ATTACHMENT E

SAND77-1060
Unlimited Release

SPTH3: A Subroutine for Finding Shortest Sabotage Paths

Bernie L. Hulme, Diane B. Holdridge

Prepared by Sandia Laboratories, Albuquerque, New Mexico 87115
and Livermore, California 94550 for the United States Nuclear
Regulatory Commission under ERDA Contract AT(29-1)-789.

Printed July 1977

SF 2900 Q(7-73)

SAND 77-1060
Unlimited Release
Printed July 1977

SPTH3: A SUBROUTINE FOR FINDING SHORTEST SABOTAGE PATHS

Bernie L. Hulme
Numerical Mathematics Division

Diane B. Holdridge†
Applied Mathematics Division

Sandia Laboratories
Albuquerque, New Mexico 87115

ABSTRACT

This document explains how to construct a sabotage graph which models any fixed-site facility and how to use the subroutine SPTH3 to find shortest paths in the graph. The shortest sabotage paths represent physical routes through the site which would allow an adversary to take advantage of the greatest weaknesses in the system of barriers and alarms. The subroutine SPTH3 is a tool with which safeguards designers and analysts can study the relative effects of design changes on the adversary routing problem. In addition to showing how to use SPTH3, this report discusses the methods used to find shortest paths and several implementation details which cause SPTH3 to be extremely efficient.

†Presently in Nuclear Waste Technology Division.

Printed in the United States of America

Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22161
Price: $4.00 Microfiche $3.00

TABLE OF CONTENTS

1. Introduction

This report documents the latest code for finding shortest paths in the sabotage graphs described in [4]. A sabotage graph is a network which models a fixed-site facility. Shortests paths in the graph represent physical routes along which a saboteur could minimize time, or detection probability, or some other quantity reflected in the graph weights. Currently path length in SPTH3 is the ordinary sum of the node and arc weights in the path, so that either delay times (for barrier penetration at nodes and travel along arcs) or else distances could be used as weights. So far shortest-time paths have been of principal interest. A trivial modification to SPTH3 would allow it to accept detection probability weights and produce sabotage paths which minimize cumulative detection probability. In the remainder of the report we shall think of the graph weights as times.

SPTH3 addresses the problem of a simultaneous attack by several teams each having a single target. This provides a lower bound for the path length of a single team with several targets to attack sequentially, and it also addresses a very real possibility which would place great stress upon the safeguards system. For further details motivating this model and considering the length-independence of different shortest-time paths, see [4].

It must be understood that the graph model and the pathfinding techniques do not take into account battles between adversaries and defenders. Such encounters require stochastic models. The sabotage graphs used for pathfinding have constant weights which give representative (perhaps minimum or average) values of delay times. Since the delay times are in reality random variables, the graph-theoretic method of this

report should be viewed as a deterministic approach to finding sabotage routes which exploit any weaknesses in the barrier and alarm systems. Such paths require further evaluation either by simulation methods such as FESEM [2] to assess the paths' effects upon guard encounters or else by probabilistic methods such as EASI [1] to predict the likelihood of sabotage interruption. Both of these path evaluators require a path to be given. Thus, SPTH3 may be used to derive input for FESEM or EASI or else simply to indicate relative vulnerabilities in the barrier and alarm systems.

This document explains both what SPTH3 does and how to use it. Those readers interested only in how to use it may ignore Section 6. The shortest path algorithm embedded in SPTH3 is that of Dijkstra [3] as modified by Yen [6,5]. This algorithm is the best available. However, its use in SPTH3 differs somewhat from the description given in [4]. Rather than searching outward to the boundary from each target as described in [4], SPTH3 searches inward from the boundary to all nodes. A substantial reduction in storage requirements and tenfold reductions in run time have resulted from this and other improvements to the pathfinding code.

2. Description of SPTH3

SPTH3 finds shortest paths in a special graph called a sabotage graph. This graph models a fixed-site facility to some level of detail thought to be appropriate for the user's purpose. The details for constructing the graph are given in the next section. In brief, the nodes are important locations in the plant (say, perimeter gates, building doors, windows, vents, stairwells, storage vaults, and vital equipment locations), and the arcs are physical paths from one location to another.

There are three types of nodes: (i) boundary nodes located at possible perimeter penetration points, (ii) barrier nodes located on internal barriers at possible penetration points, and (iii) target nodes* at vital equipment or material locations. The simultaneous sabotage problem is to find all the shortest paths from the set of boundary nodes to each target. Given the graph (as a list of arcs) and the arc and node weights, SPTH3 uses the Dijkstra-Yen algorithm described in Section 6.5 to search from the boundary nodes of the graph until the lengths of the shortest paths to all the other nodes are known. During the search SPTH3 keeps a list of the immediate predecessor(s) of each node along a shortest path from the boundary. This allows all shortest paths to each target to be retraced and stored without further arithmetic following the Dijkstra-Yen search.

The output from SPTH3 is simply a list of the directed arcs which belong to the shortest paths directed from the boundary to all the targets, together with a list of the number and length of the shortest paths to each target. It is possible to have more than one shortest path to any node, and this number can be obtained easily by a simple procedure explained in Section 6.7.

3. The Sabotage Graph

The first and most important part of the use of SPTH3 is the construction and weighting of the sabotage graph. This involves three steps:

(1) partitioning the drawings of the plant into regions,

---

*Formerly called hardware nodes.

(2) specifying the nodes and arcs of the graph model,

(3) weighting the nodes and arcs with times (or detection probabilities) derived from test data or analyst judgement.

3.1 The Regions

The boundary and the important internal barriers (fences, walls, floors, stairwells, etc.) naturally partition a map of the site into regions R_r, $r = 1,2,\ldots$. A region is a very general area within which saboteurs may travel unimpeded by barriers. Some regions may appear on the drawing as disjoint domains, e.g., a stairwell or elevator region may appear as the union of the areas in which it intersects each floor of a building. However, if there are no significant delays to entering the stairwell, then the floors and the connecting stairwell may be treated as one region. The region structure is simply an aid to constructing the nodes and arcs which constitute the sabotage graph.

3.2. The Graph G

Next, the analyst must carefully place nodes at all the important locations. Since the model is discrete there is some arbitrariness in the node selection process, and an analyst may want to try various graphs differing in the number and location of the nodes. The node set should include representative penetration points along the boundary and along the barriers between regions as well as all targets of interest inside the regions.

Once the regions and the nodes are specified, the arcs are determined by a fixed rule which gives sabotage graphs their special structure. Every pair of nodes in region R_r is connected by an arc, forming a complete subgraph G_r, $r = 1,2,\ldots$.

An interface between two regions may contain more than one barrier node, and this requires special attention from the analyst. Two reasons for using such multiple barrier nodes are to model barriers that (a) have varying hardness, or (b) have such physical extent that the arc lengths are significantly affected by the node locations. An arc joining node i to node j is denoted by the unordered integer pair (i,j) or (j,i). Multiple barrier nodes on a single barrier cause an arc in one region to have the same name, or node pair, as some arc in the adjacent region (Figure 1a). In order that each arc have a unique node pair as its endpoints, the analyst must split each of the multiple barrier nodes into two barrier nodes connected by an arc (Figure 1b). Our decision to list the arcs of G region by region for purposes of computer input requires that each arc belong to a single region. Consequently each arc introduced by splitting multiple barrier nodes must belong to its own specially created region.

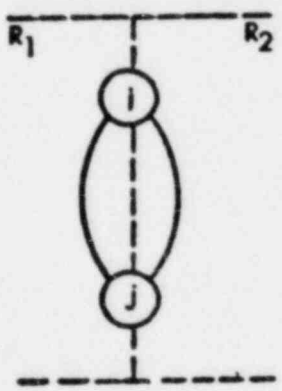

Figure 1a.
Multiple barrier nodes.

