) AND (NF,PMAX) DATA RECORDS ARE SUCCESSIVELY READ.
C
   40 READ (5,801) I1,I2,I3
  801 FORMAT(3I5)
      IF(I1.EQ.0) GO TO 100
      IF(I3.EQ.0) GO TO 50
      K=I1
      NS=I2
C
C  THE K SHORTEST DISTINCT PATH LENGTHS FROM NODE NS TO ALL NODES OF
C  THE NETWORK ARE CALCULATED.
C
```

```
      CALL DSWP(NS,I3)
      GO TO 40
C
C  UP TO PMAX OF THE PATHS HAVING THE K SHORTEST PATH LENGTHS FROM NODE
C  NS TO NODE NF ARE DETERMINED.
C
   50 CALL TRACE(NS,I1,I2)
      GO TO 40
  100 RETURN
      END
```

```
      SUBROUTINE DSWP(NS,IMAX)
C
C
C  THIS SUBROUTINE IMPLEMENTS THE DOUBLE-SWEEP METHOD IN ORDER TO
C  CALCULATE THE K SHORTEST (DISTINCT) PATH LENGTHS FROM NODE NS TO
C  ALL NODES OF A NETWORK.  THE NETWORK IS ASSUMED TO CONTAIN NO
C  SELF-LOOPS AND ALL CIRCUITS IN THE NETWORK ARE REQUIRED TO HAVE
C  POSITIVE LENGTH.  THE REQUIRED K SHORTEST PATH INFORMATION IS
C  PRINTED OUT ON UNIT 6.
C
C  THE VARIABLES IN THE CALLING SEQUENCE ARE
C
C     NS   = THE NODE FROM WHICH K SHORTEST PATH LENGTHS TO ALL NODES
C            ARE REQUIRED.
C     IMAX = THE MAXIMUM NUMBER OF DOUBLE-SWEEP ITERATIONS ALLOWED.
C
C  THE VARIABLES AND ARRAYS IN COMMON ARE
C
C     N    = THE NUMBER OF NODES IN THE NETWORK. (NODES ARE ASSUMED
C            NUMBERED SEQUENTIALLY FROM 1 TO N.)
C     MU   = THE NUMBER OF ARCS (I,J) WITH I LESS THAN J.
C     ML   = THE NUMBER OF ARCS (I,J) WITH J LESS THAN I.
C     LLEN = AN ARRAY WHOSE J-TH ENTRY IS THE NUMBER OF ARCS (I,J)
C            WITH J LESS THAN I.
C     LINC = AN ARRAY CONTAINING THE NODES I INCIDENT TO NODE J WITH
C            I GREATER THAN J, LISTED IN ORDER OF INCREASING J.
C     LVAL = AN ARRAY CONTAINING THE ARC LENGTH VALUES CORRESPONDING
C            TO ARCS IN LINC.
C     ULEN = AN ARRAY WHOSE J-TH ENTRY IS THE NUMBER OF ARCS (I,J)
C            WITH I LESS THAN J.
C     UINC = AN ARRAY CONTAINING THE NODES I INCIDENT TO NODE J WITH
C            I LESS THAN J, LISTED IN ORDER OF INCREASING J.
C     UVAL = AN ARRAY CONTAINING THE ARC LENGTH VALUES CORRESPONDING
C            TO ARCS IN UINC.
C     INF  = A NUMBER LARGER THAN ANY PATH LENGTH.  A NONEXISTENT
C            PATH IS ASSIGNED THE LENGTH INF.
C     K    = THE NUMBER OF DISTINCT PATH LENGTHS REQUIRED.
C     X    = AN ARRAY WHOSE (I,J)-TH ENTRY WILL EVENTUALLY CONTAIN
C            THE J-TH SHORTEST (DISTINCT) PATH LENGTH FROM NODE NS
C            TO NODE I.
C
C  ADDITIONAL VARIABLES ARE
C
C     ITNS = THE NUMBER OF ITERATIONS PERFORMED.
C     INDX = A VARIABLE RETURNED FROM XMULT WHICH = 1 IF CONVERGENCE
C            HAS OBTAINED AND = 0 OTHERWISE.
C
C  THE FOLLOWING VARIABLES MUST LIE WITHIN THE (INCLUSIVE) RANGES
C
C        VARIABLE   LOWER BOUND   UPPER BOUND
C
C           N            2           1000
C          MU            0           5000
C          ML            0           5000
```

