# A COMPUTER NETWORK SIMULATION UTILIZING GRAPH THEORY TO CALCULATE MEASURES OF EFFECTIVENESS

by

RUSSELL DEAN THOMAS

B.S., Kansas State University, 1983

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCES

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

Eddie R. Fowler
Major Professor

## ACKNOWLEDGEMENTS

The author would like to thank the Air Force Systems Command, the Air Force Office of Scientific Research, and the Southeastern Center for Electrical Engineering Education for providing him with the opportunity to spend two very worthwhile and interesting summers at the Air Force Weapons Laboratory (AFWL). He would also like to acknowledge the Air Force Weapons Laboratory Nuclear Technology Directorate for providing an interesting research problem, a well-stocked Technical Library, and the use of the NTC VAX 11/780. Furthermore, he would like to thank his fellow AFWL researchers Dr. Eddie Fowler, Ron Hightower, and Kurt Ziegler for their assistance.

He would like to thank the Department of Electrical and Computer Engineering for providing him with the opportunity, facilities, and the use of a Z-100 to complete the research. Additionally, he would like to thank the members of his graduate committee Dr. Eddie R. Fowler, Dr. Everett E. Haft, and Dr. Virgil E. Wallentine for all of their help and guidance. Finally, he would like to thank Mary Langman for her careful typing of this report and his lovely wife Dee Anne for her support through it all.

TABLE OF CONTENTS

## 1.0 Introduction

When dealing with computer networks that serve the Department of Defense (DOD), the effects of a wartime load and degradation on the network must always be considered. Many measures which calculate the effectiveness of the network after degradation are called Measures of Effectiveness (MOE's). Measures which determine the survivability of a network after degradation are called Measures of Survivability (MOS's).

Most of the MOE's can be found using graphical methods. Some of these MOE's come directly from well-known algorithms written to describe a network graphically. For example, Dijkstra's Shortest Path First Algorithm and Ford and Fulkerson's Minimum-Cut Maximum-Flow Algorithm are the best known of these.

The program GRAFTHY was written to meet the need for a simulation model which calculates the MOE's of a computer network. To better appreciate how GRAFTHY operates, it is necessary to have a good understanding of how computer networks are modeled graphically and represented in the program and how each MOE and MOS is defined and calculated. All of these concepts are discussed in this paper.

## 1.1 Background and Motivation

The United States Air Force (USAF) operational capability is dependent upon its Command, Control, Communications, and Intelligence (C3I) systems, and computer networks are an important part of these systems. A 1974 DOD study projected that by the 1980's there would be approximately 2,500 computers and 20,000 terminals used by the DOD community which would require data-communications facilities (1). Therefore, there is motivation to model these computer networks and test the models to ensure that the computer networks will be effective in their tasks.

## 1.3 Objectives and Scope

The objective of this paper is to model computer networks and determine their effectiveness using the graph theory approach. Although a graph theory background is helpful, it is not necessary for an understanding of this paper. However, a good understanding of matrix theory, boolean algebra, and computer programming is necessary for understanding some concepts and all programs included.

## 2.0 Characterizing Computer Networks Using Graphical Methods

A computer network is an "interconnection of autonomous computers" geographically remote from one another (2). These computer networks can be modeled with nodes and links, where the nodes are the computers and the links are the leased land lines, radio frequency links, satellite links, etc. which interconnect the computers. For example, Figures 2.1, 2.2, and 2.3 show graphical models for two real world networks: the Advanced Research Projects Agency computer network (ARPANET) and the high-capacity transmission facilities for American Telephone and Telegraph (AT&T). Each node and link will have its own delay (corresponding to a weight or cost on that node or link) and its own probability of survival.

### 2.1 Link and Node Weights

When representing computer networks by graphical means, delays at the computers and on the links can be represented as weights on the nodes and links of the graphical model. Throughout this paper, a low weight corresponds to a high bit rate on the links and nodes, and therefore a low delay. There is no reason why high weights could not correspond to high bit rates, but the advantages of using low weights as high bit rates will become evident throughout this paper.

Since GRAFTHY was intended to be user friendly for nontechnical users and flexible enough for users to play "what if" games, the weights input to the program can either correspond to a relative delay on the links and nodes or to an actual bit rate on the links and nodes. For example, if there are two links in the computer network and the second link is twice as slow as the first link, then the first and second link weights could be 1 and 2 respectively. Weights of 2 and 4, 4 and 8, etc. would also work. Therefore the user is not required to know what the actual bit rates are but only relative delay between them.

However, if the bit rates for the network under test are known, then the bit rates must be converted to the proper weights. To convert the bit rates on the nodes and links to weights, a standard should be chosen. Then each bit rate should be divided into the standard to determine their weights. This can be expressed as:

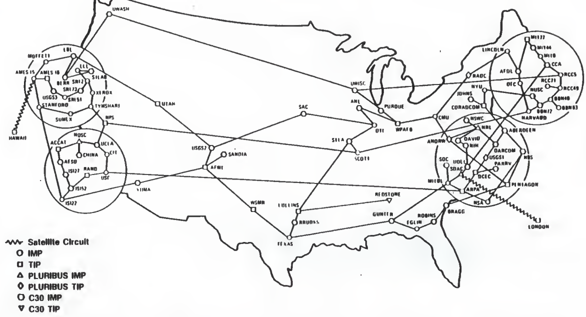

Figure 2.1 The ARPANET geographic map, July 1982. Source: "An Information Briefing on the Defense Data Network (DDN)" by the Defense Communications Agency.

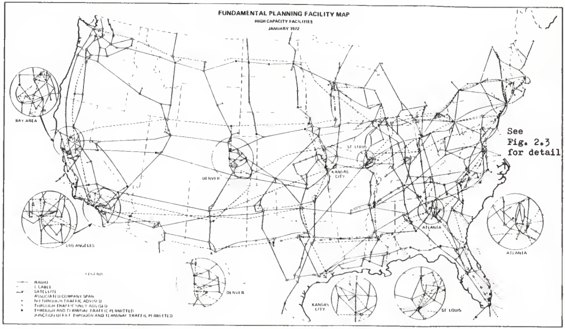
FUNDAMENTAL PLANNING FACILITY MAP
HIGH CAPACITY FACILITIES
JANUARY 1972
BAY AREA
LOS ANGELES
DENVER
KANSAS CITY
ST LOUIS
ATLANTA
See Fig. 2.3 for detail
LEGEND
RADIO
L CABLE
SATELLITE
ASSOCIATED COMPANY SPAN
NO THROUGH TRAFFIC ADVISED
THROUGH TRAFFIC ONLY ADVISED
THROUGH AND TERMINAL TRAFFIC PERMITTED
JUNCTION OFFICE THROUGH AND TERMINAL TRAFFIC PERMITTED

Figure 2.2 High-capacity transmission facilities in the United States. Source: Bell Laboratories; Engineering Operations in the Bell System; Western Electric Co., Inc.; Indianapolis, Ind.; c. 1977.

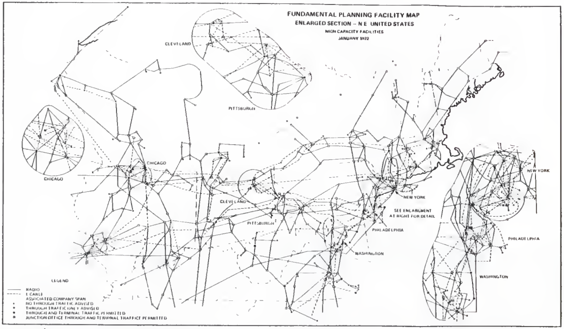
FUNDAMENTAL PLANNING FACILITY MAP
ENLARGED SECTION – N E UNITED STATES
HIGH CAPACITY FACILITIES
JANUARY 1972
CLEVELAND
PITTSBURGH
CHICAGO
CHICAGO
CLEVELAND
PITTSBURGH
NEW YORK
SEE ENLARGMENT AT RIGHT FOR DETAIL
PHILADELPHIA
WASHINGTON
NEW YORK
PHILADELPHIA
WASHINGTON
LEGEND
RADIO
L CABLE
ASSOCIATED COMPANY SPAN
NO THROUGH TRAFFIC ADVISED
THROUGH TRAFFIC ONLY ADVISED
THROUGH AND TERMINAL TRAFFIC PERMITTED
JUNCTION OFFICE THROUGH AND TERMINAL TRAFFIC PERMITTED

Figure 2.3 High-capacity transmission facilities in the United States. Source: Bell Laboratories; Engineering Operations in the Bell System; Western Electric Co., Inc.; Indianapolis, Ind.; c. 1977.

$$\text{WEIGHT} = \frac{R}{\text{STD}} \qquad (2.1)$$

where

| | |
|---|---|
| WEIGHT | is the weight of the node or link in the model, |
| R | is the bit rate of the node or link in the network, |
| STD | is the standard bit rate chosen. |

For example, if there are three links in the computer network which have bit rates of 56 Kbits per second (bps), 19.2 Kbps, and 9600 bps respectively, and the standard is 56 Kbps, then the link weights are 1, 2.917, and 5.714 respectively.

All outputs from GRAFTHY will be weights corresponding to the delay on the nodes or links. If the actual bit rates for the nodes and links were unknown, and only the relative delays were known, then the output weights are weights relative to the input weights. If the bit rates were known for the network under test, then the output weights can be converted directly to bit rates for the network.

All weights on nodes and links will be greater than or equal to zero. In GRAFTHY, a zero weight does not correspond to an infinite bit rate (no delay), but to the link or node being inoperative.

## 2.2 Link and Node Costs

A weight that can be either positive or negative is called a cost. The negative costs correspond to nodes and links that should be traversed when crossing the network from point A to point B. High positive cost links and nodes should be avoided when traversing the network.

There are algorithms available which analyze networks with links and node costs, but these algorithms require more storage and longer run times than those which analyze networks with weights. Since run time is a primary consideration when calculating the MOE's, it is for the reason stated above that negative weights will not be used. This will not cause any loss of generality when describing the operation of the network and calculating the corresponding MOE's.

Generally, when describing graphical networks, the nodes have no weight. Only the links have weight. One method of describing a graphical network with nodes that have weight is to replace the nodes being modeled with supernodes consisting of an input node, an output node, and a directed link going from the input node to the output node. See Figure 2.4. However, this is not a desirable technique to use because of computer networks always have delay associated with them.

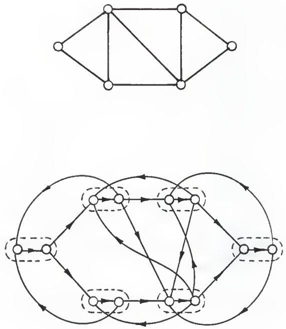

Figure 2.4 A graphical supernode model (below) of a network with node weights (shown above).

If computer networks were modeled in the fashion described above, then the number of nodes in the graphical model describing the computer network would be twice the number of nodes in the computer network being modeled. The number of directed links in the graphical model would be twice the number of bidirectional links in the computer network being modeled, plus the number of directed links in the network being modeled, plus the number of nodes in the computer network (since each computer is now modeled by two nodes and a directed link). In other words,

$$\mathrm{MNON} = 2 * \mathrm{NON} \tag{2.2}$$

where

MNON is the number of nodes in the model,
NON is the number of nodes in the network,

and

$$\mathrm{MNODL} = 2 * \mathrm{NOBL} + \mathrm{NODL} + \mathrm{NON} \tag{2.3}$$

where

| | |
|---|---|
| MNODL | is the number of directed links in the model, |
| NOBL | is the number of bidirectional links in the network, |
| NODL | is the number of directed links in the network, |
| NON | is the number of nodes in the network. |

Since the node modeling technique described above is undesirable, it was not used. Instead, all the algorithms used in GRAFTHY were modified to accept node weights and calculate the MOE's without using the supernode technique. An examination of the programs in the Appendices will reveal the techniques used in the modification.

The reason for minimizing the number of nodes and links in the model is to save run time. A six-node communication network modeled with a 12-node graphical model would be more than four times slower than a six-node graphical model.

## 2.3 Link and Node Probabilities of Survival

When representing computer networks by graphical means, the node and link probabilities of survival of that node or link in the model correspond to either the probability of survival of that node or link in the network, or to the probability that a message in the network will traverse that node or link without experiencing a bit in error.

Since either option is available, GRAFTHY, a network-oriented program, could easily be made message-oriented.

Since link and node probabilities of exactly one are unrealistic, erroneous results will occur if probabilities of one are input to GRAFTHY.

## 2.4 Graphical Degradation

When dealing with the degradation of a computer network, many degradation techniques can be used dependent upon how the computer network is modeled. Other simulation models are binary for degradation purposes. Links and nodes are either up or down (the link or node probability is either one or zero respectively).

With GRAFTHY, the user has a high-resolution model. Link and node probabilities of survival can range anywhere between zero and one, instead of being only the values of zero and one as in the other simulations. GRAFTHY also allows elements of each node to fail while other elements in the same node remain active. For example, a node's receiver may fail while its transmitter could still be operative, but with a lower probability of successful transmission. Table 2.1 demonstrates how to transform network degradation into graphical degradation for specific events.

| Communication network | Graphical model |
|---|---|
| 1. If the directed link from A to B down, then | 1. Remove link from A to B. |
| 2. If the transmitter from A to B at node A down, then | 2. Remove link from A to B. |
| 3. If the receiver from A to B at node B is down, then | 3. Remove link from A to B. |
| 4. If the bidirectional link between points A to B goes down, then | 4. Remove both directed links, A to B and B to A. |
| 5. If the node has only one transmitter for all links and the transmitter fails, then | 5. Remove all outgoing links from node A. |
| 6. If the receiver at node A receives all incoming signals and the receiver fails, then | 6. Remove all incoming links from node A. |
| 7. If node A goes down completely, then | 7. Remove all incoming and outgoing links from node A. |

Table 2.1 Changes needed in the graphical model to simulate network degradation for specific events.

## 2.5 Matrix Representation of Graphs

There are many matrix representations of bidirectional, directed and mixed graphs (3). They are the Adjacency Matrix, the Connection Matrix, the Reachability Matrix, the Incidence Matrix, the Circuit Matrix, and the Cut-Set Matrix.

### 2.5.1 Adjacency Matrix

The Adjacency Matrix is a square matrix of size NON x NON, where NON is the number of nodes in the network, whose elements are given as follows:

$a(i, j) = 1$ if there is a branch from node i to node j,

$a(i, j) = 0$ if there is no branch from node i to node j.

The Adjacency Matrix works for either directed or bidirectional graphs.

### 2.5.2 Connection Matrix

The Connection Matrix is a matrix of size NON x NON where NON is the number of nodes in the graph, whose elements are given as follows:

$a(i, j) =$ weight from node i to node j if a link exists,

$a(i, j) = 0$ if no link exists between node i and node j.

The Connection Matrix is good for bidirectional, directed, or mixed graphs.

### 2.5.3 Reachability Matrix

The Reachability Matrix is a matrix of size NON x NON, where NON is the number of nodes in the network, whose elements are given by:

$a(i, j) = 1$ if there is at least one path from node i to node j,

$a(i, j) = 0$ if there is no path from node i to node j.

The Reachability Matrix is good for bidirectional, directed, or mixed graphs.

### 2.5.4 Incidence Matrix

The Incidence Matrix is a matrix of size NON x NOL, where NON is the number of nodes of the network and NOL is the number of links in the network, whose elements for bidirectional graphs are given as follows:

a(i, j) = 1 if link j is connected to node i,

a(i, j) = 0 if link j is not connected to node i.

For directed link graphs,

a(i, j) = +1 if link j is directed out of node i,

a(i, j) = -1 if link j is directed into node i,

a(i, j) = 0 if link j is not connected to node i.

Every column of the Incidence Matrix will contain two entries. Each link must begin and end at some node, and since by definition a link can contain no other node, this is a quick check to ensure the Incidence Matrix is correct.

### 2.5.5 Circuit Matrix

The Circuit Matrix is a matrix of size NOC x NOL, where NOC is the number of circuits in the network and NOL is the number of links in the network, whose elements are given as follows:

a(i, j) = 1 if link j is in circuit i,

a(i, j) = 0 if link j is not in circuit i.

The Circuit Matrix works only for bidirectional graphs.

Every column of the Circuit Matrix will add to the same number. This is a quick check to see if all the loops have been identified, but the check is not absolute. There are special cases where the test could fail.

Also, the product of the Incidence Matrix and the transpose of the Circuit Matrix equals zero in field modulo-2 algebra. This can be expressed as

$$IC^T = 0 \tag{2.4}$$

where

I is the Incidence Matrix,
D is the Circuit Matrix.