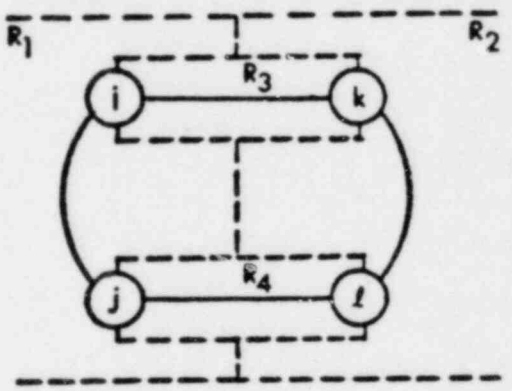

Figure 1b.
Splitting of multiple barrier nodes.

The regions created by splitting multiple barrier nodes are numbered also, giving a total of N_R regions R_r, each having a complete subgraph G_r.

The union of all these subgraphs is the sabotage graph*

$$G = \bigcup_{r=1}^{N_R} G_r ,$$

whose nodes are then numbered as follows:

target nodes, 1 to n_1,

barrier nodes, $n_1 + 1$ to $n_1 + n_2$,

boundary nodes, $n_1 + n_2 + 1$ to $n_1 + n_2 + n_3 = N$.

An example is shown in Figure 2, where squares are used for boundary nodes, circles for barrier nodes, and shaded circles for targets. Three nodes on the barrier $R_1 \cap R_2$ have been split.

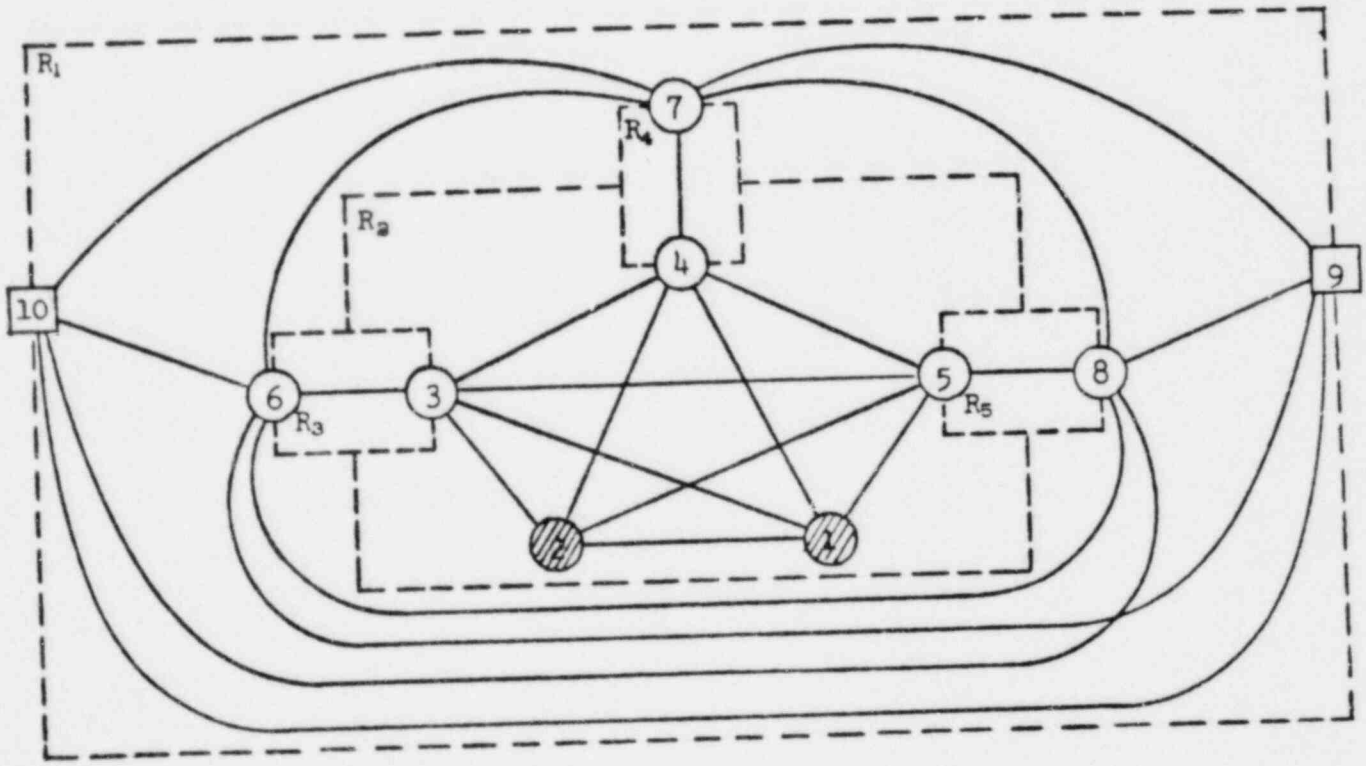

Figure 2.

A Sabotage Graph

*This graph differs slightly from the graph of [4] in which boundary-boundary and target-target arcs were omitted.

2

Notice that every barrier and boundary node belongs to exactly two regions (counting the infinite off-site region), and every target node belongs to only one region. Also, due to the completeness of the subgraphc, the paths of G represent all the physically meaningful ways for saboteurs to proceed using only the given penetration points and targets.

3.3. The Weights

Both the nodes and the arcs of G are assigned constant, nonnegative weights. The node weights $w_i \geq 0$, $1 \leq i \leq N$, are penetration and target destruction times, while the arc weights $a_{i,j} = a_{j,i} \geq 0$, $(i,j) \in G$, are transit times. Realizing that these constants are simply representative values of random variables that depend on the physical characteristics of the barriers and the amount and type of equipment postulated for the attacking force, the analyst must decide whether to be conservative and use minimum values reflecting best possible adversary performance or else use intermediate or average values.

When the arc weights are minimum values, they will automatically satisfy a regional triangle inequality. That is, if arcs (i,j), (j,k) and (k,i) belong to G_r and their weights represent minimal transit times, then it must be true that

$$a_{i,k} \leq a_{i,j} + a_{j,k} . \tag{1}$$

Since saboteurs may travel at different rates in different regions, this inequality need not hold for triangles whose arcs lie in different regions. SPTH3 tests all the triangle inequalities (1) for each region simply as a check on data consistency for the user's benefit. The pathfinding algorithm will perform perfectly well, of course, whether or not the triangle inequalities hold, so that this data check can be deleted from

SPTH3.

The weighting of the nodes and arcs which result from the splitting of multiple barrier nodes may be done in many ways as long as the weights of the two new nodes and their connecting arc sum to the barrier penetration time.

A weighting of the graph from Figure 2 is given in Figure 3.