```
C             K            2              20
C
C
      DIMENSION LLEN(1000),LINC(5000),LVAL(5000)
      INTEGER ULEN(1000),UINC(5000),UVAL(5000),X
      COMMON /BLK1/ N,MU,ML,LLEN,LINC,LVAL,ULEN,UINC,UVAL
      COMMON /BLK2/ INF,K
      COMMON /BLK3/ X(1000,20)
      N1=N-1
C
C  THE INITIAL APPROXIMATION MATRIX X IS FORMED.
C
      DO 20 I=1,N
      DO 20 J=1,K
   20 X(I,J)=INF
      X(NS,1)=0
      ITNS=1
C
C  THE CURRENT X IS MODIFIED THROUGH MATRIX MULTIPLICATION WITH THE
C  LOWER TRIANGULAR PORTION OF THE ARC LENGTH MATRIX.
C
   30 IFIN=ML
      INDX=1
      DO 40 I=N1,1,-1
      IF(LLEN(I).EQ.0) GO TO 40
      IS=IFIN-LLEN(I)+1
      CALL XMULT(I,IS,IFIN,LINC,LVAL,INDX)
      IFIN=IS-1
   40 CONTINUE
      IF(ITNS.EQ.1) GO TO 50
C
C  TEST FOR CONVERGENCE.
C
      IF(INDX.EQ.1) GO TO 100
C
C  THE CURRENT X IS MODIFIED THROUGH MATRIX MULTIPLICATION WITH THE
C  UPPER TRIANGULAR PORTION OF THE ARC LENGTH MATRIX.
C
   50 ITNS=ITNS+1
      IS=1
      INDX=1
      DO 60 I=2,N
      IF(ULEN(I).EQ.0) GO TO 60
      IFIN=IS+ULEN(I)-1
      CALL XMULT(I,IS,IFIN,UINC,UVAL,INDX)
      IS=IFIN+1
   60 CONTINUE
C
C  TEST FOR CONVERGENCE.
C
      IF(INDX.EQ.1) GO TO 100
      ITNS=ITNS+1
C
C  A TEST IS MADE TO SEE IF TOO MANY ITERATIONS HAVE BEEN PERFORMED.
C
      IF(ITNS.LT.IMAX) GO TO 30
      WRITE (6,900) IMAX
  900 FORMAT(' NUMBER OF ITERATIONS EXCEEDS',I5)
```

```
      GO TO 200
C
C  THE SOLUTION MATRIX X IS PRINTED OUT ON UNIT 6, TOGETHER WITH THE
C  VALUES FOR K, NS AND ITNS.
C
  100 WRITE (6,901) K,NS
  901 FORMAT(1H1,10X,'K=',I3,' SHORTEST PATH LENGTHS FROM NODE',I4//2X,
     1'TO'/1X,'NODE'//)
      DO 130 I=1,N
  130 WRITE (6,902) I,(X(I,J),J=1,K)
  902 FORMAT(' ',I3,6X,(10I9))
      WRITE (6,903) ITNS
  903 FORMAT(//1HO,'NUMBER OF ITERATIONS REQUIRED FOR CONVERGENCE =',I5)
  200 RETURN
      END
```