Field modulo-2 algebra is simply Boolean algebra using the exclusive-OR operator. The product above determines whether the identified loops are valid ones. It will not determine whether all loops have been identified.

### 2.5.6 Cut-Set Matrix

The Cut-Set Matrix is a matrix of size NOCS x NOL, where NOCS is the number of cut-sets of the graph and NOL is the number of links in the graph, whose elements are given as follows:

$a(i, j) = 1$ if link j is in cut-set i,

$a(i, j) = 0$ if link j is not in cut-set i.

The Cut-Set Matrix is good for bidirectional graphs only.

The product of the Circuit Matrix and the transpose of the Cut-Set Matrix equals zero in field modulo-2 algebra. In other words,

$$CS^T = o \tag{2.5}$$

where

C is the Circuit Matrix,
S is the Cut-Set Matrix.

The product determines whether the identified cut-sets are valid ones.

## 3.0 Measures of Effectiveness

MOE's determine how effective the computer network is depending on the criterion used. Criterion range from shortest delay to highest reliability. MOE's can be divided into two groups: those which require a given source and destination node and those which only have meaning when the network is viewed as a whole.

## 3.1 Source to Destination Measures of Effectiveness

A list of MOE's which require a given source and destination node and a description of how they are obtained follows.

### 3.1.1 Shortest Delay Path Measure of Effectiveness

The Shortest Delay Path MOE gives the shortest delay path between a given source and destination node taking into account processing delay at each of the nodes along the path, including the source and destination nodes. Dijkstra's Shortest Path First Algorithm is recommended as the best method of determining the shortest delay path between a given source and destination (2,4,5).

#### 3.1.1.1 Dijkstra's Shortest Path First Algorithm

Dijkstra's Shortest Path First Algorithm gives each node a temporary label corresponding to the weight from the source to the destination. This label, initially infinity, is an upper bound for the total weight from the source to the destination node.

During program execution, nodes fall in two classes, either permanent or temporary. A permanent label on a node means there is no shorter path weight from the source to that destination. A temporary label means no shortest path to that destination has been found yet.

After each iteration in the program, exactly one node is made permanent with its corresponding weight the shortest path weight from source to that destination. Details of the algorithm are given below.

##### 3.1.1.1.1 Initialization

Step 1: a) Set the weight from the source node to itself equal to the weight at that node.

b) Set the weight from the source to all other nodes to infinity.

c) Make the label of the source node permanent. In other words, there is no shorter path.

d) Make the label of all other nodes temporary.

e) Set the intermediate source node equal to the source node.

#### 3.1.1.1.2 Label Updating

Step 2: For all temporary labeled adjacent nodes to the intermediate source node, change each adjacent node weight to either its present value or to the accumulated weight from the intermediate source node to the adjacent node, choosing whichever is a minimum.

#### 3.1.1.1.3 Making a Label Permanent

Step 3: For all the temporary labeled nodes, find the node whose weight is a minimum.

Step 4: Make that node the new intermediate source node, and make its label permanent.

Step 5: a) If only a path from source to destination is required, and if the intermediate source node is the destination node, then stop. Otherwise, go to Step 2.

b) If a path from the source to every other node is required, and if all the labels are permanent, stop. All the paths are shortest paths. If there are temporary labels present, go to Step 2.

### 3.1.1.2 Two Language Implementations of the Shortest Path First Algorithm

The program listings in Appendices A and B are quite well-commented, complete with a listing of the variables used, and need no further explanation. A user's manual is provided in Appendix C..

### 3.1.1.3 Shortest Path First Algorithm Output

The output from Dijkstra's Shortest Path First Algorithm will be a weight corresponding to the delay between the given source and destination. To convert this weight to a delay in seconds, the message length should be divided by the standard and multiplied by the shortest delay path weight. This can be expressed as

$$D = \frac{ML}{STD} \, SDPW \qquad (3.1)$$

where

| | |
|---|---|
| D | is the delay from source to destination in seconds, |
| ML | is the message length in bits, |
| STD | is the standard bit rate chosen in bps, |
| SDPW | is the shortest delay path weight. |

For example, if weight 1 (the standard) corresponds to a bit rate of 56 Kbps, the message length is 1 K bits, and the shortest delay path weight is 7, then the delay from the source to destination is 0.125 seconds.

### 3.1.2 Highest Reliable Path Measure of Effectiveness

The Highest Reliable Path MOE gives the highest reliable path from source to destination, taking into account the probability of node survival at each node along the path, including the source and destination node.

#### 3.1.2.1 Highest Reliable Path Algorithm

To calculate the Highest Reliable Path, Dijkstra's Shortest Path First Algorithm is used. Previous to executing the Shortest Path First Algorithm, the negative logarithm of the probability matrix is found, and the negative logarithm probability matrix is used in place of the connection matrix. The Shortest Path First Algorithm will then add the minimum logarithms, which is the same as multiplying the highest probabilities along the path.

#### 3.1.2.2 Two Language Implementations of the Highest Reliable Path Algorithm

The program listings in Appendices A and B show the preprocessor and the call to Dijkstra's Shortest Path First Algorithm. The preprocessor takes the negative logarithm of the Probability of Survival Matrix and passes the negative logarithm probability matrix to Dijkstra's Shortest Path First Algorithm in place of the connection matrix. A user's manual is provided in Appendix C.

### 3.1.3 Reachability and Limited Reachability Measures of Effectiveness

The Reachability MOE tells whether a given destination node can be reached from a given source node in any number of links along the path. The Limited Reachability MOE tells whether the destination node can be reached in a given number of links or less from the source node.

#### 3.1.3.1 Reachability and Limited Reachability Algorithm

To calculate the Reachability, Dijkstra's Shortest Path First Algorithm is applied here also. Before executing the algorithm, all weights on the links are set to one, and the node weights are set to zero. The Shortest Path First Algorithm will then add the minimum weights along the path. Since all weights are equal to one, the addition just totals the number of links along the path.

The Limited Reachability is found by comparing the number of links along the path to the threshold number. If the number of links is less than or equal to the threshold number, then the corresponding element of the Reachability Matrix is one. If the number of links is greater than the threshold, then the element is zero.

#### 3.1.3.2 Two Language Implementations of the Reachability and Limited Reachability Algorithm

The program listings in Appendices A and B show the preprocessor and the call to Dijkstra's Shortest Path First Algorithm. The preprocessor sets all weights on the links to one and sets the node weights to zero. A user's manual is provided in Appendix C.

### 3.1.4 Maximum Throughput Measure of Effectiveness

The Maximum Throughput MOE gives the maximum throughput between a given source and destination node taking into account the processing delay at each of the nodes, including the source and the destination nodes. Ford and Fulkerson's Min-Cut Max-Flow Algorithm is recommended as the best method of determining the maximum throughput between a given source and destination (4,5).

### 3.1.4.1 Ford and Fulkerson's Minimum-Cut Maximum-Flow Algorithm

Ford and Fulkerson's Min-Cut Algorithm gives each node a temporary label corresponding to flow leaving all nodes. This label, initially infinity, is an upper bound for the flow across the minimum cut.

During program execution, a label on the node can be either labeled and scanned, labeled and unscanned, or unlabeled and unscanned. If a node is labeled and scanned, then all of the adjacent links to the node have been examined. If a node is labeled and unscanned, then there are still adjacent links to be examined. Finally, if a node is unlabeled and unscanned, then none of the adjacent links have been examined.

Before executing Ford and Fulkerson's Min-Cut Max-Flow Algorithm, however, all weights on the nodes and links are inverted by a preprocessor. This is necessary since the program is searching for the minimum cut corresponding to low throughput. Since low weights in the GRAFTHY model correspond to high bit rates, the weights must be inverted so the program will find the low bit rate links.

The algorithm then pushes flow along a path from source to destination, saturating the link or node with the lowest weight along the path. This process continues until no more flow can proceed to the destination node. The saturated nodes and links form the mimimum cut. Details of the algorithm are given below.

#### 3.1.4.1.1 Initialization

Step 1: a) Label the source node labeled and unscanned.

b) Label all other nodes unlabeled and unscanned.

#### 3.1.4.1.2 Label Updating

Step 2: Choose any labeled unscanned node (intermediate source node), and for all adjacent unlabeled unscanned nodes (intermediate destination nodes),

a) If the flow from the intermediate destination node to the intermediate source node is greater than zero, then label the intermediate destination node as labeled and unscanned. Also, change the flow exiting the intermediate destination node to either the flow already present on that link or to the flow being pulled along that link, choosing whichever is a minimum. Label the intermediate source node as labeled and scanned, and

b) If the flow being pushed along the link between the intermediate source node and the intermediate destination node is less than the capacity of the link, then label the intermediate destination node as labeled and unscanned. Also change the flow entering the intermediate destination node to either the flow being pushed along that link or to the capacity of the link minus the flow already present on the link, choosing whichever is a minimum. Label the intermediate source node as labeled and unscanned.

Step 3: Repeat Step 2 until either:

a) The destination node is labeled. In this case, go to Step 4, or

b) The destination node is unlabeled and no more labels can be placed. For this case, the algorithm terminates with the interface between the set of labeled links and the set of unlabeled links the minimum cut.

### 3.1.4.1.3 Flow Augmentation

Step 4: Let a pointer (intermediate destination node) be the destination node.

Step 5: a) If there is flow directed into the intermediate destination node, then increase the flow along the link between the previous intermediate source node and the intermediate destination node by the minimum flow along the path being examined.

b) If there is flow directed out of the intermediate destination node, then decrease the flow along the link between the previous intermediate source node and the intermediate destination node by the minimum flow along the path being examined.

Step 6: a) If the pointer node is the source node, then go to Step 1, repeating the process with the improved flow calculated in Step 5.

b) If the pointer node is not the source node, then make the previous intermediate source node the intermediate destination node and go to Step 5.

### 3.1.4.2 Two Language Implementations of the Minimum-Cut Maximum-Flow Algorithm

The program listings in Appendices A and B are quite well-commented, complete with a listing of the variables used, and need no further explanation. A user's manual is provided in Appendix C.

### 3.1.4.3 Minimum-Cut Maximum-Flow Output

The output from Ford and Fulkerson's Min-Cut Max-Flow Algorithm will be a weight corresponding to the throughput between the given source and destination. To convert this weight to a throughput in bits per second, the standard should be multiplied by a maximum-flow weight. In other words,

$$T = STD * MFW \qquad (3.2)$$

where

T is the maximum throughput between a given source and destination in bps,
STD is the standard chosen in bps,
MFW is the maximum-flow weight.

For example, if weight 1 (the standard) corresponds to a bit rate of 9600 bps, and the maximum flow weight is 1.20, then the maximum throughput from source to destination is 11.52 Kbps.

### 3.1.5 Number of Link Independent Paths Measure of Effectiveness

The Number of Link Independent Paths MOE, also known as Arc Connectivity, k-Arc Connectivity, and Degree, gives the number of link independent paths between the source and destination. Furthermore, this MOE shows that any k-1 links can be removed without disconnecting the network.

### 3.1.5.1 Number of Link Independent Paths Algorithm

To calculate the number of link independent paths, Ford and Fulkerson's Min-Cut Max-Flow Algorithm is used. Before executing the Min-Cut Max-Flow Algorithm, all weights on the links and nodes are set to one. Therefore, the minimum cut totals the number of links in the minimum cut separating source and destination.

### 3.1.5.2 Two Language Implementations of the Number of Link Independent Paths Algorithm

The program listings in Appendices A and B show the preprocessor and the call to Ford and Fulkerson's Min-Cut Max-Flow Algorithm. The preprocessor sets all weights on the links and nodes to one. A user's manual is provided in Appendix C.

### 3.1.6 Number of Node Independent Paths Measure of Effectiveness

The Number of Node Independent Paths MOE, also known as Node Connectivity and k-Node Connectivity, gives the number of node independent paths between the source and destination. Also, this MOE shows that k-1 nodes, excluding the given source and destination, can be removed without disconnecting the source and destination nodes. This MOE was not calculated and is only mentioned here for the sake of completeness.

### 3.1.7 Reliability Measure of Effectiveness

The source to destination Reliability MOE gives a probability that at least one path exists between the source and destination. There are three ways of finding the source to destination reliability. They are path enumeration, cut-set enumeration, and state enumeration. GRAFTHY uses state enumeration since it was easiest to implement.

#### 3.1.7.1 Reliability Using Path Enumeration

The basic algorithm for finding the source to destination reliability using path enumeration is as follows:

Step 1: Find all successful paths between the source and destination.

Step 2: Find all the required intersections of the paths (the union of the elements).

Step 3: Replace each element with its probability of success and carry out the multiplication.

Step 4: Sum all the reliability expressions according to set theory.

### 3.1.7.2 Reliability Using Cut-Set Enumeration

The basic algorithm for finding the source to destination reliability using cut-set enumeration is as follows:

Step 1: Find all cut-sets which separate the source and destination nodes.

Step 2: Find all the required intersections of the cut-sets (the union of the elements).

Step 3: Replace each element with its probability of failure and carry out the multiplication.

Step 4: Sum all the reliability expressions according to set theory.

Step 5: One minus the sum calculated above gives the probability that at least one path exists between the source and destination.

### 3.1.7.3 Reliability Using State Enumeration

The basic algorithm for finding the source to destination reliability using state enumeration is as follows:

Step 1: Find all possible combinations of the states.

Step 2: For each combination which gives a success, find the product of the probabilities of failures of the down elements and the probabilities of successes of the up elements.

Step 3: Sum all the terms above.

### 3.1.7.4 Reliability Run Time

Since calculating the exact reliability is a lengthy process for large networks, regardless of the technique used, two independent MOE's were developed to give quickly calculated approximations of network behavior using the probabilities of link and node survival of the network. These MOE's are Reliable Throughput and Network Reliability. Since these MOE's are network MOE's, and not source to destination MOE's, they will be discussed later in the paper.

#### 3.1.7.5 Two Language Implementations of the Reliability Algorithm Using State Enumeration

The program listings in Appendices A and B are quite well-commented, complete with a listing of the variables used, and need no further explanation. A user's manual is provided in Appendix C.

### 3.1.8 Availability Measure of Effectiveness

The Availability MOE uses the Reliability MOE to find such parameters as Mean Time Between Failures of links and nodes. This MOE was not calculated because reliability as a function of time is not known in a wartime network. Since no one is certain of exactly how degradation is going to occur on the network, Mean Time Between Failures has no meaning. Therefore, the time is better spent calculating other MOE's.

## 3.2 Network Measures of Effectiveness

Many network MOE's can be found by finding the average of the source to destination MOE's. For example, Average Delay, Average Throughput, and Average Reliability are all average of source to destination MOE's. Since finding the average of a MOE is a trivial matter and since the average of a MOE does not change the basic definition of the MOE (average delay is still delay, average throughput is still throughput), these types of MOE's will not be discussed further.

However, some network MOE's cannot be determined by taking the average of source to destination MOE's because no corresponding MOE exists on the source to destination level. These type of network MOE's, and a description of how they are obtained, follows.

### 3.2.1 Connectivity Measure of Effectiveness

The Connectivity MOE gives the connectivity of the network, where connectivity is defined as the number of communicating node pairs after attack, divided by the number of originally communicating node pairs (6). For bidirectional networks, this definition poses no problem. The pair 1,2 is the same as the pair 2,1. However, in a directed or mixed network, order does make a difference. Therefore, to take care of this problem, the derivation of connectivity and the modifications necessary to expand the definition to include directed and mixed graphs are shown below.

## 3.2.1.1 Connectivity Derivation

Assume the number of nodes in a maximally connected newtowk is NON. Since in a computer network a self-loop is meaningless (a node can always talk to itself), the node pairs are given by: 1,2; 1,3; ...; 2,1; 2,3;...; NON-1,NON. Thus there are NON different ways to choose the first node and NON-1 different ways to choose the second node. Therefore, the number of node pairs (NONP) is given by

$$\text{NONP} = \text{NON}\ (\text{NON}-1) \tag{3.3}$$

For a bidirectional network there are duplications in the pairs. In other words, the pair 1,2 is the same as the pair 2,1. Therefore, the number of node pairs for a bidirectional network (NONPB) is given by

$$\text{NONPB} = \frac{\text{NON}\ (\text{NON}-1)}{2} \tag{3.4}$$

For directed or mixed networks, the order does make a difference and the number of node pairs for a directed or mixed network (NONPD) is given by

$$\text{NONPD} = \text{NON}\ (\text{NON}-1) \tag{3.5}$$

It would seem that the number of node pairs for a bidirectional network is half as great as the number of node pairs for a directed or mixed graph. But since connectivity is a ratio of terms, it will be shown that the two in the denominator of the bidirectional term drops out.