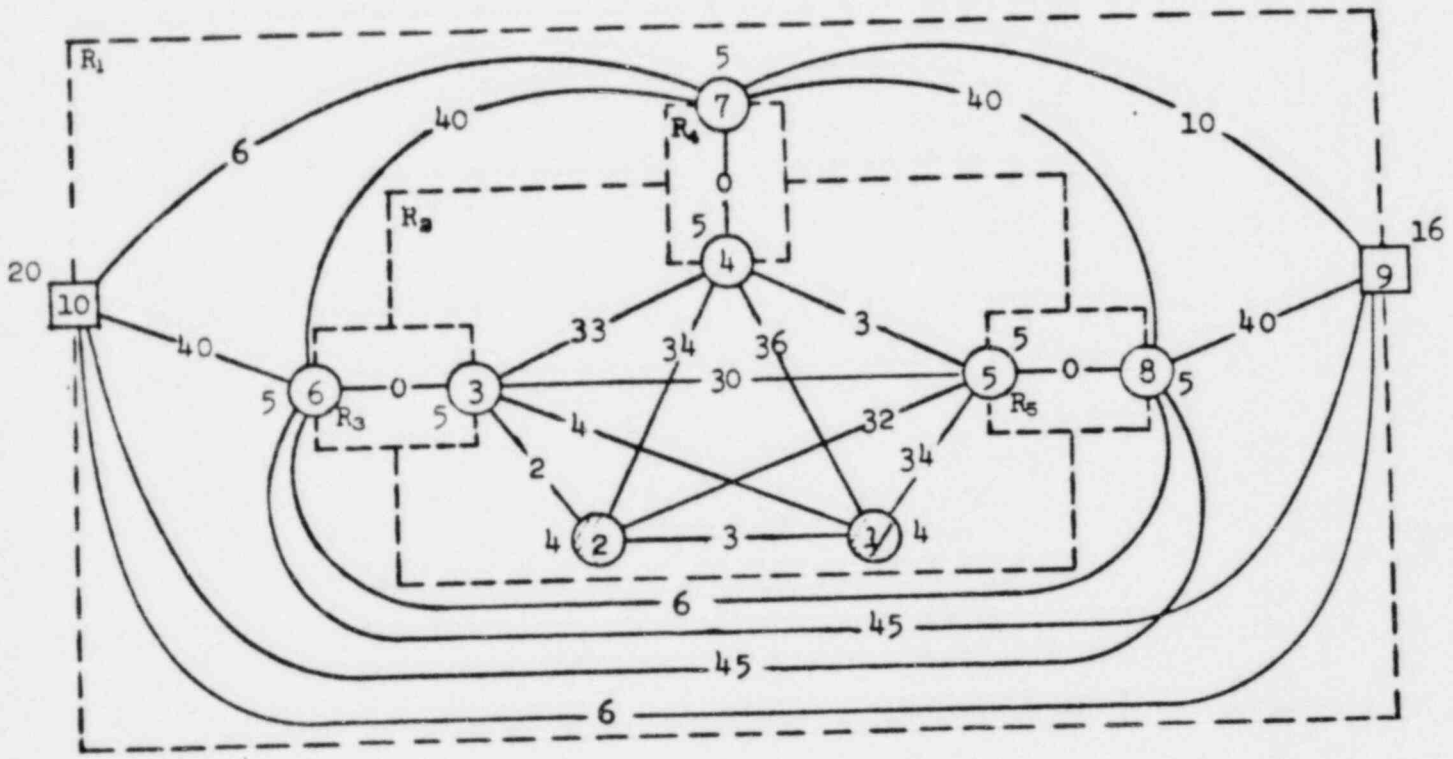

Figure 3.

A Weighted Sabotage Graph

SPTH3 does not accept directed nodes and arcs, i.e., nodes and arcs whose weights depend on the direction of travel. Of course, directed nodes and arcs could have been allowed, because Dijkstra's algorithm works equally well for directed graphs (digraphs) as for undirected graphs. However, we have deliberately omitted directedness from the sabotage

graphs because, in the applications for which SPTH3 is intended, arc lengths are essentially the same in both directions and the doors locked on only one side are normally locked on the side first encountered by the saboteur.

4. How to Use SPTH3

4.1. The Call List

The call list for subroutine SPTH3 is

SPTH3(N1,N2,N3,NA,W,MR,II,JJ,AWT,MAXE,IEDGE,NE,NSP,XMINL).

The dummy arguments have the following meanings:

N1 - the number of target nodes,

N2 - the number of barrier nodes,

N3 - the number of boundary nodes,

NA - the number of arcs,

W(·) - the node weight vector, dimensioned N=N1+N2+N3,

(the next four vectors, dimensioned NA, give the arcs as quadruples consisting of a region, two nodes, and an arc weight)

MR(·) - the region index vector,

II(·) - a node index vector,

JJ(·) - a node index vector,

AWT(·) - the arc weight vector,

MAXE - the maximum number of edges (arcs) in the digraph S of shortest paths directed from the boundary to all target nodes,

IEDGE(·,·) - the edges in the digraph S, each edge being given by three indices -- the region and two ordered nodes, dimensioned (3,MAXE),

NE - the number of edges in digraph S,

NSP(·) - the number of shortest paths to each target, dimensioned N1,

XMINL(·) - the length of the shortest paths to each target, dimensioned N1.

4.2. Input

To use SPTH3, first construct a weighted sabotage graph as indicated in Section 3. Next, in the program which calls SPTH3, dimension W by N, the vectors MR, II, JJ and AWT by NA, and the vectors NSP and XMINL by N1. Set MAXE to some guess at the maximum number of edges S will have, and dimension IEDGE by (3,MAXE). Then store the node weights in W and the arc data in MR, II, JJ and AWT. The node weights are given in the obvious order: W(I) for node I, $1 \le I \le N$. Although there is no obvious order in which to give the arc data, a very special ordering of the arcs is required.

The arcs are given by the quadruples

$$MR(K),\ II(K),\ JJ(K),\ AWT(K),\ 1 \le K \le NA\ .$$

All the arcs of one region are listed consecutively, and the regions may be given in any order. For example, in a three region problem, the arcs of region two could be listed first, followed by the arcs of regions three and one. The arcs of each region, however, must be listed as if they were taken row by row from the strictly upper triangular part of some node adjacency matrix. For example, if the nodes of one region are {16,9,21,4,7}, then an acceptable arc ordering based on the given node ordering is (16,9), (16,21), (16,4), (16,7), (9,21), (9,4), (9,7), (21,4), (21,7), (4,7). Notice that the arc ordering for a region may be based on any ordering of the region's nodes. But once a node ordering is chosen for the region, the arcs must be given by pairing the first node with each other node in order, then pairing the second node with each

following node in order, and similarly for the third node, etc.

The reason for this requirement is that it produces tremendous savings in storage. The special arc ordering allows SPTH3 to quickly compute the address of any arc weight and, thereby, completely eliminates the need for the usual N x N direct distance matrix. For graphs with several hundred nodes this is very important.

4.3. Work Arrays

SPTH3 has several work arrays whose dimensions must be set by the user before running the job. The meanings of these arrays are explained in the program comments and in Section 6. In order to use SPTH3 it is sufficient for the user to set the following dimensions:

XLABEL,NPATH,IPERM,ITEMP,NEXT - N,

NODE - (N,4),

IREG - (NR,2), where NR = the number of regions,

NPOOL - 40, an arbitrary setting for an unpredictable total number of extra predecessors for nodes which have more than one predecessor along shortest paths. SPTH3 prints a message when this dimension needs to be increased. In this case, the results should be considered incomplete, and the problem should be rerun with a larger dimension for NPOOL.

4.4. Output

The output consists of

MAXE,IEDGE,NE,NSP and XMINL ,

whose meanings are given above.

MAXE and NE serve as flags and must be tested upon return from SPTH3 to see if a normal execution took place. If MAXE = 0 upon return, there was a failure of the triangle inequality on the arc weights of some region, a message was printed, and the pathfinding algorithm was not

executed. The user should correct the arc data indicated by the message and try again. Also, it is possible to return with NE = 0. This means that pathfinding was aborted because some node could not be reached from the boundary. The user must check the arc list to be sure that all the arcs are present in each region.

It should be noted that the weight vectors W and AWT are changed by SPTH3 in the following way:

$$W(I) \leftarrow W(I)/2. \ , \quad N1 + 1 \le I \le N1 + N2 \ ,$$

$$AWT(K) \leftarrow AWT(K) + W\big(II(K)\big) + W\big(JJ(K)\big) \ , \quad 1 \le K \le NA \ .$$

If necessary, the user may restore W and AWT to their input values by first subtracting $W\big(II(K)\big) + W\big(JJ(K)\big)$ from AWT(K), for $1 \le K \le NA$, and then doubling each barrier node weight. This change of W and AWT is also related to the above mentioned storage economy because it allows the input vector AWT to be used for the direct distance storage in lieu of the standard N x N matrix normally used in Dijkstra's algorithm.

5. Examples

5.1. A Sample Problem

Let us take the weighted graph of Figure 3 as an example. In Figure 4 this graph is shown with the digraph S of shortest sabotage paths superimposed in dark lines.