```
      SUBROUTINE XMULT(I,IS,IFIN,INC,VAL,INDX)
C
C
C  THIS SUBROUTINE WILL PERFORM THE APPROPRIATE MATRIX MULTIPLICATION
C  OF X BY THE I-TH COLUMN OF THE LOWER (OR UPPER) PORTION OF THE ARC
C  LENGTH MATRIX.  IN EFFECT, THE CURRENT K SHORTEST PATH LENGTHS
C  ASSOCIATED WITH NODE I ARE ADJUSTED BY CONSIDERING NODES WHICH
C  ARE INCIDENT TO NODE I.  IF AN IMPROVEMENT CAN BE MADE, THE VARIABLE
C  INDX WILL RETURN WITH THE VALUE 0.
C
C  THE VARIABLES AND ARRAYS IN THE CALLING SEQUENCE ARE
C
C     I    = THE COLUMN OF THE ARC LENGTH MATRIX TO BE MULTIPLIED ON
C            THE LEFT BY X.  THE CURRENT K SHORTEST PATH LENGTHS TO
C            NODE I WILL BE IMPROVED IF POSSIBLE.
C     IS   = THE STARTING POSITION IN LISTS INC AND VAL WHERE ARC
C            INFORMATION FOR NODE I CAN BE FOUND.
C     IFIN = THE FINAL POSITION IN LISTS INC AND VAL WHERE ARC
C            INFORMATION FOR NODE I CAN BE FOUND.
C     INC  = AN ARRAY CONTAINING NODE INCIDENCE INFORMATION, EITHER
C            LINC OR UINC.
C     VAL  = AN ARRAY CONTAINING ARC LENGTH INFORMATION, EITHER
C            LVAL OR UVAL.
C     INDX = A VARIABLE WHICH IS SET EQUAL TO 0 IF AN IMPROVEMENT CAN
C            BE MADE IN THE K SHORTEST PATH LENGTHS TO NODE I.
C
C  THE VARIABLES AND ARRAYS IN COMMON ARE
C
C     INF  = A NUMBER LARGER THAN ANY PATH LENGTH.  A NONEXISTENT PATH
C            IS ASSIGNED THE LENGTH INF.
C     K    = THE NUMBER OF DISTINCT PATH LENGTHS REQUIRED.
C     X    = THE CURRENT APPROXIMATION MATRIX FOR K SHORTEST PATH
C            LENGTHS FROM A GIVEN NODE TO ALL NODES OF THE NETWORK.
C
C  ADDITIONAL VARIABLES AND ARRAYS ARE
C
C     A    = AN AUXILIARY ARRAY USED IN FINDING THE K SMALLEST DISTINCT
C            ELEMENTS OF A SET.
C     MAX  = THE CURRENT MAXIMUM ELEMENT OF ARRAY A.
C     IXV  = A FEASIBLE PATH LENGTH TO NODE I FROM THE GIVEN NODE.
C
C
      DIMENSION INC(5000)
      INTEGER VAL(5000),A(20),X
      COMMON /BLK2/ INF,K
      COMMON /BLK3/ X(1000,20)
C
C  INITIALIZE A TO THE CURRENT K SHORTEST PATH LENGTHS FOR NODE I, IN
C  STRICTLY INCREASING ORDER.
C
      DO 10 J=1,K
   10 A(J)=X(I,J)
      MAX=A(K)
C
```

```
C  EACH NODE OF INC INCIDENT TO NODE I IS EXAMINED.
C
      DO 100 L=IS,IFIN
      II=INC(L)
      IV=VAL(L)
C
C  TEST TO SEE WHETHER IXV IS TOO LARGE TO BE INSERTED INTO A.
C
      DO 90 M=1,K
      IX=X(II,M)
      IF(IX.GE.INF) GO TO 100
      IXV=IX+IV
      IF(IXV.GE.MAX) GO TO 100
C
C  IDENTIFY THE POSITION INTO WHICH IXV CAN BE INSERTED.
C
      DO 30 J=K,2,-1
      IF(IXV-A(J-1)) 30,90,50
   30 CONTINUE
      J=1
   50 JJ=K
   70 IF(JJ.LE.J) GO TO 80
      A(JJ)=A(JJ-1)
      JJ=JJ-1
      GO TO 70
   80 A(J)=IXV
C
C  IF AN INSERTION HAS BEEN MADE IN A, SET INDX = 0.
C
      INDX=0
      MAX=A(K)
   90 CONTINUE
  100 CONTINUE
      IF(INDX.EQ.1) GO TO 120
C
C  UPDATE THE K SHORTEST PATH LENGTHS TO NODE I.
C
      DO 110 J=1,K
  110 X(I,J)=A(J)
  120 RETURN
      END
```