After degradation, the network may consist of k disjoint networks (in which some of the disjoint networks may consist of only one node). For a bidirectional disjoint network after degradation, the number of connected node pairs is given by the sum of the connected node pairs from all the disjoint networks divided by two.

$$\text{NONPB} = \frac{\sum_{i=1}^{k} \text{NON}_i^{\prime}\ (\text{NON}_i^{\prime}-1)}{2} \tag{3.6}$$

For a directed or mixed disjoint network after degradation, the number of host pairs is given by the sum of the connected node pairs from all the disjoint networks.

$$\text{NONPD} = \sum_{i=1}^{k} \text{NON}_i \cdot (\text{NON}_i - 1) \qquad (3.7)$$

Remembering that connectivity is defined as the number of connected node pairs after degradation divided by the number of connected node pairs before degradation, connectivity for the bidirectional network (CONB) is

$$\text{CONB} = \frac{\dfrac{\sum_{i=1}^{k} \text{NON}_i (\text{NON}_i - 1)}{\cancel{2}}}{\dfrac{\text{NON}(\text{NON}-1)}{\cancel{2}}} \qquad (3.8)$$

Note the two drops out in both denominators. The connectivity for the directed or mixed graph case (COND) is

$$\text{COND} = \frac{\sum_{i=1}^{k} \text{NON}_i (\text{NON}_i - 1)}{\text{NON}(\text{NON}-1)} \qquad (3.9)$$

Note the two connectivities are equal.

### 3.2.1.2 Connectivity Algorithm

To calculate the connectivity of the network, the number of connected node pairs before degradation and the number of connected node pairs after degradation must be determined, and Dijkstra's Shortest Path First Algorithm can be of use here.

The routing matrix for the network is determined by Dijkstra's Shortest Path First Algorithm. The routing matrix is a form of the reachability matrix where the elements of the matrix contain either the adjacent node to the source node along a path to the destination node, or zero if the destination node cannot be reached from the source. If the element of the routing matrix is non-zero, then the pair of nodes being examined is connected; otherwise the pair is disconnected. The number of connected node pairs is found by totaling the number of non-zero elements in the routing matrix.

After degradation, the routing matrix is again determined for the degraded network, and the number of connected node pairs is calculated. The connectivity for the network is determined by dividing the number of connected node pairs after degradation by the number of connected node pairs before degradation.

#### 3.2.1.2 Two Language Implementations of the Connectivity Algorithm

The program listings in Appendices A and B show the preprocessor and the call to Dijkstra's Shortest Path First Algorithm. The preprocessor sets all weights on the links and nodes to one. A user's manual is provided in Appendix C.

### 3.2.2 Connected Network Reliability Measure of Effectiveness

The Connected Network Reliability MOE gives the probability that every connected node pair before degradation is still connected after degradation. This MOE is calculated at the same time as the source to destination Reliability MOE using state enumeration.

### 3.2.3 Reliable Throughput Measure of Effectiveness

The Reliable Throughput MOE gives the reliable throughput of the network, where reliable throughput is defined as the sum of the link and node probabilities after degradation divided by the original sum of the link and node probabilities.

#### 3.2.3.1 Reliable Throughput Algorithm

To calculate the reliable throughput of the network, the sum of the link and node probabilities before degradation and the sum of the link and node probabilities after degradation must be determined. Once they are determined, the after degradation sum divided by the before degradation sum gives the reliable throughput of the network.

#### 3.2.3.2 Two Language Implementations of the Reliable Throughput Algorithms

The program listings in the Appendices A and B are quite well-commented, complete with a listing of variables, and need no further explanation. A user's manual is provided in Appendix C.

## 3.2.4 Network Reliability Measure of Effectiveness

The Network Reliability MOE gives the network reliability of the network, where network reliability is defined as the product of the connectivity of the network and the reliable throughput of the network. The Network Reliability MOE should not be confused with the average of the source and destination Reliability MOE discussed earlier. They are not the same.

### 3.2.4.1 Network Reliability Algorithm

To calculate the network reliability, both the connectivity and the reliable throughput of the network must be known. The technique for determining these terms is described above. Once the terms are known, their product gives the network reliability.

### 3.2.4.2 Two Language Implementations of the Network Reliability Algorithm

The program listings in Appendices A and B show the preprocessor and the calls to both the Connectivity Algorithm and the Reliable Throughput Algorithm. The preprocessor sets all weights on the links and nodes to one. A user's manual is provided in Appendix C.

## 4.0 Measures of Survivability

A network may be considered survivable if:

a) any node can communicate with any other node;

b) communication paths exist between specified pairs of nodes;

c) the largest communicating section after attack exceeds a specified threshold;

d) the shortest surviving path after degradation does not exceed a specified limit;

e) the average fraction of specified pairs of nodes communicating after attack exceeds some limit;

f) the average and/or maximum time needed to transmit a message from source to destination does not exceed a specified time limit;

g) the expected percentage of surviving nodes receiving a message during a given time interval exceeds some level;

h) the probability of a specific node receiving a message during a given time interval exceeds some time limit (5,7,8,9).

Many of these MOS's can be found using the MOE's previously discussed.

## 5.0 Testing and Validation

Several different types of network configurations were input to GRAFTHY to test and validate its output. They were a six-node maximally connected network, a six-node ring network, a six-node line network, a six-node star network, and the six-node general network shown in Figure 2.4. In all the above cases, correct outputs from GRAFTHY were obtained. An example run for a three-node network is shown in Appendix D.

## 6.0 Recommendations for Further Research

GRAFTHY is a powerful, user-friendly simulation which is network-oriented. The inputs reflect the network topology, and the output describes how well the network functions after degradation. In message-oriented simulations, the inputs reflect the protocols at the node, and the outputs describe whether messages were received or not. These simulations do not tell the user how "good" the networks are. However, GRAFTHY, by describing the network, does infer whether messages can be received or not. Because network-oriented simulations describe the network effectiveness and also infer whether messages are received, and message-oriented simulations only describe whether messages are received, a comparison should be made between the two types of simulations to determine fully the number of advantages of network-oriented simulations such as GRAFTHY.

## 7.0 Summary and Conclusions

The military presently utilizes computers and computer networks in all facets of their C3I systems. As a consequence of this use, the C3I system operational effectiveness is directly related to the degradation level of these computer networks. Thus there is an intense interest on the part of military commanders to have an understanding as to how strategic and tactical environments degrade the operational capability of computer networks within their C3I systems. This interest can only be satisfied with considerable analysis of computer networks within various hostile environments. Further, this analysis usually requires a simulation model as one of the analytic tools.

GRAFTHY was written to meet the need for a simulation model which calculates the degradation level (effectiveness/survivability) of these computer networks. Since GRAFTHY calculates many MOE's, each MOE representing a different criterion for the level of degradation of the network, the user can determine which criteria affect the network most. Thus commanders gain a better understanding of the degree to which different measures affect the operational capability of their C3I systems.

## 8.0 Literature Cited

1. A brochure on the Defense Data Network; The Defense Communications Agency

2. Tanenbaum, Andrew S.; Computer Networks; Prentice-Hall, Inc.; Englewood Cliffs, N.J.; c. 1981

3. Henley, Ernest J. and R.A. Williams; Graph Theory in Modern Engineering; Academic Press Inc.; New York, N.Y.; c. 1973

4. Christofides, Nicos; Graph Theory: An Algorithmic Approach; Academic Press Inc.; New York, N.Y.; c. 1975

5. Frank, Howard and Ivan T. Frisch; Communication, Transmission, and Transportation Networks; Addison-Wesley Publishing Co., Inc.; Reading, Mass.; c. 1971

6. Sevcik, Peter J., Graeme J. Williams, and Bruce L. Hitson; "Defense Data Network Survivability;" Bolt Beranek and Newman, Inc. for the Defense Communications Agency; IEEE Press; New York, N.Y.; c. 1982

7. Frank, Howard; "Survivability Analysis of Command and Control Communications Networks - Part I," IEEE Transactions on Communications, Vol. COM-22, No. 5; May 1974

8. Frank, Howard; "Survivability Analysis of Command and Control Communications Networks - Part II," IEEE Transactions on Communications, Vol. COM-22, No. 5; May 1974

9. Frank, Howard and Ivan T. Frisch; "Analysis and Design of Survivable Networks," IEEE Transactions on Communications, Vol. COM-18, No. 5; Oct. 1970

## 9.0 Further Reading on Graph Theory

Abraham, J.A.; "An Improved Algorithm for Network Reliability," IEEE Transactions on Reliability, Vol. R-28, No. 1; April 1979

Aggarwal, K.K., K.B. Misra, and J.S. Gupta; "A Simple Method for Reliability Evaluation of a Communication System," IEEE Transactions on Communications; May 1975

Aggarwal, K.K., K.B. Misra, and J.S. Gupta; "A Fast Algorithm for Reliability Evaluation," IEEE Transactions on Reliability, Vol. R-24, No. 1; April 1975

Arunkumar, S. and S.H. Lee; "Enumeration of All Minimal Cut-Sets for a Node Pair in a Graph," IEEE Transactions, Vol. R-28, No. 1; April 1979

Ball, Michael O,; "Complexity of Network Reliability Computations," Networks, Vol. 10; 1980

Bellman, Richard, Kenneth L. Cooke, and Jo Ann Lockett; Algorithms, Graphs, and Computers; Academic Press Inc.; New York, N.Y.; c. 1970

Biegal, John E.; "Determination of Tie Sets and Sets for a System Without Feedback," IEEE Transactions on Reliability, Vol R-26, No. 1; April 1977

Colella, A.M., M.J. O'Sullivan, and D.J. Carlino; Systems Simulation; D.C. Heath and Co.; Lexington, Mass.; c. 1974

Even, Shimon and R. Endre Tarjan; "Network Flow and Testing Graph Connectivity," SIAM Journal on Computing, Vol. 4, No. 4; Dec. 1975

Gomory, R.E. and T.C. Hu; "Multi-Terminal Network Flows," SIAM Journal on Applied Mathematics, Vol. 9, No. 4; Dec. 1961

Haray, Frank and Edgar M. Palmer; Graphical Enumeration; Academic Press Inc.; New York, N.Y.; c. 1973

Hu, T.C.; Integer Programming and Network Flows; Addison-Wesley Publishing Co., Inc.; Reading, Mass.; c. 1969

Kershenbaum, A. and R.M. Van Slyke; "Recursive Analysis of Network Reliability," Networks, Vol. 3; 1973

Khan, N.M., K. Rajamani, and S.K. Banerjee; "A Direct Method to Calculate the Frequency and Duration of Failures for Large Networks," IEEE Transactions on Reliability, Vol. R-26, No. 5; Dec. 1977

Kim, Young H., Kenneth E. Case, and P.M. Ghare; "A Method for Computing Complex System Reliability," IEEE Transactions on Reliability, Vol. R-21, No. 4; Nov. 1972

Locks, Mitchell O.; "Relationships Between Minimal Path Sets and Cut Sets," IEEE Transactions on Reliability, Vol. R-27, No. 2; June 1978

Locks, Mitchell O.; "Inverting and Minimalizing Path Sets and Cut Sets," IEEE Transactions on Reliability, Vol. R-27, No. 2; June 1978

Mihram, G. Arthur; Simulation: Statistical Foundations and Methodology; Academic Press Inc.; New York, N.Y.; c. 1972

Minieka, Edward; Optimization Algorithms for Networks and Graphs; Marcel Dekker, Inc.; New York, N.Y.; c. 1978

Moranda, Paul B.; "Event-Altered Rate Models for General Reliability Analysis," IEEE Transactions on Reliability, Vol. R-28, No. 5; Dec. 1979

Petrovic, Radivoj and Slobodan Jovanovic; "Two Algorithms for Determining the Most Reliable Path of a Network," IEEE Transactions on Reliability, Vol. R-28, No. 2; June 1979

Rai, Suresh and K.K. Aggarwal; "An Efficient Method for Reliabilities Evaluation of a General Network," IEEE Transactions on Reliability, Vol. R-27, No. 3; Aug. 1978

Read, Ronald C.; Graph Theory and Computing; Academic Press Inc.; New York, N.Y.; c. 1972

Satyanarayana, A. and Jane N. Hagstrom; "Combinatorial Properties of Directed Graphs Useful in Computing Network Reliability," Networks, Vol. 11; 1981

Schnorr, C.P.; "Bottlenecks and Edge Connectivity in Unsymmetrical Networks," SIAM Journal on Computing; Vol. 8, No. 2; May 1979

Sharma, Roshan Lal, Paulo J.T. de Sousa, and Ashok D. Ingle; Network Systems; Van Nostrand Reinhold Co., Inc.; New York, N.Y.; c. 1982

Shoemaker, S. (Ed.); Computer Networks and Simulations II; North-Holland Publishing Co.; Amsterdam, The Netherlands; c. 1982

Shogan, Andrew W.; "A Recursive Algorithm for Bounding Network Reliability," IEEE Transactions on Reliability, Vol. R-26, No. 5; Dec. 1977

Springer, Clifford H., Robert E. Herlihy, Robert T. Mall, and Robert I. Beggs; Probabilistic Models; Richard D. Irwin, Inc.; Homewood, Ill.; c. 1968

Tillman, F.A., C.H. Lie, and Hwang; "Simulation Model of Mission Effectiveness for Military Systems," IEEE Transactions on Reliability, Vol. R-27, No. 3; Aug. 1978

Van Slyke, R. and H. Frank; "Network Reliability Analysis: Part I," Networks, Vol. 1; 1972

Whitehouse, Gary E.; Systems Analysis and Design Using Network Techniques; Prentice-Hall, Inc.; Englewood Cliffs, N.J.; c. 1973

Wilkov, Robert S.; "Analysis and Design of Reliable Computer Networks," IEEE Transactions on Communications, Vol. COM-20; June 1972

## 10.0 Further Reading on Computer Networks

Bell Laboratories; Engineering Operations in the Bell System; Western Electric Co., Inc.; Indianapolis, Ind.; c. 1977

Blanc, Robert P. and Ira W. Cotton (Ed.); Computer Networking; IEEE Press; New York, N.Y.; c. 1976

Cantor, David G. and Mario Gerla; "Optimal Routing in a Packet-Switched Computer Network," IEEE Transactions on Computers, Vol. C-23, No. 10; Oct. 1974

Davies, Donald W. and Derek L.A. Barber; Communication Networks for Computers; John Wiley & Sons, Inc.; London, England; c. 1973

Davies, Donald W., Derek L.A. Barber, W.L. Price, and C.M. Solomonides; Computer Networks and Their Protocols; John Wiley & Sons Ltd.; Chichester, England; c. 1979

Frank, Howard, R.E. Kahn, and L. Kleinrock; "Computer Communications Network Design: Experience with Theory and Practice," Large-Scale Networks: Theory and Design; IEEE Press; New York, N.Y.; c. 1976

Green, Jr., Paul E. and Robert W. Lucky (Ed.); Computer Communications; IEEE Press; New York, N.Y.; c. 1974

Kleinrock, Leonard; Queueing Systems Volume II: Computer Applications; John Wiley & Sons, Inc.; New York, N.Y.; c.. 1976

Martin, James; Telecommunications and the Computer, 2nd ed.; Prentice-Hall, Inc.; Englewood Cliffs, N.J.; c. 1976

McQuillan, John M. and Vinton G. Cerf; Tutorial: A Practical View of Computer Communication Protocols; IEEE Press; New York, N.Y.; c. 1978

McQuillan, John M., Ira Richer, and Eric C. Rosen; "The New Routing Algorithm for the ARPANET," IEEE Transactions on Communications, Vol. COM-28, No. 5; May 1980

Schwartz, Mischa; Computer-Communication Network Design and Analysis; Prentice-Hall, Inc.; Englewood Cliffs, N.J.; c. 1977