The input consists of

$$N1 = 2, \quad N2 = 6, \quad N3 = 2, \quad NA = 23 \ ,$$

$$W = \{4.,4.,5.,5.,5.,5.,5.,5.,16.,20.\} \ ,$$

MR	II	JJ	AWT
1	6	7	40.
1	6	8	6.
1	6	9	45.
1	6	10	40.
1	7	8	40.
1	7	9	10.
1	7	10	6.
1	8	9	40.
1	8	10	45.
1	9	10	6.
2	1	2	3.
2	1	3	4.
2	1	4	36.
2	1	5	34.
2	2	3	2.
2	2	4	34.
2	2	5	32.
2	3	4	33.
2	3	5	30.
2	4	5	3.
3	3	6	0.
4	4	7	0.
5	5	8	0.

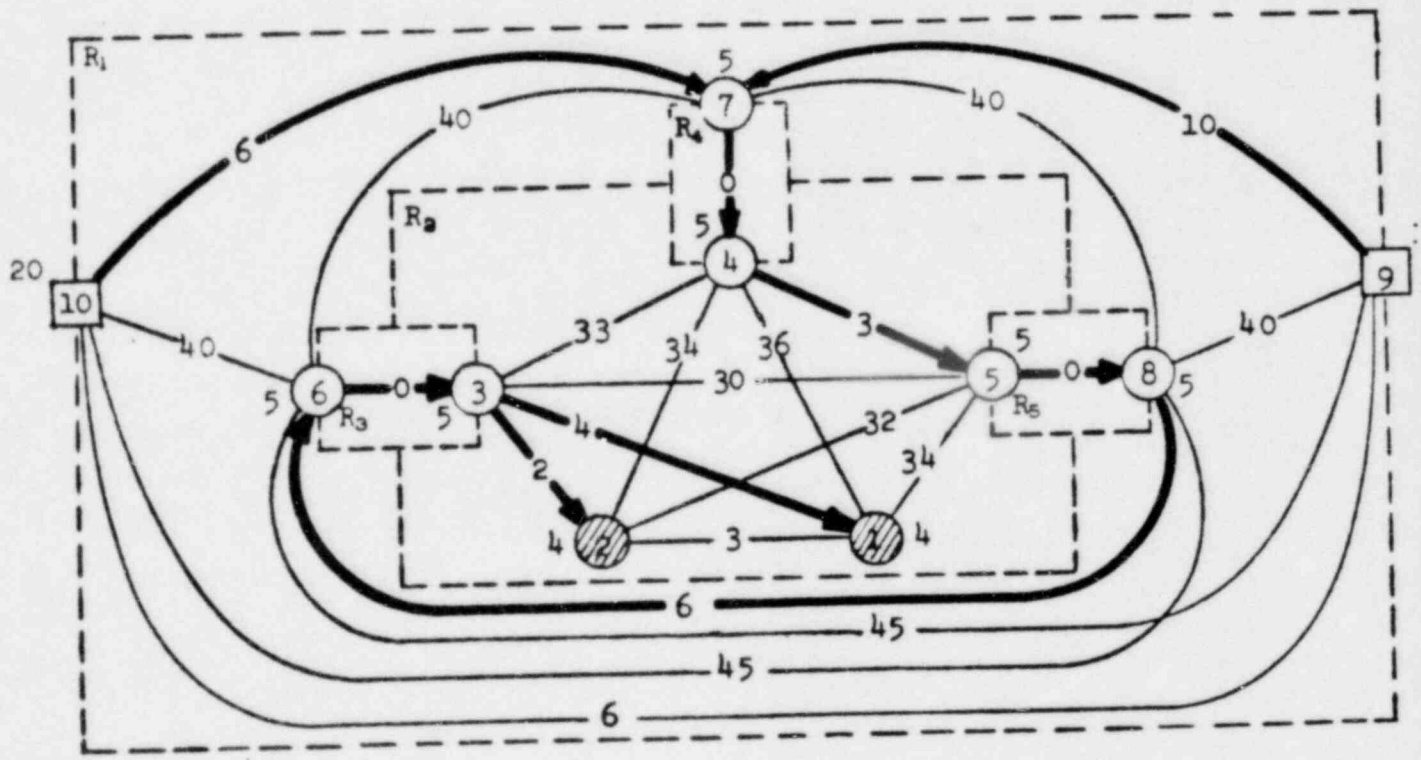

Figure 4.

The Digraph S of Shortest Sabotage Paths Superimposed on a Weighted Sabotage Graph

The output is

Target	NSP	XMINL
1	2	73.
2	2	71.

IEDGE (NE = 9)

Region	Node	Node
2	3	1
2	3	2
3	6	3
1	8	6
5	5	8
2	4	5
4	7	4
1	10	7
1	9	7 .

Since the digraph S is given completely by IEDGE, S is easily drawn by simply darkening the edges indicated by the ordered node pairs in IEDGE. NSP and XMINL show that there are two shortest paths to each target, those to node 1 having length 73. and those to node 2 having length 71.

5.2. Run Time and Array Storage Results

SPTH3 is extremely fast. The Dijkstra-Yen algorithm used for pathfinding has a worst-case run time on the order of $O(N^2)$ for an N-node graph. Although SPTH3 also checks all the triangle inequalities in each region, does the bookkeeping for multiple predecessors, and retraces and counts the number of shortest paths to each target, it uses almost a negligible amount of run time. As shown in Table I a realistic sized problem of 310 nodes and 1191 arcs requires only about a second and a half of CDC 6600 run time. Consequently, we feel that execution time for shortest path problems should be of very little concern to most users.

SPTH3 is also very storage efficient. As mentioned in Section 4.4, the N x N matrix of direct distances normally used in Dijkstra's algorithm

has been eliminated in favor of a storage scheme which overwrites the input vector AWT with direct distances between node centers. Thus, the array storage, which dominates SPTH3' storage requirements for larger problems, is linear in the number of nodes, arcs, regions, etc. In particular SPTH3's arrays need

$$10N + 4NA + 2NR + 2N1 + 3NE + 40$$

storage locations, where 40 is an unpredictable dimension that proved adequate for our set of test problems. Table I also gives the array storage requirements for the larger problems in the test set.

The sample problem in Figure 4 is Problem 3 below.

Table I

CDC 6600 Run Time and Array Storage Results

Problem	N1-N2-N3	Nodes N	Arcs NA	Regions NR	Edges in S NE	Array Storage	Run Time (seconds)
1	1-5-1	7	13	4	3	-	0.004
2	5-1-2	8	16	2	6	-	0.004
3	2-6-2	10	23	5	9	-	0.005
4	4-8-2	14	34	6	12	-	0.007
5	1-8-8	17	32	8	1	-	0.009
6	1-10-8	19	40	8	1	411	0.009
7	1-31-14	46	112	20	2	996	0.041
8	20-58-4	82	514	35	43	3155	0.167
9	30-120-5	155	656	82	81	4681	0.446
10	10-296-4	310	1191	192	38	8422	1.563