```
      SUBROUTINE TRACE(NS,NF,PMAX)
C
C
C  THIS SUBROUTINE WILL TRACE OUT THE PATHS CORRESPONDING TO THE K
C  DISTINCT SHORTEST PATH LENGTHS FROM NODE NS TO NODE NF.  AT MOST
C  PMAX SUCH PATHS WILL BE GENERATED.  IT IS ASSUMED THAT ALL CIRCUITS
C  IN THE NETWORK HAVE POSITIVE LENGTH.  MOREOVER ONLY PATHS HAVING
C  AT MOST 5000 ARCS WILL BE PRODUCED.  AN ERROR MESSAGE WILL INDICATE
C  WHEN THIS CONDITION IS NOT FULFILLED.  THE REQUIRED PATHS BETWEEN
C  NODES NS AND NF ARE PRINTED OUT ON UNIT 6.
C
C  THE VARIABLES IN THE CALLING SEQUENCE ARE
C
C     NS,NF = THE INITIAL AND FINAL NODES OF ALL K SHORTEST PATHS
C             BEING GENERATED.
C     PMAX  = THE MAXIMUM NUMBER OF PATHS TO BE GENERATED BETWEEN
C             NODES NS AND NF.
C
C  THE VARIABLES AND ARRAYS IN COMMON ARE
C
C     INF   = A NUMBER LARGER THAN ANY PATH LENGTH.  A NONEXISTENT PATH
C             IS ASSIGNED THE LENGTH INF.
C     K     = THE NUMBER OF DISTINCT PATH LENGTHS REQUIRED.  K SHOULD
C             LIE IN THE RANGE 2 TO 20 INCLUSIVE.
C     X     = AN ARRAY WHOSE (I,J)-TH ENTRY IS THE J-TH SHORTEST
C             (DISTINCT) PATH LENGTH FROM NODE NS TO NODE I.
C     START = AN ARRAY WHOSE J-TH ELEMENT INDICATES THE FIRST POSITION
C             ON INC WHERE NODES INCIDENT TO NODE J ARE LISTED.
C     INC   = AN ARRAY CONTAINING NODES I WHICH ARE INCIDENT TO NODE J,
C             LISTED IN ORDER OF INCREASING J.
C     VAL   = AN ARRAY CONTAINING THE ARC LENGTH VALUES CORRESPONDING
C             TO ARCS IN INC.
C
C  ADDITIONAL VARIABLES AND ARRAYS ARE
C
C     JJ    = INDEX OF THE PATH LENGTH FROM NS TO NF BEING EXPLORED.
C             JJ CAN TAKE ON VALUES FROM 1 TO K.
C     NP    = THE NUMBER OF PATHS FROM NS TO NF FOUND.
C     KK    = CURRENT POSITION OF LIST P.
C     P     = AN ARRAY CONTAINING NODES ON A POSSIBLE PATH FROM NS TO NF.
C     Q     = AN ARRAY WHOSE I-TH ELEMENT GIVES THE POSITION, RELATIVE
C             TO START, OF NODE P(I) ON THE INC LIST FOR P(I-1).
C     PV    = AN ARRAY WHOSE I-TH ELEMENT IS THE ARC LENGTH EXTENDING
C             FROM NODE P(I) TO NODE P(I-1).
C
C
      INTEGER P(5000),Q(5000),PV(5000),START,VAL,X,PMAX
      COMMON /BLK2/ INF,K
      COMMON /BLK3/ X(1000,20)
      COMMON /BLK4/ START(1001),INC(10000),VAL(10000)
C
C  INITIALIZATION PHASE.
C
      DO 10 I=1,5000
```