## Appendix A: A FORTRAN GRAFTHY

```
      PROGRAM GRAFTHY
C
C Written by:  Russell O. Thomas
C Date:  August 3, 1984
C
C This is the declaration block.
      OIMENSIDN CONNECT(100,100), CUT(1DO,100), CUT2(1DD,100)
      DIMENSIDN HRPM(100,100), IND(100,10D), IND2(1D0,100)
      DIMENSION LENGTH(100,100), LENGTH2(100,100), LENM(1D0,1D0)
      DIMENSIDN PDS(100,100), PRDB(100,100), PROB2(1DD,1DD)
      DIMENSION SPM(100,100), TEMP(100,100), WGT(1DD,1DD)
      DIMENSION WGT2(1DD,1DD)
      INTEGER CON, CONAD, CONBD, HRP, HRPM, LENM, MCMF, NDLIP, NR, RCH
      INTEGER RT, SDP, SDRL, SPM, TDTAL
      REAL ADPOCT, BOPOCT, CONNECT, CUT, CUT2, IND, IND2, LENGTH
      REAL LENGTH2, POS, PROB, PRDB2, RAT1, RAT2, RAT3, RTAD, RTBD
      REAL TEMP, WGT, WGT2
C
C***********************************************************************
C
C Input Block
C
C Choose the MDE's.
      CALL CHDDSE_MOE(CON,HRP,MCMF,NOLIP,NR,RCH,RT,SDP,SDRL)
C
C Input the network configuration.
      CALL DESCRIBE_NETWDRK(NDN,CONNECT,PDS)
C
C***********************************************************************
C
C Before-Degradation Output
C
C Calculate and output the Shortest Delay Path MDE if the MDE was
C requested.
      IF (SDP.EQ.1) CALL OIJKSTRA(NDN,CDNNECT,1,SPM,WGT)
C
C Calculate and output the Highest Reliable Path MDE if the MDE was
C requested.
      IF (HRP.EQ.1) THEN
C
C This loop takes the negative logarithm of the probability matrix.
        DD 2D I=1,NDN
          DO 1D J=1,NDN
C
C If the link or node exists, then take the negative logarithm of its
C probability of survival.  Dtherwise, leave the probability zero.
            IF (PDS(I,J).NE.0) THEN
              TEMP(I,J)=-ALOG(PDS(I,J))
            ELSE
              TEMP(I,J)=D
            ENDIF
```

```
 10        CONTINUE
 20      CONTINUE
C
C  Call Dijkstra's Shortest Path First Algorithm and pass the negative
C  logarithm probability matrix in place of the connection matrix.
         CALL DIJKSTRA(NDN,TEMP,2,HRPM,PROB)
       ENDIF
C
C  Calculate and output the Reachability MOE if the MDE was requested.
        IF (RCH.EQ.1) THEN
C
C  This loop sets all link weights to one and all node weights to zero.
         DO 40 I=1,NON
           DO 3D J=1,NDN
C
C  Set node weights to zero.
             IF (I.EQ.J) THEN
               TEMP(I,J)=0
C
C  If a link exists between the two nodes being examined, then set its
C  weight to one.
             ELSE IF (POS(I,J).NE.0) THEN
               TEMP(I,J)=1
C
C  If no link exists, then leave the weight zero.
             ELSE
               TEMP(I,J)=0
             ENOIF
 30        CDNTINUE
 40      CONTINUE
C
C  Call Dijkstra's Shortest Path First Algorithm and pass the binary
C  connection matrix (whose node weights are zero and link weights are
C  one) in place of the original connection matrix.
         CALL DIJKSTRA(NDN,TEMP,3,LENM,LENGTH)
       ENDIF
C
C  Calculate and output the Maximum Throughput MDE if the MDE was
C  requested.
       IF (MCMF.EQ.1) THEN
C
C  This loop inverts all the node and link weights.
        DD 60 I=1,NON
          DO 5D J=1,NDN
C
C  If the link or node exists, then invert its weight.  Dtherwise,
C  leave the weight zero.
             IF (CDNNECT(I,J).NE.D) THEN
               TEMP(I,J)=1/CDNNECT(I,J)
             ELSE
               TEMP(I,J)=0
             ENDIF
 50        CONTINUE
 60      CONTINUE
C
C  Call Ford and Fulkerson's Min-Cut Max-Flow Algorithm and pass the
C  inverted connection matrix in place of the original connection
C  matrix.
         CALL FORD_FULKERSON(NON,TEMP,1,CUT)
       ENDIF
C
C  Calculate and output the Number of Link Independent Paths MOE if the
C  MOE was requested.
       IF (NOLIP.EQ.1) THEN
C
C  The CHANGE subroutine will return a binary connection matrix whose
C  node and link weights are one.
         CALL CHANGE(NON,CONNECT,TEMP)
C
C  Call Ford and Fulkerson's Min-Cut Max-Flow Algorithm and pass the
C  binary connection matrix in place of the original connection
C  matrix.
         CALL FORD_FULKERSON(NON,TEMP,2,INO)
       ENDIF
C
C  Calculate and output the Reliability MOE if the MOE was requested.
       IF (SORL.EQ.1) THEN
C
C  The CHANGE subroutine will return a binary connection matrix whose
C  node and link weights are one.
         CALL CHANGE(NON,CONNECT,TEMP)
C
C  Call the connectivity algorithm to determine the number of connected
C  node pairs.
         CALL CONNECTIVITY(NON,TEMP,TOTAL)
C
C  Call the reliability algorithm and pass the binary connection matrix
C  and the probability connection matrix.
         CALL RELIABILITY(NON,TEMP,POS,TOTAL)
       ENDIF
C
C  Calculate the before-degradation term for the Connectivity MOE if
C  the MOE was requested.
       IF (CON.EQ.1) THEN
C
C  The CHANGE subroutine will return a binary connection matrix whose
C  node and link weights are one.
         CALL CHANGE(NON,CONNECT,TEMP)
         CALL CONNECTIVITY(NON,TEMP,CONBO)
       ENDIF
C
C  Calculate the before-degradation term for the Relative Throughput
C  MOE if the MOE was requested.
       IF (RT.EQ.1) CALL RELIABLE_THROUGHPUT(NON,POS,RTBD)
```

```
C
C  Calculate the before-degradation term for the Network Reliability
C  MOE if the MOE was requested.
      IF (NR.EQ.1) THEN
C
C  The CHANGE subroutine will return a binary connection matrix whose
C  node and link weights are one.
        CALL CHANGE(NON,CONNECT,TEMP)
        CALL NETWORK_RELIABILITY(NON,TEMP,POS,BDPDCT)
      ENDIF
C
C***********************************************************************
C
C  Event Block
C
C  Input the changes to the network.
 65   CALL EVENT(NON,CONNECT,POS)
C
C***********************************************************************
C
C  After-Degradation Output
C
C  Calculate and output the Shortest Delay Path MOE if the MOE was
C  requested.
      IF (SDP.EQ.1) CALL OIJKSTRA(NON,CONNECT,1,SPM,WGT2)
C
C  Calculate and output the Highest Reliable Path MOE if the MOE was
C  requested.
      IF (HRP.EQ.1) THEN
C
C  This loop takes the negative logarithm of the probability matrix.
        OO BO I=1,NON
          DO 70 J=1,NON
C
C  If the link or node exists, then take the negative logarithm of its
C  probability of survival.  Otherwise, leave the probability zero.
            IF (POS(I,J).NE.O) THEN
              TEMP(I,J)=-ALOG(POS(I,J))
            ELSE
              TEMP(I,J)=O
            ENDIF
 70       CONTINUE
 BO     CONTINUE
C
C  Call Dijkstra's Shortest Path First Algorithm and pass the negative
C  logarithm probability matrix in place of the connection matrix.
        CALL DIJKSTRA(NON,TEMP,2,HRPM,PROB2)
      ENDIF
C
C  Calculate and output the Reachability MOE if the MOE was requested.
      IF (RCH.EQ.1) THEN
C
```

```
C  This loop sets all link weights to one and all node weights to zero.
       OO 100 I=1,NON
         DO 90 J=1,NON
C
C  Set node weights to zero.
           IF (I.EQ.J) THEN
             TEMP(I,J)=0
C
C  If a link exists between the two nodes being examined, then set its
C  weight to one.
           ELSE IF (POS(I,J).NE.O) THEN
             TEMP(I,J)=1
C
C  If no link exists, then leave the weight zero.
           ELSE
             TEMP(I,J)=0
           ENOIF
 90      CONTINUE
 100   CONTINUE
C
C  Call Oijkstra's Shortest Path First Algorithm and pass the binary
C  connection matrix (whose node weights are zero and link weights are
C  one) in place of the original connection matrix.
         CALL OIJKSTRA(NON,TEMP,3,LENM,LENGTH2)
       ENOIF
C
C  Calculate and output the Maximum Throughput MOE if the MOE was
C  requested.
       IF (MCMF.EQ.1) THEN
C
C  This loop inverts all the node and link weights.
       OO 120 I=1,NON
         OO 110 J=1,NON
C
C  If the link or node exists, then invert its weight.  Otherwise,
C  leave the weight zero.
           IF (CONNECT(I,J).NE.O) THEN
             TEMP(I,J)=1/CONNECT(I,J)
           ELSE
             TEMP(I,J)=0
           ENOIF
 110     CONTINUE
 120   CONTINUE
C
C  Call Ford and Fulkerson's Min-Cut Max-Flow Algorithm and pass the
C  inverted connection matrix in place of the original connection
C  matrix.
         CALL FORO_FULKERSON(NON,TEMP,1,CUT2)
       ENOIF
C
C  Calculate and output the Number of Link Independent Paths MOE if the
C  MOE was requested.
```

```
        IF (NOLIP.EQ.1) THEN
C
C  The CHANGE subroutine will return a binary connection matrix whose
C  node and link weights are one.
          CALL CHANGE(NON,CONNECT,TEMP)
C
C  Call Ford and Fulkerson's Min-Cut Max-Flow Algorithm and pass the
C  binary connection matrix in place of the original connection
C  matrix.
          CALL FORD_FULKERSON(NON,TEMP,2,INO2)
        ENDIF
C
C  Calculate and output the Reliability MOE if the MOE was requested.
        IF (SORL.EQ.1) THEN
C
C  The CHANGE subroutine will return a binary connection matrix whose
C  node and link weights are one.
          CALL CHANGE(NON,CONNECT,TEMP)
C
C  Call the reliability algorithm and pass the binary connection matrix
C  and the probability connection matrix.
          CALL RELIABILITY(NON,TEMP,POS,TOTAL)
        ENDIF
C
C  Calculate the after-degradation term for the Connectivity MOE if the
C  MOE was requested.
        IF (CON.EQ.1) THEN
C
C  The CHANGE subroutine will return a binary connection matrix whose
C  node and link weights are one.
          CALL CHANGE(NON,CONNECT,TEMP)
          CALL CONNECTIVITY(NON,TEMP,CONAO)
          RAT1=CONAO/CONBO
          PRINT 130, RAT1
 130      FORMAT (' The connectivity of the network is ',F5.3)
          PRINT *,' '
        ENDIF
C
C  Calculate the after-degradation term for the Relative Throughput MOE
C  if the MOE was requested.
        IF (RT.EQ.1) THEN
          CALL RELIABLE_THROUGHPUT(NON,POS,RTAO)
          RAT2=RTAO/RTBO
          PRINT 140, RAT2
 140      FORMAT (' The reliable throughput of the network is ',F5.3)
          PRINT *,' '
        ENDIF
C
C  Calculate the after-degradation term for the Network Reliability
C  MOE if the MOE was requested.
        IF (NR.EQ.1) THEN
C
```

```
C  The CHANGE subroutine will return a binary connection matrix whose
C  node and link weights are one.
         CALL CHANGE(NON,CONNECT,TEMP)
         CALL NETWORK_RELIABILITY(NON,TEMP,POS,AOPOCT)

         RAT3=AOPOCT/BOPOCT
         PRINT 150, RAT3
 150     FORMAT (' The network reliability is ',F5.3)
         PRINT *,' '
      ENDIF
C
C  Return to the Event Block.
      GO TO 65
      ENO


      SUBROUTINE CHOOSE_MOE(CON,HRP,MCMF,NOLIP,NR,RCH,RT,SOP,SORL)
C
C  Written by:  Russell O. Thomas
C  Oate:  July 30, 1984
C
C  This is the declaration block.
      INTEGER CON, HRP, MCMF, NOLIP, NR, RCH, RT, SOP, SDRL
C
C  Purpose of the subroutine:
C
C          This subroutine determines which MOE's are required by the
C    user.
C
C  Variables used in the subroutine:
C
C    CON     the answer to the Connectivity MOE question
C                 0  no
C                 1  yes
C    HRP     the answer to the HigHest Reliable Path MOE question
C                 0  no
C                 1  yes
C    MCMF    the answer to the Maximum Flow MOE question
C                 0  no
C                 1  yes
C    NOLIP   the answer to the Number of Link Independent Paths MOE
C            question
C                 0  no
C                 1  yes
C    NR      the answer to the Network Reliability MOE question
C                 0  no
C                 1  yes
C    RCH     the answer to the Reachability MOE question
C                 0  no
C                 1  yes
C    RT      the answer to the Reliable Throughput MOE question
C                 0  no
C                 1  yes
```

```
C     SOP    the answer to the Shortest Oelay Path MOE question
C                 0  no
C                 1  yes
C     SORL   the answer to the Reliablity MOE question
C                 0  no
C                 1  yes
C
C***********************************************************************
C
      TYPE 10
      TYPE 20
      TYPE 30
      TYPE 40
      TYPE 50
 10   FORMAT ('0','    The following is a list of Measures of Effective
     &ness (MOE''s) which this')
 20   FORMAT (' algorithm will calculate, and all the MOE''s will be
     & calculated unless the')
 30   FORMAT (' algorithm is told otherwise.  To prevent a MOE from
     & being calculated, just')
 40   FORMAT (' type in a negative response when prompted.  The default
     & answer is yes (the MOE')
 50   FORMAT (' is needed).')
C
C  The framework is the same for the following blocks of code.  The
C  user is asked if a particular MOE is needed (the answer originally
C  assumed yes).  In a call to the REAO subroutine, the input is read
C  and determined if negative.  If the input was negative, then the
C  answer is no.  If the input was non-negative, then the answer is
C  yes.
      TYPE 60
 60   FORMAT ('0','Is the Shortest Oelay Path MOE needed?')
      SDP=1
      CALL REAO(SOP)
      PRINT 65, SOP
 65   FORMAT (' SOP  = ',I1)
C
      TYPE 70
 70   FORMAT ('0','Is the Highest Reliable Path MOE needed?')
      HRP=1
      CALL REAO(HRP)
      PRINT 75, HRP
 75   FORMAT (' HRP  = ',I1)
C
      TYPE 80
 80   FORMAT ('0','Is the Reachability MOE needed?')
      RCH=1
      CALL READ(RCH)
      PRINT 85, RCH
 85   FORMAT (' RCH  = ',I1)
C
      TYPE 90
 90   FORMAT ('0','Is the Maximum Flow MOE needed?')
      MCMF=1
      CALL REAO(MCMF)
      PRINT 95, MCMF
 95   FORMAT (' MCMF = ',I1)
C
      TYPE 100
 100  FORMAT ('0','Is the Number of Link Independent Paths MOE
     & needed?')
      NOLIP=1
      CALL REAO(NOLIP)
      PRINT 105, NOLIP
 105  FORMAT (' NOLIP= ',I1)
C
      TYPE 110
 110  FORMAT ('0','Is the Source to Oestination Reliability MOE
     & needed?')
      SORL=1
      CALL READ(SORL)
      PRINT 115, SORL
 115  FORMAT (' SORL = ',I1)
C
      TYPE 120
 120  FORMAT ('0','Is the Connectivity MOE needed?')
      CON=1
      CALL REAO(CON)
      PRINT 125, CON
 125  FORMAT (' CON  = ',I1)
C
      TYPE 130
 130  FORMAT ('0','Is the Reliable Throughput needed?')
      RT=1
      CALL REAO(RT)
      PRINT 135, RT
 135  FORMAT (' RT   = ',I1)
C
      TYPE 140
 140  FORMAT ('0','Is the Network Reliability MOE needed?')
      NR=1
      CALL READ(NR)
      PRINT 145, NR
 145  FORMAT (' NR   = ',I1)
      PRINT *,' '
      RETURN
      ENO


      SUBROUTINE REAO(MOE)
C
C  Written by:  Russell O. Thomas
C  Date:  July 30, 1984
C
```

```
C  This is the declaration block.
       CHARACTER*2 CHR
       INTEGER MOE
C
C  Purpose of the subroutine:
C
C          This subroutine determines whether a negative answer has been
C    inputted.
C
C  Variables used in the subroutine:
C
C    CHR  the user input
C    MDE  the answer to the MDE question
C              0  no
C              1  yes
C
C***********************************************************************
C
C  Read the user input.
       READ 5, CHR
 5     FORMAT (A2)
C
C  If the response was negative, set the answer to zero.  Otherwise,
C  leave the answer one.
       IF ((CHR.EQ.'n').DR.(CHR.EQ.'N').OR.(CHR.EQ.'no').DR.(CHR.EQ.
      &'ND')) MDE=D
       RETURN
       END


       SUBRDUTINE DESCRIBE_NETWORK(NON,CDNNECT,PDS)
C
C  Written by:  Russell D. Thomas
C  Date:  July 3D, 1984
C
C  This is the declaration block.
       OIMENSIDN CONNECT(10D,1DO), POS(1DD,1DO)
       INTEGER D, NDN, S
       REAL CONNECT, P, PDS, W
C
C  Purpose of the subroutine:
C
C         This subroutine reads in the network.
C
C  Variables used in the subroutine:
C
C    D    the destination node
C    NDN  the number of nodes in the network
C    P    the probability of link or node survival
C    S    the source node, sometimes both the source and destination
C         node
C    W    the link or node weight
```

```
C
C    CONNECT()  the connection matrix for the network
C    POS()      the probability connection matrix
C
C***********************************************************************
C
      TYPE 10
 10   FORMAT ('0','Input the number of nodes.')
C
C  Read the number of nodes in the network.
      REAO 20, NON
 20   FORMAT (I2)
      PRINT 30, NON
 30   FORMAT (' The number of nodes is ',I2)
      PRINT *,' '
C
C  This loop initializes the connection matrix and the probability
C  connection matrix to zero.
      OO 50 I=1,NON
         OO 40 J=1,NON
            CONNECT(I,J)=0
            POS(I,J)=0
 40      CONTINUE
 50   CONTINUE
C
      TYPE 51
      TYPE 52
 51   FORMAT ('0','     This is the input block to the program.  At this
     & point, input the node, the')
 52   FORMAT (' node weight, and its probability of survival.  Input
     & 0 0 0 when finished.')
C
C  Read the node, its weight, and its probability of survival.
 53   REAO *, S, W, P
      IF (S.EQ.0) GO TO 55
         CONNECT(S,S)=W
         POS(S,S)=P
      PRINT 54, S, W, P
 54   FORMAT (' ',I2,2X,F5.2,2X,F5.3)
      GO TO 53
 55   PRINT *,' '
C
      TYPE 60
      TYPE 70
      TYPE 80
      TYPE 90
      TYPE 100
      TYPE 110
      TYPE 120
 60   FORMAT ('0','     This is the input block for all the
     & bidirectional links.  8idirectional')
 70   FORMAT (' links have the same link weights from source to
     & destination and from destination')
 80   FORMAT (' to source.  Bidirectional links also have equal
     & probabilities of link survival')
 90   FORMAT (' from source to destination and destination to source.
     &  The input block for')
 100  FORMAT (' directed links follows.  Input the source node,
     & destination node, link weight,')
 110  FORMAT (' and the probability of link survival.  Input 0 0 0 0
     & when finished with the')
 120  FORMAT (' bidirectional link input.')
C
C  Read the two nodes, the link weight, and the probability of link
C  survival.