6. Some Details of Implementation

6.1. A Flow Chart

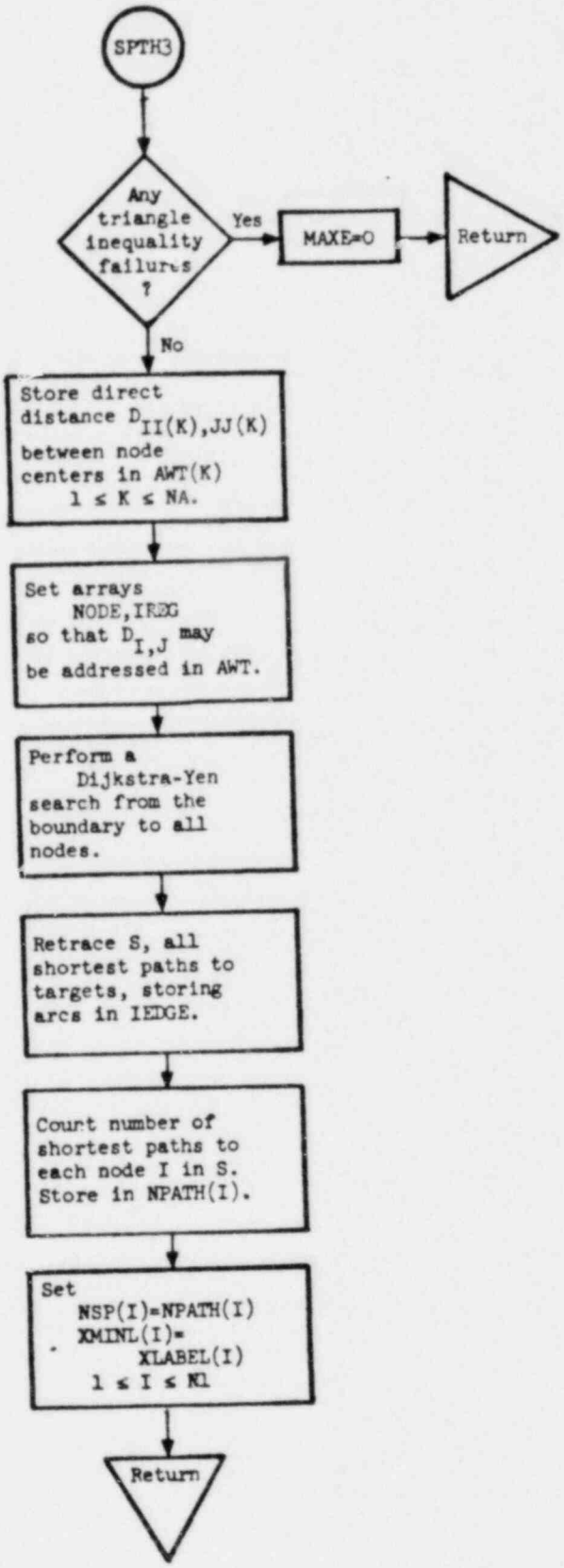

Figure 5.

A Flow Chart of SPTH3

6.2. The Triangle Inequality Tests

Finding the triangles of each region is very easy given the completeness of the subgraph and the special ordering of the arcs. For example, consider the arcs of region R_2 in the graph of Figure 4,

II	JJ	AWT
1	2	3.
1	3	4.
1	4	36.
1	5	34.
2	3	2.
2	4	34.
2	5	32.
3	4	33.
3	5	30.
4	5	3.

All the triangles containing arc (1,2) are found by taking each arc $(1,\ell)$ listed below (1,2) together with each arc $(2,\ell)$. In this case, the triangles are (1,2,3), (1,2,4) and (1,2,5). Next, all the triangles containing arc (1,3) are obtained by taking each arc $(1,\ell)$ listed below (1,3) together with each arc $(3,\ell)$. This yields triangles (1,3,4) and (1,3,5). Finally, triangle (1,4,5) is obtained in a similar way.

Of course, each triangle (i,j,k) has three corresponding inequalities which are tested separately

$$a_{i,j} \leq a_{i,k} + a_{k,j} ,$$

$$a_{i,k} \leq a_{i,j} + a_{j,k} ,$$

$$a_{j,k} \leq a_{j,i} + a_{i,k} .$$

As mentioned earlier, this portion of SPTH3 could be deleted without affecting the pathfinding procedure.

6.3. Computing Direct Distances

Dijkstra's algorithm is defined in terms of a direct distance matrix D, whose elements

$$(2) \qquad d_{i,j} = \begin{cases} \text{direct distance from } i \text{ to } j\ , & (i,j) \in G\ , \\ 0\ , & i = j\ , \\ \infty\ , & \text{otherwise.} \end{cases}$$

The algorithm applies to any digraph having weights only on the arcs, since the nodes are thought of as points. A sabotage graph has node weights w_i which SPTH3 combines with the arc weights $a_{i,j}$ in forming the distances $d_{i,j}$ between node centers. The procedure is to halve the barrier node weights (since these are the intermediate nodes on paths from the boundary to the targets) and then to add each arc weight to its endpoint weights, i.e.

$$(3) \qquad w_i = w_i/2.\ , \quad n_1 + 1 \le i \le n_1 + n_2\ ,$$

$$(4) \qquad d_{i,j} = a_{i,j} + w_i + w_j\ , \quad (i,j) \in G\ .$$

6.4. Storing and Accessing Direct Distances

SPTH3 avoids creating the N x N matrix D by storing the positive, finite, D values in the input arc weight vector AWT when (4) is computed, i.e

$$(5) \qquad \begin{aligned} &I = II(K)\ , \quad J = JJ(K) \\ &AWT(K) = AWT(K) + W(I) + W(J)\ , \quad 1 \le K \le NA\ . \end{aligned}$$

This saves a great deal of storage for large N because the number of arcs in G is always small with respect to N^2 (see Table I).

The penalty we pay for this storage efficiency is in the cost of finding any particular $d_{I,J}$ in AWT. That is, when Dijkstra's algorithm needs $d_{I,J}$, SPTH3 must find a value K such that

(6) $$AWT(K) = d_{I,J} ,$$

instead of just referencing D(I,J). Fortunately, the completeness of the subgraphs and the special ordering of arcs in each subgraph allows K to be computed very quickly given I and J (see the run times in Table I).

To facilitate this index computation, two integer arrays are constructed prior to the Dijkstra-Yen search. NODE(I,·) contains the two regions and two local node numbers for node I. The local node number in each region is determined by the node ordering used to order the arcs of the region. For the sample problem in Section 5.1,

$$NODE(7,\cdot) = \{1,2,4,2\} ,$$

meaning that node 7 is the second node of region 1 and the second node of region 4. Since only the finite, internal regions are numbered, boundary nodes have only one region number and one local node number. Similarly for target nodes which belong to only one region.

IREG(R,·) contains the first word address minus one in AWT of the arcs of region R followed by the number of nodes in R. Thus, when SPTH3 needs $d_{I,J}(I \neq J)$, it does the following:

(a) compares region numbers for I and J to see if arc (I,J) belongs to G,

(b) if $(I,J) \notin G$, no address computation is needed since $d_{I,J} = \infty$,

(c) if $(I,J) \in G_R$, then the first word address minus one of the arcs of region R, the number of nodes in region R, the local number of node I in R, ℓ_I, and the local number of node J in R, ℓ_J, are combined to yield

$$K = \begin{cases} IREG(R,1) + (\ell_I-1)IREG(R,2) - \ell_I(\ell_I+1)/2 + \ell_J , & \ell_I \leq \ell_J , \\ IREG(R,1) + (\ell_J-1)IREG(R,2) - \ell_J(\ell_J+1)/2 + \ell_I , & \ell_J < \ell_I . \end{cases}$$

K satisfies (6).

6.5. The Dijkstra-Yen Search

The Dijkstra-Yen search finds the length of the shortest paths from the boundary to every node I in G and stores the value in XMINL(I), $1 \le I \le N$. During the search the (immediate) predecessor of node J along a shortest path is stored in NEXT(J), $1 \le J \le N$. When some node has more than one such predecessor, the extras are stored in a vector NPOOL. A link (an index for NPOOL) is stored in the upper portion of the word NEXT(J) to indicate where the second predecessor of J is stored. This second predecessor, NPOOL(LINK), is linked to another entry in NPOOL if there is a third predecessor of J, etc.* These data allow the efficient retracing of all (not just one) shortest paths to each target, as explained in the next section.

In Dijkstra's algorithm [3] each node has a label which eventually becomes the length of the shortest paths from the boundary to the node. These labels are temporary as long as they represent only the shortest path lengths currently found by the search, and they become permanent labels as soon as they are known to be the absolutely shortest lengths.