```
      P(I)=0
      Q(I)=0
   10 PV(I)=0
      JJ=1
      IF(NS.EQ.NF) JJ=2
      NP=0
      IF(X(NF,JJ).LT.INF) GO TO 15
      WRITE (6,909) NS,NF
  909 FORMAT(1H1,'THERE ARE NO PATHS FROM NODE',I4,' TO NODE',I4)
      GO TO 200
   15 WRITE(6,901) NS,NF
  901 FORMAT(1H1,'THE K SHORTEST PATHS FROM NODE',I4,' TO NODE',I4//1H0,
     1'PATH   LENGTH   NODE SEQUENCE'//)
C
C  THE JJ-TH DISTINCT PATH LENGTH IS BEING EXPLORED.
C
   20 KK=1
      LAB=X(NF,JJ)
      IF(LAB.EQ.INF) GO TO 200
      LL=LAB
      P(1)=NF
   30 LAST=0
C
C  NODES INCIDENT TO NODE P(KK) ARE SCANNED.
C
   40 NT=P(KK)
      IS=START(NT)
      DO 45 ND=NT,1000
      IF(START(ND+1).NE.0) GO TO 48
   45 CONTINUE
   48 IF=START(ND+1)-1
      II=IS+LAST
   50 IF(II.GT.IF) GO TO 90
      NI=INC(II)
      NV=VAL(II)
      LT=LAB-NV
C
C  A TEST IS MADE TO SEE IF THE CURRENT PATH CAN BE EXTENDED BACK TO
C  NODE NI.
C
      DO 60 J=1,K
      IF(X(NI,J)-LT) 60,80,70
   60 CONTINUE
   70 II=II+1
      GO TO 50
   80 KK=KK+1
      IF(KK.GT.5000) GO TO 190
      P(KK)=NI
      Q(KK)=II-IS+1
      PV(KK)=NV
      LAB=LT
C
C  TESTS ARE MADE TO SEE IF THE CURRENT PATH CAN BE EXTENDED FURTHER.
C
      IF(LAB.NE.0) GO TO 30
      IF(NI.NE.NS) GO TO 30
C
C  A COMPLETE PATH FROM NS TO NF HAS BEEN GENERATED AND IS PRINTED
```

```
C  OUT ON UNIT 6.
C
      NP=NP+1
      WRITE (6,902) NP,LL,(P(J),J=KK,1,-1)
  902 FORMAT(1X, I4,I8,5X,(20I5))
      IF(NP.GE.PMAX) GO TO 200
   90 LAST=Q(KK)
      P(KK)=0
      LAB=LAB+PV(KK)
      KK=KK-1
      IF(KK.GT.0) GO TO 40
C
C  THE EXPLORATION OF THE CURRENT JJ-TH DISTINCT PATH LENGTH IS ENDED.
C
      JJ=JJ+1
      IF(JJ.GT.K) GO TO 200
      GO TO 20
  190 WRITE(6,903)
  903 FORMAT(1H0,'NUMBER OF ARCS IN PATH EXCEEDS 5000')
  200 RETURN
      END
```

6. References

[1] Dreyfus, S., An Appraisal of Some Shortest-Path Algorithms, Operations Res. **17**, 395–412 (1969).

[2] Minieka, E., and Shier, D., A Note on an Algebra for the k Best Routes in a Network, J. Inst. Maths Applics **11**, 145–149 (1973).

[3] Rosen, S. (editor), Programming Systems and Languages, Ch. 2B (McGraw-Hill Book Co., New York, 1967).

[4] Shier, D., Iterative Methods for Determining the k Shortest Paths in a Network, submitted for publication.

(Paper 78B3–410)