 130  REA0 *, S, O, W, P
      IF (S.EQ.0) GO TO 150
         CONNECT(S,O)=W
         CONNECT(O,S)=W
         POS(S,O)=P
         POS(O,S)=P
      PRINT 140, S, O, W, P
 140  FORMAT (' ',I2,2X,I2,2X,F5.2,2X,F5.3)
      GO TO 130
 150  PRINT *,' '
C
      TYPE 160
      TYPE 170
      TYPE 180
 160  FORMAT ('0','     This is the input block for the directed links.
     &  Input the source node,')
 170  FORMAT (' destination node, link weight, and the probability of
     & link survival.  Input')
 180  FORMAT (' 0 0 0 0 when finished.')
C
C  Read the source node, the destination node, the link weight, and the
C  probability of link survival.
 190  REA0 *, S, O, W, P
      IF (S.EQ.0) GO TO 210
         CONNECT(S,O)=W
         POS(S,O)=P
      PRINT 200, S, O, W, P
 200  FORMAT (' ',I2,2X,I2,2X,F5.2,2X,F5.3)
      GO TO 190
 210  PRINT *,' '
      RETURN
      END


      SUBROUTINE CHANGE(NON,CONNECT,TEMP)
C
C  Written by:  Russell O. Thomas
C  Oate:  July 30, 1094
C
```

```
C  This is the declaration block.
      OIMENSION CONNECT(100,100), TEMP(100,100)
      REAL CONNECT, TEMP
      INTEGER NON
C
C  Purpose of the subroutine:
C
C         This subroutine calculates the binary connection matrix for
C    the network.
C
C  Variables used in the subroutine:
C
C    NON  the number of nodes in the network
C
C    CONNECT()  the connection matrix for the network
C    TEMP()     the binary connection matrix whose node and link
C               weights are one
C
C***********************************************************************
C
C  This loop sets all node and link weights to one.
      00 80 I=1,NON
        00 70 J=1,NON
C
C  If the node or link exists, then set its weight to one.  Otherwise,
C  leave the weight zero.
          IF (CONNECT(I,J).NE.0) THEN
            TEMP(I,J)=1
          ELSE
            TEMP(I,J)=0
          ENOIF
 70     CONTINUE
 80   CONTINUE
      RETURN
      ENO


      SUBROUTINE EVENT(NON,CONNECT,POS)
C
C  Written by:  Russell O. Thomas
C  Oate:  July 30, 1984
C
C  This is the declaration block.
      OIMENSION CONNECT(100,100),POS(100,100)
      REAL CONNECT, P, POS, W
      INTEGER O, NON, S, TST
C
C  Purpose of the subroutine:
C
C         This subroutine reads in changes to the network.
C
C  Variables used in the subroutine:
```

```
C
C     D    the destination node
C     NON  the number of nodes in the network
C     P    the probability of link or node survival
C     S    the source node, sometimes the source and destination
C          node
C     TST  the user input
C     W    the link or node weight
C
C     CONNECT()  the connection matrix for the network
C     POS()      the probability connection matrix
C
C*********************************************************************
C
C  Read whether or not the user wishes to continue.
      TYPE 5
 5    FORMAT ('0','Should the program continue? (default yes)')
      TST=1
      CALL REAO(TST)
C
C  If the response was negative, then terminate the program.
      IF (TST.EQ.0) STOP
C
      TYPE 10
      TYPE 20
      TYPE 30
      TYPE 40
 10   FORMAT ('0','     This is the event block for the program.  At
     & this point, the weights and')
 20   FORMAT (' probabilities of the nodes and links can be changed,
     & or nodes and links may be')
 30   FORMAT (' removed from the network entirely.  Input the
     &  degradated nodes and links only.')
 40   FORMAT (' All other nodes and links will remain the same.')
C
      TYPE 50
      TYPE 60
 50   FORMAT ('0','This is the event block for the nodes.  Input the
     & node, the new node weight, and')
 60   FORMAT (' the new probability of node survival.  Input 0 0 0 when
     & finished.')
C
C  Read the node, its weight, and its probability of survival.
 70   REAO *, S, W, P
      IF (S.EQ.0) GO TO 100
         CONNECT(S,S)=W
         POS(S,S)=P
C
C  If either the node weight or the probability of node survival was
C  set to zero, then remove the node from the network.
         IF ((W.EQ.0).OR.(P.EQ.0)) THEN
            DO 80 I=1,NON
```

```
               CONNECT(I,S)=0
               CONNECT(S,I)=0
               POS(I,S)=0
               POS(S,I)=0
 80         CONTINUE
            W=0
            P=0
         ENOIF
      PRINT 90, S, W, P
 90   FORMAT (' ',I2,2X,F5.2,2X,F5.3)
      GO TO 70
 100  PRINT *,' '
C
      TYPE 110
      TYPE 120
      TYPE 125
 110  FORMAT ('0','This is the event block for the bidirectional links.
     &  Input the two nodes, the')
 120  FORMAT (' new link weight, and the new probability of link
     & survival. Input 0 0 0 0')
 125  FORMAT (' when finished.')
C
C  Read the two nodes, the link weight, and the probability of link
C  survival.
 130  READ *, S, O, W, P
      IF (S.EQ.0) GO TO 150
         CONNECT(S,O)=W
         CONNECT(O,S)=W
         POS(S,O)=P
         POS(O,S)=P
C
C  If either the link weight or the probability of link survival was
C  set to zero, then remove the link from the network.
         IF ((W.EQ.0).OR.(P.EQ.0)) THEN
            CONNECT(S,O)=0
            CONNECT(D,S)=0
            POS(S,O)=0
            POS(O,S)=0
            W=0
            P=0
         ENOIF
      PRINT 140, S, O, W, P
 140  FORMAT (' ',I2,2X,I2,2X,F5.2,2X,F5.3)
      GO TO 130
 150  PRINT *,' '
C
      TYPE 160
      TYPE 170
      TYPE 180
 160  FORMAT ('0','This is the event block for the directed links.
     &  Input the source node, the')
 170  FORMAT (' destination node, the new link weight, and the new
     & probability of link survival.')
 180  FORMAT (' Input 0 0 0 0 when finished.')
C
C  Read the source node, the destination node, the link weight, and the
C  probability of link survival.
 190  READ *, S, D, W, P
      IF (S.EQ.0) GO TO 210
         CONNECT(S,D)=W
         POS(S,D)=P
C
C  If either the link weight or the probability of link survival was
C  set to zero, then remove the link from the network.
        IF ((W.EQ.0).OR.(P.EQ.0)) THEN
           CONNECT(S,D)=0
           POS(S,D)=0
           W=0
           P=0
        ENDIF
      PRINT 200, S, D, W, P
 200  FORMAT (' ',I2,2X,I2,2X,F5.2,2X,F5.3)
      GO TO 190
 210  PRINT *,' '
      RETURN
      END


      SUBROUTINE DIJKSTRA(NON,CONNECT,A,SPM,WGT)
C
C  Written by:  Russell O. Thomas
C  Date:  July 26, 1984
C
C  This is the declaration block.
       DIMENSION CONNECT(100,100), SPM(100,100), WGT(100,100)
       DIMENSION ADJ(100), LABEL (100), WEIGHT(100)
       INTEGER A, ADJ, I, J, K, LABEL, NON, S, SPM
       REAL CONNECT, MIN, WEIGHT, WGT
C
C  Purpose of the subroutine:
C
C         This subroutine finds the shortest delay path (or highest
C    reliable path, or shortest length path) between all sources and
C    destinations.  The subroutine also calculates the routing matrix
C    for the network.
C
C  Variables used in the subroutine:
C
C    A     the Measure of Effectiveness criterion
C              1  shortest delay path
C              2  highest reliable path
C              3  minimum number of links
C              4  minimum number of links
C    I     the adjacent node, sometimes the source node
```

```
C   . J    the destination node
C     K    the intermediate source node
C     NON  the number of nodes in the network
C     MIN  the minimum weight along the shortest path
C     S    the source node
C
C     AOJ()      the intermediate source nodes which form the path from
C                source to destination
C     LABEL()    the label for the node
C                     0  temporary
C                     1  permanent
C     WEIGHT()   the total weight along the path from source to
C                destination
C
C     CONNECT()   the connection matrix for the network
C     SPM()       the routing directory for routing messages through the
C                 network
C     WGT()       the total weight from source to destination
C
C***********************************************************************
C
C  Initialization
C
C  Choose the source node.
 200   DO 360 S=1,NON
C
C  This loop initializes the arrays.
       DO 220 I=1,NON
C
C  Set the weight to all nodes equal to infinity.
         WEIGHT(I)=1.0E38
C
C  Set the adjacent nodes to zero (meaning none found yet).
         AOJ(I)=0
C
C  Make the weights to all nodes temporary (temporary=0, permanent=1).
         LABEL(I)=0
 220   CONTINUE
C
C  Set the weight from the source node to itself equal to the weight at
C  that node.
       WEIGHT(S)=CONNECT(S,S)
C
C  Make the length from the source to the source permanent.  (In other
C  words, there is no shorter path from the source to the source.)
       LABEL(S)=1
C
C  Make the intermediate source node the source node.
       K=S
C
C***********************************************************************
C
```

```
C  Label Updating
C
C  This statement checks to see if all the paths have been made
C  permanent.  If they have, then go to the output block.  If not,
C  continue.
        DO 280 J=1,NON-1
C
C  This loop finds all the adjacent nodes which are connected to the
C  intermediate source node, and stores their weight from the original
C  source node to the new adjacent node.
          DO 240 I=1,NON
C
C  If there is a link present, then the node being examined is an
C  adjacent node.
            IF (CONNECT(K,I).NE.0) THEN
C
C  If the label on the adjacent node is temporary, then continue.
              IF (LABEL(I).NE.1) THEN
C
C  If the weight from the original source node to the adjacent node
C  being examined is greater than or equal to the weight the adjacent
C  node already has, then this particular path being examined is not(!)
C  the shortest path from the original source node to the adjacent node
C  being examined.
                IF (WEIGHT(K)+CONNECT(K,I)+CONNECT(I,I).LT.WEIGHT(I))
     &          THEN
C
C  If at this point, the path being examined is(!) a shorter path to
C  that particular adjacent node from the original source node.  Put
C  the intermediate source node on the adjacent node's adjacent list.
                  ADJ(I)=K
C
C  Record the new weight in the weight array.
                  WEIGHT(I)=WEIGHT(K)+CONNECT(K,I)+CONNECT(I,I)
                ENDIF

              ENDIF
            ENDIF
 240      CONTINUE
C
C***********************************************************************
C
C  Making a Label Permanent
C
C  Set the minimum weight to infinity.
          MIN=1.0E38
C
C  Set the intermediate source node to zero.
          K=0
C
C  This loop chooses the next intermediate source node by selecting the
C  node with the smallest weight from the original source node.
          DO 260 I=1,NON
```

```
C
C  If the node has already been made permanent, then choose another
C  because it has already been an intermediate source node.  Otherwise,
C  continue.
            IF (LABEL(I).EQ.0) THEN
C
C  If the total weight from the original source node to the node being
C  examined is less than the minimum weight, then continue.
              IF (WEIGHT(I).LT.MIN) THEN
C
C  If at this point, then there is a new minimum path weight and a new
C  intermediate source node.  Set the minimum weight equal to the newly
C  found total weight.
                MIN=WEIGHT(I)
C
C  Make the node being examined the new intermediate source node.
                K=I
              ENDIF
            ENDIF
 260      CONTINUE
C
C  If the intermediate source node remained zero, then the nodes left
C  temporary cannot be reached from the original source node.
          IF (K.EQ.0) GO TO 300
C
C  Make the new intermediate source node permanent.
          LABEL(K)=1
 280    CONTINUE
C
C*************************************************************************
C
C  Path Block
C
C  This loop calculates the routing directory for the network, and
C  saves the total weight from the source to the destinations.
 300    DO 340 I=1,NON
C
C  Save the total weight from the source to destination.
          WGT(S,I)=WEIGHT(I)
C
C  If no path exists between the source and destination being examined,
C  then enter a zero in the routing directory.
          IF (ADJ(I).EQ.0) THEN
            SPM(S,I)=0
C
C  If the destination node is an adjacent node to the source, then
C  enter the destination node in the routing directory.
          ELSE IF (ADJ(I).EQ.S) THEN
            SPM(S,I)=I
C
C  If the destination node is not adjacent to the source, then retrace
C  the path to determine the adjacent node to the source which lies
```

```
C  along the path.
         ELSE
           X=I
 32D       Y=X
           X=ADJ(X)
           IF (X.NE.S) GD TD 320
           SPM(S,I)=Y
         ENDIF
 340    CDNTINUE
 360  CDNTINUE
C
C  If the call to Dijkstra's Shortest Path First Algorithm was made
C  from the reliability algorithm, then return.
      IF (A.EQ.4) RETURN
C
C***********************************************************************
C
C  Output Block
C
C  Print the total path weight and the routing directory.
      DD 4D0 I=1,NDN
        DD 380 J=1,NON
          IF (A.EQ.1) THEN
            IF (WGT(I,J).LT.1.DE1D) THEN
              PRINT 365, I, J, SPM(I,J), I, J, WGT(I,J)
 365          FDRMAT (' SPM(',I2,',',I2,') = ',I2,10X,'WGT(',I2,',',I2,
     &        ')   =',F5.2)
            ELSE
              PRINT 366, I, J, SPM(I,J), I, J
 366          FORMAT (' SPM(',I2,',',I2,') = ',I2,10X,'WGT(',I2,',',I2,
     &        ')   = infinity')
            ENDIF
          ENDIF
          IF (A.EQ.2) PRINT 37D, I, J, SPM(I,J), I, J, EXP(-WGT(I,J))
 37D        FORMAT (' HRPM(',I2,',',I2,')= ',I2,10X,'PROB(',I2,',',I2,
     &      ') = ',F5.3)
          IF (A.EQ.3) THEN
            IF (WGT(I,J).LT.1.0E10) THEN
              PRINT 375, I, J, SPM(I,J), I, J, WGT(I,J)
 375          FDRMAT (' LENM(',I2,',',I2,')= ',I2,1DX,'LENGTH(',I2,',',
     &        I2,')= ',F3.1)
            ELSE
              PRINT 376, I, J, SPM(I,J), I, J
 376          FDRMAT (' LENM(',I2,',',I2,')= ',I2,1DX,'LENGTH(',I2,',',
     &        I2,')= infinity')
            ENDIF
          ENDIF
 38D    CDNTINUE
 4DD  CDNTINUE
      PRINT *,' '
      RETURN
      END
      SUBROUTINE FORD_FULKERSON(NON,CONNECT,W,CUT)
C
C  Written by:  Russell O. Thomas
C  Date:  July 26, 1984
C
C  This is the declaration block.
      DIMENSION CONNECT(100,100), CUT(100,100), FLOW(100,100)
      DIMENSION ADJ(100), DIR(100), TMPFLO(100), LABEL(100), SCAN(100)
      INTEGER ADJ, D, DIR, I, J, LABEL, NON, S, SCAN, W, Z
      REAL CONNECT, CUT, TMPFLO, FLOW
C
C  Purpose of the subroutine:
C
C         This subroutine finds the maximum throughput (or number of
C     link independent paths) between all sources and destinations.
C
C  Variables used in the subroutine:
C
C     D     the destination node
C     I     the intermediate source node, sometimes the source node
C     J     the intermediate destination node, sometimes the destination
C           node
C     NON   the number of nodes
C     S     the source node
C     W     the Measure of Effectiveness criterion
C                1  maximum throughput
C                2  link independent paths
C     Z     the pointer node (intermediate destination node when
C           retracing the path from source to destination).
C
C     ADJ()      the intermediate source nodes which form the path from
C                source to destination
C     DIR()      the direction of flow
C                     -1  flow leaving the intermediate destination node
C                      0  neutral (no flow entering or leaving)
C                      1  flow entering the intermediate destination node
C     LABEL()    a label on the node
C                     0  unlabeled
C                     1  labeled
C     SCAN()     a label on the node
C                     0  unscanned
C                     1  scanned
C     TMPFLO()   the temporary flow being pushed along the path from
C                source to destination
C
C     CONNECT()   the connection matrix for the network
C     CUT()       the maximum flow between source and destination
C     FLOW()      the augmented flow in the network
C
C***********************************************************************
```

```
C
C  Initialization
C
C  This loop initializes the output array to zero.
 160  DO 200 I=1,NON
        DO 180 J=1,NON
          CUT(I,J)=0
 180    CONTINUE
 200  CONTINUE
C
C  Choose the source node.
      DO 440 S=1,NON
C
C  Choose the destination node.
        DO 420 D=1,NON
C
C  If the source node and the destination node are one and the same,
C  then choose a new destination node.  Otherwise, continue.
          IF (S.NE.D) THEN
C
C  This loop initializes the flow along all links to zero.
            DO 240 I=1,NON
              DO 220 J=1,NON
                FLOW(I,J)=0
 220          CONTINUE
 240        CONTINUE
C
C  This loop initializes the arrays.
 260        DO 280 I=1,NON
C
C  Set the adjacent node to zero (meaning none found yet).
              ADJ(I)=0
C
C  Set the direction of flow to neutral (meaning no flow into or out of
C  the node yet).
              DIR(I)=0
C
C  Make all nodes unlabelled and unscanned.
              LABEL(I)=0
              SCAN(I)=0
C
C  Set the temporary flow entering and leaving all nodes equal to
C  infinity.
              TMPFLO(I)=1.0E38
 280        CONTINUE
C
C  Make the source node labelled and unscanned.
            LABEL(S)=1
C
C***********************************************************************
C
C  Label Updating
```

```
C
C  This outer loop is necessary because the search for the minimum cut
C  is very dependent on how the nodes are numbered.
           DO 380 K=1,NON
C
C  This loop chooses an intermediate source node along a path from the
C  source node to the destination node.
             DO 360 I=1,NON
C