SPTH3 initially sets all boundary node labels to zero and all other labels to ∞ (a large machine number). The boundary node label XLABEL(N) is made permanent first by setting IPERM(1)=N. All the other labels are temporary. From the last permanently labeled node I, all the temporary labels XLABEL(J) are examined and reduced if

$$\text{XLABEL(J)} > \text{XLABEL(I)} + d_{I,J} ,$$

where $d_{I,J}$ must be found in AWT as described in the previous section. Each

*We are indebted to Louann Grady, 5741, for giving us this idea for linking together multiple predecessors.

time XLABEL(J) is reduced, the predecessor I is stored in NEXT(J). If

$$XLABEL(J) = XLABEL(I) + d_{I,J},$$

(to machine precision), then I is an extra predecessor of J which must be stored in the next available entry of NPOOL, and for which an additional link must be created at the end of the chain beginning with the link in the upper portion of NEXT(J). After all the temporary nodes have been examined from I, the one with the least temporary label is made permanent by placing it in IPERM. This is correct because the nonnegativity of the $d_{I,J}$ implies that this label cannot be reduced further. In the case of a tie, it does not matter which of the nodes is permanently labeled next. I is set to this new permanent node number and the process is repeated until all the nodes are permanently labeled. Notice that IPERM has become a list of the nodes of G in order of nondecreasing distance from the boundary, and the shortest sabotage path lengths are XLABEL(J), $1 \le J \le N1$.

Yen's contribution [6] to this search procedure is one of improved implementation. Rather than letting J range from 1 to N at each stage and asking if each J is temporary, Yen suggests the following coding device. Initialize ITEMP(I) = I, $1 \le I \le N$, and K = N - 1. The temporary nodes, then, are the first K entries of ITEMP. While trying to reduce each temporary node label, keep track of the minimizing temporary node IP as well as its position IQ in ITEMP. Then, when IP is stored in IPERM, set ITEMP(IQ) = ITEMP(K). This has the double effect of removing IP from ITEMP and leaving the new set of temporary nodes in the first K entries of ITEMP. Yen's modification saves SPTH3 about 25% in run time.

If, because of omissions in arc data, some node is isolated from the boundary, its infinite label will eventually become the least temporary

label at some stage. When this happens, NE is set to zero and SPTH3 returns to the user, who must supply the missing data before trying again.

6.6. Retracing the Shortest Paths

Given the predecessors NEXT(·) and possibly some predecessors in NPOOL(·), SPTH3 can retrace the digraph S of all the shortest paths from the boundary to the targets. No label-arithmetic is needed. As each directed arc of S is obtained, it is stored as an ordered pair of integers in IEDGE(2,·) and IEDGE(3,·), and the arc's region number is stored in IEDGE(1,·). The retrace proceeds as follows.

Initialize NE = 0. Find the last target node in IPERM, i.e., the one furthest from the boundary, and set J to this node number. Let I = NEXT(J), the predecessor of J, assuming J has only one. Add one to NE, and if NE does not exceed MAXE, store arc (I,J) and its region number in IEDGE(·,NE). Now record the fact that a shortest sabotage path passes through I by negating ITEMP(I) if it is positive. If J has another predecessor, it is NPOOL(LINK), where LINK is packed in the upper portion of NEXT(J). In this case, set I = NPOOL(LINK), repeat the above arc-storage process and continue until all the predecessors of J contribute arcs to IEDGE. (Notice that all the arcs of S leading into J are listed consecutively in IEDGE.) Set J to the node in IPERM just before the one last used to set J, i.e., let J range over the nodes in the order opposite that in which they were made permanent. If J is a target node or a barrier node with negative ITEMP(J), then repeat the above procedure for this new J. If J is a boundary node or a barrier node that does not belong to a shortest sabotage path, then skip it, continuing until J = IPERM(1) has been treated.

Thus, IEDGE contains all the arcs in the digraph S of all shortest sabotage paths. Moreover, the second nodes of these arcs occur in the

order of nonincreasing distance from the boundary.

6.7. Counting the Shortest Paths

The fact that the second nodes in IEDGE have nonincreasing distance from the boundary allows the shortest paths to be counted quickly as follows. Set

$$\text{NPATH}(I) = \begin{cases} 1\ , & \text{boundary node,} \\ 0\ , & \text{barrier or target node.} \end{cases}$$

Then for each arc K in IEDGE, taken from bottom to top, i.e., from the boundary to the last target, set

```
      I = IEDGE(2,K)
      J = IEDGE(3,K)
NPATH(J) = NPATH(J) + NPATH(I),   K = NE,...,1 .
```

The final value of NPATH(J) is just the sum of NPATH(I) over all the nodes I in S which have an arc leading to J. Notice that each such node I will have its final NPATH value computed before NPATH(I) is added to NPATH(J) because of the ordering of the arcs in IEDGE. Hence, NPATH(J), is the number of shortest paths from the boundary to each node J in S, in particular to J = 1,2,...,N1.

7. Listing of SPTH3

```
      SUBROUTINE SPTH3(N1,N2,N3,NA,W,MR,II,JJ,AWT,MAXE,IEDGE,NE,NSP,
     1 XMINL)
      DIMENSION W(1),MR(1),II(1),JJ(1),AWT(1),IEDGE(3,1),NSP(1),XMINL(1)
      COMMON XLABEL(155),NPATH(155),IPERM(155),ITEMP(155),NODE(155,4),
     1 IREG(82,2),NEXT(155),NPOOL(40)
C     SIMULTANEOUS SABOTAGE PROBLEM--ONE TEAM PER HARDWARE NODE.  USES
C     DIJKSTRA-TYPE SABOTAGE ALGORITHM--UNDIRECTED NODES, TIME WEIGHTS.
C     SAVES SHORTEST PATHS FROM OFF-SITE TO EACH HARDWARE NODE.
C     SPTH3 SEARCHES INWARD AND USES NO DISTANCE MATRIX.
C     INPUT.
C       N1   NO. OF HARDWARE NODES.
C       N2   NO. OF BARRIER NODES.
C       N3   NO. OF BOUNDARY NODES.
C       NA   NO. OF ARCS.
C        W   NODE WEIGHT VECTOR, DIMENSIONED N=N1+N2+N3.
C            W IS CHANGED.
C     ARC DATA--FOUR NA VECTORS.  ARCS MUST BE LISTED REGION BY REGION.
C               FURTHERMORE, WITHIN EACH REGION HAVING P NODES THERE MUST
C               BE P*(P-1)/2 ARCS LISTED ROW BY ROW IN STRICTLY UPPER
C               TRIANGULAR FORM.  THAT IS, (I1,I2), (I1,I3), ..., (I1,IP),
C               (I2,I3), ..., (I2,IP), (I3,I4), ..., (I3,IP), ...,
C               (IPM1,IP).
C       MR   REGION INDEX VECTOR.
C       II   NODE INDEX VECTOR.
C       JJ   NODE INDEX VECTOR.
C      AWT   ARC WEIGHT VECTOR.  AWT IS CHANGED.
C     MAXE   MAXIMUM NO. EDGES (ARCS) IN THE DIGRAPH S OF SHORTEST PATHS
C            FROM OFF-SITE TO ALL HARDWARE NODES.
C            THIS VALUE IS THE SECOND DIMENSION OF IEDGE.
C     OUTPUT.
C     IEDGE  EDGES IN THE DIGRAPH S, THE UNION OF ALL SHORTEST PATHS
C            DIRECTED FROM THE BOUNDARY TO ALL HARDWARE NODES.  IEDGE WILL
C            HOLD MAXE EDGES, EACH BEING GIVEN BY 3 INDICES -- THE REGION,
C            AND TWO ORDERED NODES.  THE EDGES ARE LISTED IN IEDGE SO
C            THAT THE SECOND NODES HAVE A DECREASING DISTANCE FROM
C            OFF-SITE.
C       NE   NO. EDGES IN DIGRAPH S.
C      NSP   NO. SHORTEST PATHS TO H, H=1,2,...,N1.
C     XMINL  LENGTH OF SHORTEST PATHS TO H, H=1,2,...,N1.
C            THE DIMENSION OF NSP AND XMINL MUST BE AT LEAST AS LARGE AS N1
C     MAXE   IF MAXE=0 UPON EXIT, THERE WAS A FAILURE OF THE TRIANGLE
C            INEQUALITY ON THE ARC WEIGHTS OF A REGION, AND THE
C            ALGORITHM WAS NOT EXECUTED.
C     WORK ARRAYS
C     FOUR VECTORS DIMENSIONED N=N1+N2+N3.
C     XLABEL  TEMPORARY AND PERMANENT DISTANCE LABELS.  THESE LABELS
C             REPRESENT THE LENGTH OF THE CURRENTLY SHORTEST
C             PATHS FROM OFF-SITE TO EACH NODE.
C      NPATH  NUMBER OF SHORTEST PATHS FROM OFF-SITE TO EACH NODE OF S.
C      IPERM  NODES WHERE DISTANCE LABELS HAVE BEEN MADE PERMANENT.
C      ITEMP  NODES WHERE DISTANCE LABELS ARE STILL TEMPORARY.
C       NODE  REGION AND LOCAL NODE NUMBERS FOR EACH NODE.
C             DIMENSION (N,4).
C             NODE(I,1), NODE(I,3) ARE REGION NUMBERS FOR NODE I.
C             NODE(I,2), NODE(I,4) ARE CORRESPONDING LOCAL NODE NUMBERS.
C
```

```
C       IREG  REGION DATA CONCERNING ARCS.  DIMENSIONED (NO. REGIONS, 2).
C             IREG(R,1) IS THE FIRST WORD ADDRESS MINUS ONE IN THE ARC
C             LIST OF THE ARCS OF REGION R.
C             IREG(R,2) IS THE NUMBER OF NODES IN REGION R, IMPLYING
C             THERE ARE IREG(R,2)*(IREG(R,2)-1)/2 ARCS IN REGION R.
C       NEXT  PREDECESSOR OF EACH NODE J ALONG A CURRENTLY SHORTEST PATH
C             FROM OFF-SITE TO J.  DIMENSIONED N.
C      NPOOL  A LINKED LIST IN WHICH ADDITIONAL PREDECESSORS MAY BE
C             STORED WHEN NODE J HAS MORE THAN ONE.  THE LINK FROM
C             NEXT(J) TO NPOOL(LINK) IS STORED IN THE LEFT 51 BITS OF
C             NEXT(J).  SIMILARLY, IF THERE IS A THIRD PREDECESSOR OF
C             J, THEN LINK1 FROM NPOOL(LINK) TO NPOOL(LINK1) IS STORED
C             IN THE LEFT 51 BITS OF NPOOL(LINK), ETC.  DIMENSIONED 40.
C             IF THE DIMENSION OF NPOOL IS CHANGED, THEN THE FOURTH
C             STATEMENT NPLDP=41 MUST BE CHANGED.  NPLDP IS THE NPOOL
C             DIMENSION PLUS ONE.  IF THE DIMENSION N IS INCREASED TO
C             MORE THAN 511 NODES, THEN THE FIRST THREE STATEMENTS MUST
C             BE CHANGED TO ALLOW MORE THAN 9 BITS IN THE RIGHT OF
C             EACH MASK.
      DATA EPS,BIG / 1.0E-13,1.0E321 /
      LTEST=1000B
      LOW=777B
      IHIGH=777777777777777000B
      NPLDP=41
      MAXEP=MAXE+1
      OMEPS=1.0-EPS
      OPEPS=1.0+EPS
      N12=N1+N2
      N=N12+N3
      NM1=N-1
      N1P=N1+1
      N12P=N12+1
C     CHECK EACH REGION FOR TRIANGLE INEQUALITY ON ARC WEIGHTS
      ICT=0
      IREG=1
    5 IREGP=IREG+1
      IF(MR(IREG) .NE. MR(IREGP)) GO TO 52
      LIKE=II(IREG)
      DO 10 I=IREGP,NA
      IF(II(I) .NE. LIKE) GO TO 15
   10 CONTINUE
      GO TO 52
   15 NM=I-IREG
      IF(NM .LE. 1) GO TO 52
   20 I2=IREG+NM
      IEND=I2-2
      NMT=NM-1
      DO 50 I1=IREG,IEND
      AWTOM=AWT(I1)*OMEPS
      I3=I1+1
      DO 45 J=1,NMT
      IF(AWT(I2)+AWT(I3) .GE. AWTOM) GO TO 35
   25 FORMAT(* TRIANGLE*3I3* FAILS.  ARC WEIGHTS--*3E15.5)
   30 PRINT 25,II(I1),JJ(I1),JJ(I3),AWT(I1),AWT(I2),AWT(I3)
      ICT=1
```

```
      GO TO 40
   35 IF(AWT(I1)+AWT(I2) .LT. AWT(I3)*OMEPS) GO TO 30
      IF(AWT(I1)+AWT(I3) .LT. AWT(I2)*OMEPS) GO TO 30
   40 I2=I2+1
   45 I3=I3+1
   50 NMT=NMT-1
      NM=NM-1
      IREG=IEND+2
      IF(NM .GE. 2) GO TO 20
   52 IREG=IBEG+1
      IF(IBEG .LT. NA) GO TO 5
  290 IF(ICT .EQ. 0) GO TO 55
      MAXE=0
      RETURN
C   COMBINE NODE WEIGHTS INTO ARC WEIGHTS
   55 DO 65 I=N1P,N12
      W(I)=0.5*W(I)
   65 CONTINUE
      DO 68 IA=1,NA
      I=II(IA)
      J=JJ(IA)
      AWT(IA)=AWT(IA)+W(I)+W(J)
   68 CONTINUE
C   SET THE ARRAYS NODE, IREG.
      DO 70 I=1,N
      NODE(I,1)=0
   70 NODE(I,3)=0
      L=1
   72 K=1
      IR=MR(L)
      IREG(IR,1)=L-1
      I=II(L)
      IF(NODE(I,1) .EQ. 0) GO TO 73
      NODE(I,3)=IR
      NODE(I,4)=K
      GO TO 74
   73 NODE(I,1)=IR
      NODE(I,2)=K
   74 K=K+1
      J=JJ(L)
      IF(NODE(J,1) .EQ. 0) GO TO 75
      NODE(J,3)=IR
      NODE(J,4)=K
      GO TO 76
   75 NODE(J,1)=IR
      NODE(J,2)=K
   76 IF(L .EQ. NA) GO TO 77
      L=L+1
      IF((I .EQ. II(L)) .AND. (IR .EQ. MR(L))) GO TO 74
      IREG(IR,2)=K
      L=L-K+K*(K-1)/2+1
      IF(L .LE. NA) GO TO 72
   77 IREG(IR,2)=K
C   DIJKSTRA-YEN SEARCH INWARD.
C   INITIALIZE.
```

```
      DO 125 I=1,N12
      XLABEL(I)=BIG
      NPATH(I)=0
      ITEMP(I)=I
  125 CONTINUE
      DO 127 I=N12P,N
      XLABEL(I)=0.
      NPATH(I)=1
      ITEMP(I)=I
  127 CONTINUE
      IPL=1
      L=1
C     PERMANENTLY LABEL NODE N.
      IPERM(1)=N
      I=N
      IR=NODE(I,1)
      LI=NODE(I,2)
      M=IREG(IR,1)+(LI-1)*IREG(IR,2)-LI*(LI+1)/2
      IR2=0
      K=NM1
      V=BIG
C     TREAT EACH TEMPORARILY LABELED NODE.
C     V IS THE SMALLEST SUCH LABEL.
  130 DO 140 IT=1,K
      J=ITEMP(IT)
      IF(NODE(J,1) .NE. IR) GO TO 131
      LJ=NODE(J,2)
      GO TO 132
  131 IF(NODE(J,3) .NE. IR) GO TO 133
      LJ=NODE(J,4)
  132 IF(LI .GT. LJ) GO TO 138
      IAR=M+LJ
      GO TO 137
  138 IAR=IREG(IR,1)+(LJ-1)*IREG(IR,2)-LJ*(LJ+1)/2+LI
      GO TO 137
  133 IF(NODE(J,1) .NE. IR2) GO TO 134
      LJ=NODE(J,2)
      GO TO 136
  134 IF(NODE(J,3) .NE. IR2) GO TO 135
      IF(IR2 .EQ. 0) GO TO 135
      LJ=NODE(J,4)
  136 IF(LI2 .GT. LJ) GO TO 139
      IAR=M2+LJ
      GO TO 137
  139 IAR=IREG(IR2,1)+(LJ-1)*IREG(IR2,2)-LJ*(LJ+1)/2+LI2
  137 DIJ=AWT(IAR)
      Z=XLABEL(I)+DIJ
      XJPEPS=XLABEL(J)*OPEPS
      IF(Z .GT. XJPEPS) GO TO 135
      XJMEPS=XLABEL(J)*OMEPS
      IF(Z .GE. XJMEPS) GO TO 300
      XLABEL(J)=Z
      NEXT(J)=I
      GO TO 135
  300 IF(IPL-NPLDP) 305,302,340
```

```
301   FORMAT(* NPOOL NEEDS TO STORE MORE LINKS*)
302   PRINT 301
      GO TO 340
305   NPRED=NEXT(J)
310   IF(NPRED .LT. LTEST) GO TO 320
      LINK=SHIFT(NPRED .AND. IHIGH,-9)
      NPRED=NPOOL(LINK)
      GO TO 310
320   NPOOL(IPL)=I
      I1=SHIFT(IPL,9) .OR. NPRED
      IF(NPRED .EQ. NEXT(J)) GO TO 330
      NPOOL(LINK)=I1
      GO TO 340
330   NEXT(J)=I1
340   IPL=IPL+1
135   IF(XLABEL(J) .GE. V) GO TO 140
      V=XLABEL(J)
      IP=J
      IQ=IT
140   CONTINUE
      IF(V .NE. BIG) GO TO 155
      NE=0
      DO 152 I=1,N1
      NSP(I)=0
      XMINL(I)=BIG
152   CONTINUE
      RETURN
C     NODE IP IS TO BE PERMANENTLY LABELED.
155   V=BIG
      L=L+1
      IPERM(L)=IP
      I=IP
      IR=NODE(I,1)
      LI=NODE(I,2)
      M=IREG(IR,1)+(LI-1)*IREG(IR,2)-LI*(LI+1)/2
      IR2=NODE(I,3)
      LI2=NODE(I,4)
      M2=IREG(IR2,1)+(LI2-1)*IREG(IR2,2)-LI2*(LI2+1)/2
      ITEMP(IQ)=ITEMP(K)
      K=K-1
      IF(K .GT. 0) GO TO 130
C     ALL NODES ARE PERMANENTLY LABELED.
C     RETRACE AND STORE THE SHORTEST PATHS TO ALL HARDWARE NODES.
      NE=0
180   J=IPERM(L)
      L=L-1
      IF(J .GT. N1) GO TO 180
182   I=NEXT(J)
183   NPRED=I
      IF(I .GE. LTEST) I=I .AND. LOW
      NE=NE+1
      IF(NE-MAXEP) 193,192,205
191   FORMAT(* DIGRAPH OF SHORTEST PATHS CONTAINS MORE THAN* I3
     1 * EDGES*)
192   PRINT 191, MAXE
```

```
      GO TO 205
  193 IEDGE(2,NE)=I
      IEDGE(3,NE)=J
      IR=NODE(J-1)
      IR2=NODE(J,3)
      IF((NODE(I,1) .EQ. IR) .OR. (NODE(I,3) .EQ. IR)) IR2=IR
      IEDGE(1,NE)=IR2
      IT=ITEMP(I)
      IF(IT .GT. 0) ITEMP(I)=-IT
  205 IF(I .EQ. NPRED) GO TO 215
      LINK=SHIFT(NPRED .AND. IHIGH,-9)
      I=NPOOL(LINK)
      GO TO 183
  215 IF(L .EQ. 0) GO TO 220
      J=IPERM(L)
      L=L-1
      IF(J .LE. N1) GO TO 182
      IF(J .GT. N12) GO TO 215
      IF(ITEMP(J) .LT. 0) GO TO 182
      GO TO 215
C     COUNT THE SHORTEST PATHS TO EACH HARDWARE NODE.
  220 K=NE
      DO 225 L=1,NE
      I=IEDGE(2,K)
      J=IEDGE(3,K)
      NPATH(J)=NPATH(J)+NPATH(I)
      K=K-1
  225 CONTINUE
      DO 230 I=1,N1
      NSP(I)=NPATH(I)
      XMINL(I)=XLABEL(I)
  230 CONTINUE
      RETURN
      END
```