C  If the node being examined is labelled and unscanned, then continue.
               IF ((LABEL(I).EQ.1).AND.(SCAN(I).EQ.0)) THEN
C
C  This loop chooses an intermediate destination node along a path from
C  the source to the destination node.
                 DO 340 J=1,NON
C
C  If there is no link present, try another node.
                   IF (CONNECT(I,J).NE.0) THEN
C
C  If the adjacent node being examined is unlabelled, then the link
C  lies along a path from the source to destination.
                     IF (LABEL(J).EQ.0) THEN
C
C  If there is flow from the intermediate destination node to the
C  intermediate source node, then continue.
                       IF (FLOW(J,I).GT.0) THEN
C
C  Put the intermediate source node on the intermediate destination
C  node's adjacent list.
                         ADJ(J)=I
C
C  Direct the flow out of the intermediate destination node.
                         DIR(J)=-1
C
C  If the temporary flow from the intermediate destination node to the
C  intermediate source node is less than flow already along that link,
C  then push that amount of temporary flow along that link.  Otherwise,
C  leave the flow the same.
                         IF (TMPFLO(I).LT.FLOW(J,I)) THEN
                           TMPFLO(J)=TMPFLO(I)
                         ELSE
                           TMPFLO(J)=FLOW(J,I)
                         ENDIF
C
C  Make the intermediate destination node labelled and unscanned.
                         LABEL(J)=1
C
C  Make the intermediate source node labelled and scanned.
                         SCAN(I)=1
                       ENDIF
C
C  If the capacity for flow through the intermediate destination node
```

```
C  is less than the capacity for flow through the adjacent link from
C  the intermediate source node to the intermediate destination node,
C  then use the capacity for flow through the intermediate destination
C  node in place of the capacity for flow through the link.
                        IF (CDNNECT(J,J).LT.CONNECT(I,J)) THEN
C
C  If the capacity of the intermediate destination node is greater than
C  the flow along the adjacent link from the intermediate source node
C  to the intermediate destination node, then continue.
                          IF (CONNECT(J,J).GT.FLDW(I,J)) THEN
C
C  Put the intermediate source node on the intermediate destination
C  node's adjacent list.
                            ADJ(J)=I
C
C  Direct the flow into the intermediate destination node.
                            DIR(J)=1
C
C  If the temporary flow from the intermediate source node to the
C  intermediate destination node is less than the amount of unused
C  capacity at the intermediate destination node, then push that amount
C  of flow to the intermediate destination node.  If the temoorary flow
C  is greater than the unused capacity at the intermediate destination
C  node, then only push as much flow as the node can handle.
                            IF (TMPFLD(I).LT.CDNNECT(J,J)-FLDW(I,J))
     &                      THEN
                              TMPFLD(J)=TMPFLD(I)
                            ELSE
                              TMPFLD(J)=CDNNECT(J,J)-FLDW(I,J)
                            ENDIF
C
C  Make the intermediate destination node labelled and unscanned.
                            LABEL(J)=1
C
C  Make the intermediate source node labelled and scanned.
                            SCAN(I)=1
                          ENDIF
                          GD TD 30D
                        ENDIF
C
C  If the flow from the intermediate source node to the intermediate
C  destination node is less than or equal to the capacity of the link,
C  then continue.
                        IF (CDNNECT(I,J).GT.FLDW(I,J)) THEN
C
C  Put the intermediate source node on the intermediate destination
C  node's adjacent list.
                          ADJ(J)=I
C
C  Direct the flow into the intermediate destination node.
                          DIR(J)=1
C
```

```
C  If the temporary flow from the intermediate source node to the
C  intermediate destination node is less than the amount of unused
C  capacity of the link, then push that amount of flow to the
C  intermediate destination node.  If the temporary flow is greater
C  than the unused capacity of the link, then only push as much flow as
C  the link can handle.
                          IF (TMPFLO(I).LT.CONNECT(I,J)-FLOW(I,J)) THEN
                            TMPFLO(J)=TMPFLO(I)
                          ELSE
                            TMPFLO(J)=CONNECT(I,J)-FLOW(I,J)
                          ENDIF
C
C  Make the intermediate destination node labelled and unscanned.
                          LABEL(J)=1
C
C  Make the intermediate source node labelled and scanned.
                          SCAN(I)=1
                        ENDIF
C
C***********************************************************************
C
C  Flow Augmentation
C
C  If the destination node has been labeled, then augment the flow
C  along the path found.
 300                      IF (LABEL(O).EQ.1) THEN
C
C  Make the intermediate destination node the destination node.
                          Z=O
C
C  If the flow is directed into the intermediate destination node, then
C  increase the flow along the link between the previous intermediate
C  source node and the intermediate destination node by the minimum
C  flow along the path being examined.
 320                      IF (DIR(Z).EQ.1) FLOW(ADJ(Z),Z)=
     &                    FLOW(ADJ(Z),Z)+TMPFLO(O)
C
C  If the flow is directed out of the intermediate destination node,
C  then decrease the flow along the link between the previous
C  intermediate source node and the intermediate destination node by
C  the minimum flow along the path being examined.
                          IF (DIR(Z).EQ.-1) FLOW(ADJ(Z),Z)=
     &                    FLOW(ADJ(Z),Z)-TMPFLO(O)
C
C  If the previous intermediate source node is the source node, then
C  repeat the process with the improved flow calculated above.  If the
C  previous intermediate source node was not the source node, then make
C  the previous intermediate source node the intermediate destination
C  node and continue retracing the path to the source node.
                          IF (ADJ(Z).EQ.S) THEN
                            GO TO 260
                          ELSE
                          Z=AOJ(Z)
                          GO TO 320
                        ENDIF
                      ENDIF
                    ENDIF
                  ENOIF
 340            CDNTINUE
              ENDIF
 360        CONTINUE
 380      CONTINUE
C
C  Since the flow entering the destination node is the same as the flow
C  across the minimum cut (if the destination node was not in the
C  minimum cut), then sum the flow entering the destination node to
C  determine the maximum throughput between source and destination.
          DO 400 I=1,NON
            CUT(S,O)=CUT(S,D)+FLDW(I,O)
 4DD      CONTINUE
C
C  It is possible that the destination node was part of the minimum cut
C  (and this will happen if the node has a very low throughput).  If
C  this is so, then the minimum cut has no meaning, and the destination
C  node is the minimum cut.  It is also possible that the minimum cut
C  is valid, but the source node is incapable of delivering that amount
C  of flow.  In this case, the source node is the minimum cut.
          IF (W.EQ.1) THEN
            IF (CUT(S,D).NE.D) THEN
              IF (CONNECT(S,S).LT.CDNNECT(D,D)) THEN
                IF (CONNECT(S,S).LT.CUT(S,D)) CUT(S,D)=
     &          CONNECT(S,S)
              ELSE
                IF (CONNECT(D,D).LT.CUT(S,D)) CUT(S,D)=
     &          CDNNECT(D,D)
              ENDIF
            ENDIF
          ENDIF

        ENDIF
 420    CONTINUE
 440  CONTINUE
C
C***************************************************************************
C
C  Output Block
C
C  Print the maximum throughput.
      DD 48D I=1,NDN
        DO 460 J=1,NON
          IF (W.EQ.1) PRINT 445, I, J, CUT(I,J)
 445        FORMAT (' CUT(',I2,',',I2,')= ',F7.5)
          IF (W.EQ.2) PRINT 45D, I, J, CUT(I,J)
 45D        FDRMAT (' IND(',I2,',',I2,')= ',F3.1)
 460    CONTINUE
 480  CONTINUE
      PRINT *,' '
      RETURN
      ENO


      SU8ROUTINE RELIABILITY(NON,CONNECT,POS,TOTAL)
C
C  Written by:  Russell O. Thomas
C  Oate:  July 28, 1984
C
C  This is the declaration block.
      OIMENSION CONNECT(100,100), POS(100,100), REL(100,100)
      OIMENSION SAVE(100,100), SPM(100,100), WGT(100,100)
      OIMENSION OEST(100), SOURCE(100), PROB(100), UPOOWN(100)
      INTEGER A, OEST, I, J, K, M, NOLAN, NON, SOURCE, SPM, TOTAL
      INTEGER UPOOWN
      REAL CON, CONNECT, POC, POS, REL, SAVE, PROB, SUM, T, WGT
C
C  Purpose of the subroutine:
C
C        This subroutine calculates the probability that at least one
C    path exists between a source and destination using state
C    enumeration.  The subroutine also calculates the probability that
C    the network is connected.
C
C  Variables used in the subroutine:
C
C    A       the Measure of Effectiveness criterion for Oijkstra's
C            Shortest Path First Algorithm
C                 1  shortest delay path
C                 2  highest reliable path
C                 3  minimum number of links
C                 4  minimum number of links
C
C    CON
C    I       the source node
C    J       the destination node
C    K
C    M       the number of up links and nodes
C    NOLAN   the number of links and nodes in the network
C    NON     the number of nodes in the network
C    TOTAL   the number of connected nodes before degradation
C
C    OEST()    the destination node
C    PROB()    the probability of the link or node survival
C    SOURCE()  the source node
C    UPOOWN()  the state of the link or node
C                   0  down
C                   1  up
C
C    CONNECT()  the connection matrix for the network
```

```
C     POS()        the probability connection matrix
C     SPM()        the routing directory for the network
C     WGT()        the total weight from source to destination
C
C***********************************************************************
C
C  Initialization
C
C  Set the number of links and nodes in the network to zero.
      NOLAN=0
C
C  Set the node/link number to one.
      K=1
C
C  This loop numbers all the nodes and links.
      DO 220 I=1,NON
        DO 200 J=1,NON
C
C  If the link or node exists, then number it.
          IF (CONNECT(I,J).NE.0) THEN
C
C  Store, for quick reference, the probability of survival, the source
C  node, and the destination node under its node/link number.
            PROB(K)=POS(I,J)
            SOURCE(K)=I
            DEST(K)=J
C
C  Increment the node/link number.
            K=K+1
C
C  Increment the number of links.
            NOLAN=NOLAN+1
          ENDIF
 200    CONTINUE
 220  CONTINUE
C
C  This loop saves the connection matrix for future use and sets the
C  reliability matrix to zero.
      DO 260 I=1,NON
        DO 240 J=1,NON
          SAVE(I,J)=CONNECT(I,J)
          REL(I,J)=0
 240    CONTINUE
 260  CONTINUE
C
C  Initialize the states of all the nodes and links to down.
      DO 320 I=1,NOLAN
        UPDOWN(I)=0
 320  CONTINUE
C
C  Make the probability that the network is connected zero.
      POC=0
```

```
C
C  Set the number of up links and nodes to zero.
      M=0
C
C***********************************************************************
C
C  State Block
C
C  Point the node/link number to the first
 340  K=1
C
C  0=OOWN, 1=UP (LINK)
 360  IF (UPOOWN(K).EQ.0) THEN
        M=M+1
C
C  Restore the link or node.
        UPOOWN(K)=1
C
C  Restore the connection matrix.
        00 400 I=1,NON
          00 380 J=1,NON
            CONNECT(I,J)=SAVE(I,J)
 380      CONTINUE
 400    CONTINUE
C
C  This loop removes all the down nodes and links from the network.
        00 440 I=1,NOLAN
C
C  If the node or link being examined is down, then remove it from the
C  network.
          IF (UPOOWN(I).EQ.0) THEN
            CONNECT(SOURCE(I),OEST(I))=0
C
C  If the node being examined is down, then remove all of its adjacent
C  links.
            IF (SOURCE(I).EQ.OEST(I)) THEN
              DO 420 J=1,NON
                CONNECT(SOURCE(I),J)=0
                CONNECT(J,OEST(I))=0
 420          CONTINUE
            ENOIF
          ENOIF
 440    CONTINUE
C
C  Call Oijkstra's Shortest Path First Algorithm to determine the
C  routing matrix for the network.  The routing matrix is just a form
C  of the reachability matrix.
        CALL OIJKSTRA(NON,CONNECT,4,SPM,WGT)
C
C  Find the product of the probabilities of failures of the down
C  elements and the probabilities of successes of the up elements.
        T=1
      DO 460 I=1,NOLAN
          IF (UPDOWN(I).EQ.0) THEN
            T=T*(1-PROB(I))
          ELSE
            T=T*PROB(I)
          ENDIF
 460    CONTINUE
C
C  If a path exists between the source and destination being examined,
C  then add in the probability to the total path reliability.
        DO 500 I=1,NON
          DO 480 J=1,NON
            IF (SPM(I,J).NE.0) REL(I,J)=REL(I,J)+T
 480      CONTINUE
 500    CONTINUE
C
C  Set the number of connected node pairs to zero.
        SUM=0
C
C  This loop totals the number of connected node pairs.
        DO 540 I=1,NON
          DO 520 J=1,NON
            IF (SPM(I,J).NE.0) SUM=SUM+1
 520      CONTINUE
 540    CONTINUE
C
C  Calculate the connectivity of the network.
        CON=SUM/TOTAL
C
C  If the network has as many connected node pairs as the original
C  network, then add in the probability to the network reliability.
        IF (CON.EQ.1) POC=POC+T
C
C  If all links and nodes are up, then go to the Output Block.
        IF (M.EQ.NOLAN) GO TO 560
        GO TO 340
      ELSE
C
C  Decrement the number of up links.
        M=M-1
C
C  Set the link or node to a down state.
        UPDOWN(K)=0
C
C  Increment the node/link pointer.
        K=K+1
        GO TO 360
      ENDIF
C
C
C*************************************************************************
C
```

```
C  Output Block
C
C  Print the source to destination reliability.
 560  DO 600 I=1,NON
        DO 580 J=1,NON
          PRINT 570, I, J, REL(I,J)
 570      FORMAT (' REL(',I2,',',I2,') = ',F5.3)
 580    CONTINUE
 600  CONTINUE
      PRINT *,' '
C
C  Print the connected network reliability.
      PRINT 620, POC
 620  FORMAT (' The probability the network is connected is ',F5.3)
      PRINT *,' '
      RETURN
      END

      SUBROUTINE CONNECTIVITY(NON,CONNECT,CON)
C
C  Written by:  Russell O. Thomas
C  Date:  July 28, 1984
C
C  This is the declaration block.
      DIMENSION CONNECT(100,100), SPM(100,100), WGT(100,100)
      INTEGER CON, NON, SPM
      REAL CONNECT, WGT
C
C  Purpose of the subroutine:
C
C         This subroutine determines the before-degradation and
C    after-degradation terms for the Connectivity MOE.
C
C  Variables used in the subroutine:
C
C    CON  the number of connected node pairs
C    NON  the number of nodes in the network
C
C    CONNECT()  the connection matrix for the network
C    SPM()      the routing directory for routing messages
C    WGT()      the total weight from source to destination
C
C***************************************************************************
C
C  Call Dijkstra's Shortest Path First Algorithm to determine the
C  routing matrix for the network.  The routing matrix is just a form
C  of the reachability matrix.
      CALL DIJKSTRA(NON,CONNECT,4,SPM,WGT)
C
C  Set the number of connected node pairs to zero.
      CON=0
C
```

```
C  This loop totals the number of connected node pairs.
      00 20 I=1,NON
        00 10 J=1,NON
          IF (SPM(I,J).NE.0) CON=CON+1
 10     CONTINUE
 20   CONTINUE
      RETURN
      ENO


      SUBROUTINE RELIABLE_THROUGHPUT(NON,POS,RT)
C
C  Written by:  Russell D. Thomas
C  Oate:  July 28, 1984
C
C  This is the declaration block.
      OIMENSION POS(100,100)
      INTEGER NON
      REAL POS, RT
C
C  Purpose of the subroutine:
C
C         This subroutine calculates the before-degradation and
C    after-degradation terms for the Relative Throughput MOE.
C
C  Variables uesd in the subroutine:
C
C    NON  the number of nodes in the network
C    RT   the sum of the link and node probabilities
C
C    POS()  the probability of survival matrix
C
C***************************************************************************
C
C  Set the sum of the link and node probabilities to zero.
      RT=0
C
C  This loop sums the link and node probabilities.
      DO 20 I=1,NON
        00 10 J=1,NON
          RT=RT+POS(I,J)
 10     CONTINUE
 20   CONTINUE
      RETURN
      ENO


      SUBROUTINE NETWORK_RELIABILITY(NON,CONNECT,POS,PDCT)
C
C  Written by:  Russell O. Thomas
C  Date:  August 1, 1984
C
```

```
C  This is the declaration block.
      OIMENSION CONNECT(100,100), POS(100,100)
      INTEGER CON, NON
      REAL CONNECT, PDCT, POS, RT
C
C  Purpose of the subroutine:
C
C         This subroutine calculates the before-degradation and
C    after-degradation terms for the Network Reliability MOE.
C
C  Variables used in the subroutine:
C
C    CON   the number of connected node pairs
C    NON   the number of nodes in the network
C    PDCT  the product of the number of connected node pairs and the
C          sum of the link and node probabilities
C    RT    the sum of the link and node probabilities
C
C    CONNECT()  the connection matrix for the network
C    POS()      the probability connection matrix
C
C************************************************************************
C
      CALL CONNECTIVITY(NON,CONNECT,CON)
      CALL RELIABLE_THROUGHPUT(NON,POS,RT)
      PDCT=CON*RT
      RETURN
      END
```