Acknowledgements

We want to thank S. L. Daniel, 5411, and G. B. Varnado, 5412, for supplying us with some of the test problems and for their helpful comments concerning the use of SPTH3.

REFERENCES

[1] H. A. Bennett, The "EASI" Approach to Physical Security Evaluation, SAND76-0500, Sandia Laboratories, Albuquerque, New Mexico, January 1977.

[2] L. D. Chapman, Effectiveness Evaluation of Alternative Fixed-Site Safeguard Security Systems, SAND75-6159, Sandia Laboratories, Albuquerque, New Mexico, July 1976.

[3] E. W. Dijkstra, A Note on Two Problems in Connexion with Graphs, Numer. Math. 1, 269-271 (1959).

[4] B. L. Hulme, Pathfinding in Graph-Theoretic Sabotage Models. I. Simultaneous Attack by Several Teams, SAND76-0314, Sandia Laboratories, Albuquerque, New Mexico, July 1976.

[5] T. A. Williams and G. P. White, A Note on Yen's Algorithm for Finding the Length of All Shortest Paths in N-Node Nonnegative-Distance Networks, J. Assoc. Comput. Mach. 20, 389-390 (1973).

[6] J. Y. Yen, Finding the Lengths of All Shortest Paths in N-Node Nonnegative-Distance Complete Networks Using $\frac{1}{2}N^3$ Additions and N^3 Comparisons, J. Assoc. Comput. Mach. 19, 423-424 (1972).

DISTRIBUTION:

USNRC Distribution Section
Attn: Robert Wade
Washington, DC 20555
NRC-13 (208)

1000 G. A. Fowler
1140 W. D. Weart
1141 L. R. Hill
1141 D. B. Holdridge (10)
1230 W. L. Stevens
1233 R. E. Smith
1233 M. D. Olman
1310 A. A. Lieber
Attn: W. F. Roherty, 1311
1700 O. E. Jones
1710 V. E. Blake
1712 J. W. Kane
1738 J. Jacobs
1739 J. D. Williams
1739 D. L. Mangan
1750 J. E. Stiegler
1750A M. N. Cravens
1750A J. M. Demontmollin
1751 T. A. Sellers
1751 J. L. Darbey
1751 B. R. Fenchel
1751 L. C. Nogales
1751 A. E. Winblad
1752 M. R. Madsen
1754 J. F. Ney
1754 J. L. Todd, Jr.
1758 T. J. Hoban
1758 D. D. Boozer
1758 R. C. Hall
1758 G. A. Kinemond
1758 W. K. Paulus
1758 I. G. Waddoups
1758 R. B. Worrell
4010 C. Winter
5000 A. Narath, Attn: Directorates 5200, 5800
5100 J. K. Galt
5110 F. L. Vook
5120 G. J. Simmons
5121 R. J. Thompson
5121 P. J. Slater
5122 L. F. Shampine
5122 B. L. Hulme (50)
5130 G. A. Samara
5150 J. E. Schirber
5160 W. Herrman
5400 A. W. Snyder
5410 D. J. McCloskey
5411 S. L. Daniel
5412 J. W. Hickman
5412 D. E. Bennett
5412 D. M. Ericson, Jr.
5412 G. B. Varnado
5700 J. H. Scott
5740 V. L. Dugan
5741 L. D. Chapman
5741 K. G. Adams
5741 H. A. Bennett
5741 D. Engi
5741 L. M. Grady
5741 R. D. Jones
5741 R. G. Roosen
5741 D. W. Sasser
5741 A. A. Trujillo
5742 S. G. Varnado
8300 B. F. Murphey
8320 T. S. Gold
8321 R. L. Rinne
8266 E. A. Aas (2)
3141 C. A. Pepmueller (Actg.) (5)
3151 W. L. Garner (3)
For ERDA/TIC (Unlimited Release)
3171-1 R. Campbell (25)
For ERDA/TIC