## Appendix B: A BASIC GRAFTHY

```
1000 'PROGRAM GRAFTHY
1005 '
1010 'Written by:  Russell D. Thomas
1015 'Date:  August 29, 1984
1020 '
1025 '****************************************************************************
1030 '
1035 'CHOOSE THE MOE'S
1040 '
1045 'Purpose of the routine:
1050 '
1055 '      This routine determines which MOE's are required by the user.
1060 '
1065 'Variables used in the routine:
1070 '
1075 '      CON%    the answer to the Connectivity MOE question
1080 .'     HRP%    the answer to the Highest Reliable Path question
1085 '      MCMF%   the answer to the Maximum Flow MOE question
1090 '      NOLIP%  the answer to the Number of Link Independent Paths MOE
1095 '              question
1100 '      NR%     the answer to the Network Reliability MOE question
1105 '      RCH%    the answer to the Reachibility MOE question
1110 '      RT%     the answer to the Reliable Throughput MOE question
1115 '      SDP%    the answer to the Shortest Delay Path MOE question
1120 '      SDRL%   the answer to the Reliability MOE question
1125 '
1130 'For the above MOE's:  0  no
1135 '                      1  yes
1140 '
1145 '****************************************************************************
1150 '
1155 CLS
1160 PRINT "     The following is a list of Measures of Effectiveness (MOE's) wh
ich this"
1165 PRINT "algorithm will calculate, and all the MOE's will be calculated unles
s the"
1170 PRINT "algorithm is told otherwise.  To prevent a MOE from being calculated
, just"
1175 PRINT "type in a negative response when prompted.  The default answer is ye
s (the MOE"
1180 PRINT "is needed)."
1185 '
1190 'The framework is the same for the following blocks of code.  The user is
1195 'asked if a particular is needed (the answer originally assumed yes).  The
1200 'input is read and determined if negative.  If the input was negative, then
1205 'the answer is no.  If the input was non-negative, then the answer is yes.
1210 '
1215 SDP%=1
1220 PRINT
1225 INPUT "Is the Shortest Delay Path MOE needed";CHAR$
1230 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN SDP%=0
1235 PRINT "SDP=";SDP%
1240 LPRINT "SDP=";SDP%
1245 '
1250 HRP%=1
```

```
1255 PRINT
1260 INPUT "Is the Highest Reliable Path MOE needed";CHAR$
1265 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN HRP%=0
1270 PRINT "HRP=";HRP%
1275 LPRINT "HRP=";HRP%
1280 '
1285 RCH%=1
1290 PRINT
1295 INPUT "Is the Reachability MOE needed";CHAR$
1300 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN RCH%=0
1305 PRINT "RCH=";RCH%
1310 LPRINT "RCH=";RCH%
1315 '
1320 MCMF%=1
1325 PRINT
1330 INPUT "Is the Maximum Flow MOE needed";CHAR$
1335 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN MCMF%=0
1340 PRINT "MCMF=";MCMF%
1345 LPRINT "MCMF=";MCMF%
1350 '
1355 NOLIP%=1
1360 PRINT
1365 INPUT "Is the Number of Link Independent Paths MOE needed";CHAR$
1370 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN NOLIP%=0
1375 PRINT "NOLIP=";NOLIP%
1380 LPRINT "NOLIP=";NOLIP%
1385 '
1390 SDRL%=1
1395 PRINT
1400 INPUT "Is the Source to Destination Reliability MOE needed";CHAR$
1405 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN SDRL%=0
1410 PRINT "SDRL=";SDRL%
1415 LPRINT "SDRL=";SDRL%
1420 '
1425 CON%=1
1430 PRINT
1435 INPUT "Is the Connectivity MOE needed";CHAR$
1440 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN CON%=0
1445 PRINT "CON=";CON%
1450 LPRINT "CON=";CON%
1455 '
1460 RT%=1
1465 PRINT
1470 INPUT "Is the Reliable Throughput needed";CHAR$
1475 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN RT%=0
1480 PRINT "RT=";RT%
1485 LPRINT "RT=";RT%
1490 '
1495 NR%=1
1500 PRINT
1505 INPUT "Is the Network Reliability MOE needed";CHAR$
1510 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN NR%=0
1515 PRINT "NR=";NR%
1520 LPRINT "NR=";NR%
1525 '
```

```
1530 '***************************************************************************
1535 '
1540 'DESCRIBE THE NETWORK
1545 '
1550 'Purpose of the routine:
1555 '
1560 '      This routine reads in the network.
1565 '
1570 'Variables used in the routine:
1575 '
1580 '      D%    the destination node
1585 '      NON%  the number of nodes in the network
1590 '      P     the probability of link or node survival
1595 '      S%    the source node, sometimes both the source and destination node
1600 '      W     the link or node weight
1605 '
1610 '      CONNECT()  the connection matrix for the network
1615 '      PROB()     the probability connection matrix
1620 '
1625 '***************************************************************************
1630 '
1635 PRINT
1640 LPRINT
1645 INPUT "How many nodes in the network";NON%
1650 PRINT "The number of nodes in the network is";NON%
1655 LPRINT "NON=";NON%
1660 '
1665 'Declare the arrays.
1670 OPTION BASE 1
1675 DIM ADJ%(NON%),DIR%(NON%),LABEL%(NON%),SCAN%(NON%),TMPFLO(NON%),WEIGHT(NON%
)
1680 DIM CONNECT(NON%,NON%),CUT(NON%,NON%),FLOW(NON%,NON%),HOLD(NON%,NON%),PROB(
NON%,NON%),SPM%(NON%,NON%),WGT(NON%,NON%)
1685 '
1690 PRINT
1695 LPRINT
1700 LPRINT "Node Input:"
1705 PRINT "     This is the input block for all nodes.  Input the node, its nod
e weight,"
1710 PRINT "and its probability of survival.  Input 0,0,0 when finished."
1715 INPUT "",S%,W,P
1720 IF (S%=0) GOTO 1755
1725   CONNECT(S%,S%)=W
1730   PROB(S%,S%)=P
1735   PRINT USING "##  ##.##  #.###";S%,W,P
1740   LPRINT USING "##  ##.##  #.###";S%,W,P
1745   GOTO 1715
1750 '
1755 PRINT
1760 LPRINT
1765 LPRINT "Bidirectional Link Input:"
1770 PRINT "     This is the input block for all bidirectional links.  Input the
 two nodes,"
1775 PRINT "the link weight, and the link probability of survival.  Input 0,0,0,
0 when"
```

```
1780 PRINT "finished."
1785 INPUT "",S%,D%,W,P
1790 IF (S%=0) GOTO 1835
1795   CONNECT(S%,D%)=W
1800   CONNECT(D%,S%)=W
1805   PROB(S%,D%)=P
1810   PROB(D%,S%)=P
1815   PRINT USING "##  ##  ##.##  #.###";S%,D%,W,P
1820   LPRINT USING "##  ##  ##.##  #.###";S%,D%,W,P
1825   GOTO 1785
1830 '
1835 PRINT
1840 LPRINT
1845 LPRINT "Directed Link Input:"
1850 PRINT "     This is the input block for all directed links.  Input the sour
ce node,"
1855 PRINT "the destination node, the link weight, and the link probability of s
urvival."
1860 PRINT "Input 0,0,0,0 when finished."
1865 INPUT "",S%,D%,W,P
1870 IF (S%=0) GOTO 1930
1875   CONNECT(S%,D%)=W
1880   PROB(S%,D%)=P
1885   PRINT USING "##  ##  ##.##  #.###";S%,D%,W,P
1890   LPRINT USING "##  ##  ##.##  #.###";S%,D%,W,P
1895   GOTO 1865
1900 '
1905 '***************************************************************************
1910 '
1915 'BEFORE-DEGRADATION OUTPUT
1920 '
1925 'Calculate and output the Shortest Delay Path MOE if the MOE was requested.
1930 A%=1
1935 IF (SDP%=1) THEN GOSUB 3955
1940 '
1945 'This loop saves the connection matrix for future use.
1950 FOR I%=1 TO NON%
1955   FOR J%=1 TO NON%
1960     HOLD(I%,J%)=CONNECT(I%,J%)
1965   NEXT J%
1970 NEXT I%
1975 '
1980 'Calculate and output the Highest Reliable Path MOE if the MOE was
1985 'requested.
1990 A%=2
1995 IF (HRP%=0) GOTO 2085
2000 '
2005 'This loop takes the negative logarithm of the probability matrix.
2010   FOR I%=1 TO NON%
2015     FOR J%=1 TO NON%
2020 '
2025 'If the link or node exists, then take the negative logarithm of its
2030 'probability of survival.  Otherwise, leave the probability zero.
2035       CONNECT(I%,J%)=0
2040       IF (PROB(I%,J%)<>0) THEN CONNECT(I%,J%)=-LOG(PROB(I%,J%))
```

```
2045      NEXT J%
2050    NEXT I%
2055 '
2060 'Call Dijkstra's Shortest Path Algorithm and pass the negative logarithm
2065 'probability matrix in place of the connection matrix.
2070    GOSUB 3955
2075 '
2080 'Calculate and output the Reachability MOE if the MOE was requested.
2085 A%=3
2090 IF (RCH%=0) GOTO 2210
2095 '
2100 'This loop sets all link weights to one and all node weights to zero.
2105    FOR I%=1 TO NON%
2110      FOR J%=1 TO NON%
2115 '
2120 'Set all weights to zero.
2125        CONNECT(I%,J%)=0
2130 '
2135 'If a link exists between the two nodes being examined, then set its weight
2140 'to one.
2145        IF (HOLD(I%,J%)<>0) THEN CONNECT(I%,J%)=1
2150 '
2155 'Set all node weights to zero.
2160        IF (I%=J%) THEN CONNECT(I%,J%)=0
2165      NEXT J%
2170    NEXT I%
2175 '
2180 'Call Dijkstra's Shortest Path First Algorithm and pass the binary
2185 'connection matrix (whose node weights are zero and link weights are one)
2190 'in place of the original connection matrix.
2195    GOSUB 3955
2200 '
2205 'Calculate and output the Maximum Throughput MOE if the MOE was requested.
2210 W%=1
2215 IF (MCMF%=0) GOTO 2310
2220 '
2225 'This loop inverts all link and node weights.
2230    FOR I%=1 TO NON%
2235      FOR J%=1 TO NON%
2240 '
2245 'If the link or node exists, then invert its weight.  Otherwise, leave the
2250 'weight zero.
2255        CONNECT(I%,J%)=0
2260        IF (HOLD(I%,J%)<>0) THEN CONNECT(I%,J%)=1/HOLD(I%,J%)
2265      NEXT J%
2270    NEXT I%
2275 '
2280 'Call Ford and Fulkerson's Min-Cut Max-Flow Algorithm and pass the inverted
2285 'connection matrix in place of the original connection matrix.
2290    GOSUB 5080
2295 '
2300 'Calculate and output the Number of Link Independent Paths MOE if the MOE
2305 'was requested.
2310 W%=2
2315 IF (NOLIP%=0) GOTO 2370
```

```
2320 '
2325 'Set all link and node weights to one.
2330   GOSUB 7415
2335 '
2340 'Call Ford and Fulkerson's Min-Cut Max-Flow Algorithm and pass the binary
2345 'connection matrix (whose node and link weights are one) in place of the
2350 'original connection matrix.
2355   GOSUB 5080
2360 '
2365 'Calculate and output the Reliability MOE if the MOE was requested.
2370 A%=4
2375 IF (SDRL%=0) GOTO 2460
2380 '
2385 'Set all link and node weights to one.
2390   GOSUB 7415
2395 '
2400 'Call the connectivity algorithm to determine the number of connected node
2405 'pairs.
2410   GOSUB 7575
2415   TOTAL%=CNP%
2420 '
2425 'Call the reliability algorithm and pass the binary connection matrix
2430 '(whose node and link weights are one) in place of the original connection
2435 'matrix.
2440   GOSUB 6505
2445 '
2450 'Calculate the before-degradation term for the Connectivity MOE if the MOE
2455 'was requested.
2460 IF (CON%=0) GOTO 2515
2465 '
2470 'Set all link and node weights to one.
2475   GOSUB 7415
2480 '
2485 'Call connectivity.
2490   GOSUB 7575
2495   CONDS%=CNP%
2500 '
2505 'Calculate the before-degradation term for the Relative Throughput MOE if
2510 'the MOE was requested.
2515   IF (RT%=0) GOTO 2545
2520     GOSUB 7730
2525     RTD=RTR
2530 '
2535 'Calculate the before-degradation term for the Network Reliability MOE if
2540 'the MOE was requested.
2545 IF (NR%=0) GOTO 2695
2550 '
2555 'Set all link and node weights to one.
2560   GOSUB 7415
2565   GOSUB 7875
2570   NRD=PDCT
2575 '
2580 '****************************************************************************
2585 '
2590 'EVENT BLOCK
```

```
2595 '
2600 'Purpose of the routine:
2605 '
2610 '     This routine reads in changes to the network.
2615 '
2620 'Variables used in the routine:
2625 '
2630 '     CHAR$  the user input
2635 '     D%     the destination node
2640 '     NON%   the number of nodes in the network
2645 '     P      the probability of link or node survival
2650 '     S%     the source node, sometimes the source and destination node
2655 '     W      the link or node weight
2660 '
2665 '     CONNECT()  the connection matrix for the network
2670 '     PROB()     the probability connection matrix
2675 '
2680 '***************************************************************************
2685 '
2690 'Determine whether the user wishes to continue.
2695 INPUT "Should the program continue (default yes)";CHAR$
2700 IF (CHAR$="no") OR (CHAR$="NO") OR (CHAR$="n") OR (CHAR$="N") THEN END
2705 '
2710 'Input the changes to the network.
2715 PRINT
2720 LPRINT
2725 PRINT "     This is the event block for the program.  At this point, the we
ights and"
2730 PRINT "probabilities of the nodes and links can be changed, or nodes and li
nks may be"
2735 PRINT "removed from the network entirely.  Input the degradated nodes and l
inks only."
2740 PRINT "All other nodes and links will remain the same."
2745 PRINT
2750 LPRINT "Node Changes:"
2755 PRINT "     This is the event block for the nodes.  Input the node, the new
 node"
2760 PRINT "weight, and the new probability of node survival.  Input 0,0,0 when
finished."
2765 INPUT "",S%,W,P
2770 IF (S%=0) GOTO 2865
2775   CONNECT(S%,S%)=W
2780   PROB(S%,S%)=P
2785 '
2790 'If either the node weight or the probability of node survival was set to
2795 'zero, then remove the node and its surrounding links from the network.
2800   IF (W<>0) AND (P<>0) GOTO 2845
2805     FOR I%=1 TO NON%
2810       CONNECT(I%,S%)=0
2815       CONNECT(S%,I%)=0
2820       PROB(I%,S%)=0
2825       PROB(S%,I%)=0
2830     NEXT I%
2835     W=0
2840     P=0
```

```
2845   PRINT USING "##  ##.##  #.###";S%,W,P
2850   LPRINT USING "##  ##.##  #.###";S%,W,P
2855   GOTO 2765
2860 '
2865 PRINT
2870 LPRINT
2875 LPRINT "Bidirectional Link Changes:"
2880 PRINT "     This is the event block for all bidirectional links.  Input the
 two nodes,"
2885 PRINT "the new link weight, and the new probability of link survival.  Inpu
t 0,0,0,0"
2890 PRINT "when finished."
2895 INPUT "",S%,D%,W,P
2900 IF (S%=0) GOTO 2995
2905   CONNECT(S%,D%)=W
2910   CONNECT(D%,S%)=W
2915   PROB(S%,D%)=P
2920   PROB(D%,S%)=P
2925 '
2930 'If either the link weight or the probability of link survival was set to
2935 'zero, then remove the link from the network.
2940   IF (W<>0) AND (P<>0) GOTO 2975
2945     CONNECT(S%,D%)=0
2950     CONNECT(D%,S%)=0
2955     PROB(S%,D%)=0
2960     PROB(D%,S%)=0
2965     W=0
2970     P=0
2975   PRINT USING "##  ##  ##.##  #.###";S%,D%,W,P
2980   LPRINT USING "##  ##  ##.##  #.###";S%,D%,W,P
2985   GOTO 2895
2990 '
2995 PRINT
3000 LPRINT
3005 LPRINT "Directed Link Changes:"
3010 PRINT "     This is the event block for all directed links.  Input the sour
ce node,"
3015 PRINT "the destination node, the new link weight, and the new probability o
f link"
3020 PRINT "survival.  Input 0,0,0,0 when finished."
3025 INPUT "",S%,D%,W,P
3030 IF (S%=0) GOTO 3130
3035   CONNECT(S%,D%)=W
3040   PROB(S%,D%)=P
3045 '
3050 'If either the link weight or the probability of link survival was set to
3055 'zero, then remove the link from the network.
3060   IF (W<>0) AND (P<>0) GOTO 3085
3065     CONNECT(S%,D%)=0
3070     PROB(S%,D%)=0
3075     W=0
3080     P=0
3085   PRINT USING "##  ##  ##.##  #.###";S%,D%,W,P
3090   LPRINT USING "##  ##  ##.##  #.###";S%,D%,W,P
3095   GOTO 3025
```

```
3100 '
3105 '**************************************************************************
3110 '
3115 'AFTER-DEGRADATION OUTPUT
3120 '
3125 'Calculate and output the Shortest Delay Path MOE if the MOE was requested.
3130 A%=1
3135 IF (SDP%=1) THEN GOSUB 3955
3140 '
3145 'This loop saves the connection matrix for future use.
3150 FOR I%=1 TO NON%
3155   FOR J%=1 TO NON%
3160     HOLD(I%,J%)=CONNECT(I%,J%)
3165   NEXT J%
3170 NEXT I%
3175 '
3180 'Calculate and output the Highest Reliable Path MOE if the MOE was
3185 'requested.
3190 A%=2
3195 IF (HRP%=0) GOTO 3285
3200 '
3205 'This loop takes the negative logarithm of the probability matrix.
3210   FOR I%=1 TO NON%
3215     FOR J%=1 TO NON%
3220 '
3225 'If the link or node exists, then take the negative logarithm of its
3230 'probability of survival.  Otherwise, leave the probability zero.
3235         CONNECT(I%,J%)=0
3240         IF (PROB(I%,J%)<>0) THEN CONNECT(I%,J%)=-LOG(PROB(I%,J%))
3245     NEXT J%
3250   NEXT I%
3255 '
3260 'Call Dijkstra's Shortest Path Algorithm and pass the negative logarithm
3265 'probability matrix in place of the connection matrix.
3270   GOSUB 3955
3275 '
3280 'Calculate and output the Reachability MOE if the MOE was requested.
3285 A%=3
3290 IF (RCH%=0) GOTO 3355
3295 '
3300 'This loop sets all link weights to one and all node weights to zero.
3305   FOR I%=1 TO NON%
3310     FOR J%=1 TO NON%
3315       CONNECT(I%,J%)=0
3320       IF (HOLD(I%,J%)<>0) THEN CONNECT(I%,J%)=1
3325       IF (I%=J%) THEN CONNECT(I%,J%)=0
3330     NEXT J%
3335   NEXT I%
3340   GOSUB 3955
3345 '
3350 'Calculate and output the Maximum Throughput MOE if the MOE was requested.
3355 W%=1
3360 IF (MCMF%=0) GOTO 3440
3365 '
3370 'This loop inverts all link and node weights.
```

```
3375   FOR I%=1 TO NON%
3380     FOR J%=1 TO NON%
3385       CONNECT(I%,J%)=0
3390       IF (HOLD(I%,J%)<>0) THEN CONNECT(I%,J%)=1/HOLD(I%,J%)
3395     NEXT J%
3400   NEXT I%
3405 '
3410 'Call Ford and Fulkerson's Min-Cut Max-Flow Algorithm and pass the inverted
3415 'connection matrix in place of the original connection matrix.
3420   GOSUB 5080
3425 '
3430 'Calculate and output the Number of Link Independent Paths MOE if the MOE
3435 'was requested.
3440 W%=2
3445 IF (NOLIP%=0) GOTO 3500
3450 '
3455 'Set all link and node weights to one.
3460   GOSUB 7415
3465 '
3470 'Call Ford and Fulkerson's Min-Cut Max-Flow Algorithm and pass the binary
3475 'connection matrix (whose node and link weights are one) in place of the
3480 'original connection matrix.
3485   GOSUB 5080
3490 '
3495 'Calculate and output the Reliability MOE if the MOE was requested.
3500 A%=4
3505 IF (SDRL%=0) GOTO 3565
3510 '
3515 'Set all link and node weights to one.
3520   GOSUB 7415
3525 '
3530 'Call the reliability algorithm and pass the binary connection matrix
3535 '(whose node and link weights are one) in place of the original connection
3540 'matrix.
3545   GOSUB 6570
3550 '
3555 'Calculate the after-degradation term for the Connectivity MOE if the MOE
3560 'was requested.
3565 IF (CON%=0) GOTO 3645
3570 '
3575 'Set all link and node weights to one.
3580   GOSUB 7415
3585 '
3590 'Call connectivity.
3595   GOSUB 7575
3600   CONN%=CNP%
3605   PRINT
3610   LPRINT
3615   CONTN=CONN%/COND%
3620   PRINT "The connectivity of the network is";CONTN
3625   LPRINT "The connectivity of the network is";CONTN
3630 '
3635 'Calculate the after-degradation term for the Relative Throughput MOE if
3640 'the MOE was requested.
3645   IF (RT%=0) GOTO 3690
```

```
3650     GOSUB 7730
3655   PRINT
3660   LPRINT
3665   PRINT "The reliable throughput of the network is";RTR/RTD
3670   LPRINT "The reliable throughput of the network is";RTR/RTD
3675 '
3680 'Calculate the after-degradation term for the Network Reliability MOE if
3685 'the MOE was requested.
3690 IF (NR%=0) GOTO 3740
3695 '
3700 'Set all link and node weights to one.
3705   GOSUB 7415
3710   GOSUB 7875
3715   PRINT
3720   LPRINT
3725   PRINT "The network reliability of the network is";PDCT/NRD
3730   LPRINT "The network reliability of the network is";PDCT/NRD
3735 '
3740 GOTO 2695
3745 '
3750 '***************************************************************************
3755 '
3760 'DIJKSTRA'S SHORTEST PATH FIRST ALGORITHM
3765 '
3770 'Purpose of the subroutine:
3775 '
3780 '     This subroutine finds the shortest delay path (or highest reliable
3785 'path, or shortest length path) between all sources and destinations.  The
3790 'subroutine also calculates the routing matrix for the network.
3795 '
3800 'Variables used in the subroutine:
3805 '
3810 '     A%    the Measure of Effectiveness criterion
3815 '                1  the shortest delay path
3820 '                2  the highest reliable path
3825 '                3  minimum number of links
3830 '                4  minimum number of links
3835 '     I%    the adjacent node, sometimes the source node
3840 '     J%    the destination node
3845 '     K%    the intermediate source node
3850 '     NON%  the number of nodes in the network
3855 '     MIN   the minimum weight along the shortest path
3860 '     S%    the source node
3865 '
3870 '     ADJ%()    the intermediate source nodes which form the path from
3875 '               source to destination
3880 '     LABEL%()  the label for the node
3885 '                 0  temporary
3890 '                 1  permanent
3895 '     WEIGHT()  the total weight along the path from source to destination
3900 '
3905 '     CONNECT()  the connection matrix for the network
3910 '     SPM%()     the routing directory for routing messages through the
3915 '                network
3920 '     WGT()      the total weight from source to destination
```

```
3925 '
3930 '****************************************************************************
3935 '
3940 'Initialization
3945 '
3950 'Choose the source node.
3955 FOR S%=1 TO NON%
3960 '
3965 'This loop initializes the arrays.
3970   FOR I%=1 TO NON%
3975 '
3980 'Set the weight to all nodes equal to infinity.
3985     WEIGHT(I%)=9.999999E+37
3990 '
3995 'Set the adjacent nodes to zero (meaning none found yet).
4000     ADJ%(I%)=0
4005 '
4010 'Make the weights to all nodes temporary (temporary=0, permanent=1).
4015     LABEL%(I%)=0
4020   NEXT I%
4025 '
4030 'Set the weight from the source node to itself equal to the weight at that
4035 'node.
4040   WEIGHT(S%)=CONNECT(S%,S%)
4045 '
4050 'Make the length from the source to the source permanent.  (In other words,
4055 'there is no shorter path from the source to the source.)
4060   LABEL%(S%)=1
4065 '
4070 'Make the intermediate source node the source node.
4075   K%=S%
4080 '
4085 '****************************************************************************
4090 '
4095 'Label Updating
4100 '
4105 'This statement checks to see if all the paths have been made permanent.
4110 'If they have, then go to the output block.  If not, continue.
4115   FOR J%=1 TO NON%
4120 '
4125 'This loop finds all the adjacent nodes which are connected to the
4130 'intermediate source node, and stores their weight from the original source
4135 'node to the new adjacent node.
4140     FOR I%=1 TO NON%
4145 '
4150 'If there is a link present, then the node being examined is an adjacent
4155 'node.
4160       IF (CONNECT(K%,I%)=0) GOTO 4250
4165 '
4170 'If the label on the adjacent node is temporary, then continue.
4175         IF (LABEL%(I%)=1) GOTO 4250
4180 '
4185 'If the weight from the original source node to the adjacent node being
4190 'examined is greater than or equal to the weight the adjacent node already
4195 'has, then this particular path being examined is not(!) the shortest path
```

```
4200 'from the original source node to the adjacent node being examined.
4205            IF (WEIGHT(K%)+CONNECT(K%,I%)+CONNECT(I%,I%)>=WEIGHT(I%)) GOTO 42
50
4210 'If at this point in the program, then the path being examined is(!) a
4215 'shorter path to that particular adjacent node from the original source
4220 'node.  Put the intermediate source node on the adjacent node's adjacent
4225 'list.
4230             ADJ%(I%)=K%
4235 '
4240 'Record the new weight in the weight array.
4245             WEIGHT(I%)=WEIGHT(K%)+CONNECT(K%,I%)+CONNECT(I%,I%)
4250    NEXT I%
4255 '
4260 '**************************************************************************
4265 '
4270 'Making a Label Permanent
4275 '
4280 'Set the minimum weight to infinity.
4285     MIN=9.999999E+37
4290 '
4295 'Set the intermediate source node to zero.
4300     K%=0
4305 '
4310 'This loop chooses the next intermediate source node by selecting the node
4315 'with the smallest weight from the original source node.
4320     FOR I%=1 TO NON%
4325 '
4330 'If the node has already been made permanent, then choose another because
4335 'it has already been an intermediate source node.  Otherwise, continue.
4340       IF (LABEL%(I%)<>0) GOTO 4405
4345 '
4350 'If the total weight from the original source node to the node being
4355 'examined is less than the minimum weight, then continue.
4360         IF (WEIGHT(I%)>=MIN) GOTO 4405
4365 '
4370 'If at this point in the program, then there is a new minimum path weight
4375 'and a new intermediate source node.  Set the minimum weight equal to the
4380 'newly found total weight.
4385           MIN=WEIGHT(I%)
4390 '
4395 'Make the node being examined the new intermediate source node.
4400           K%=I%
4405     NEXT I%
4410 'If the intermediate source node remained zero, then the nodes left
4415 'temporary cannot be reached from the original source node.
4420     IF (K%=0) GOTO 4480
4425 '
4430 'Make the new intermediate source node permanent.
4435     LABEL%(K%)=1
4440   NEXT J%
4445 '
4450 '**************************************************************************
4455 '
4460 'Path Block
4465 '
```

```
4470 'This loop calculates the routing directory for the network, and saves the
4475 'total weight from the source to the destination.
4480   FOR I%=1 TO NON%
4485 '
4490 'Save the total weight from the source to destination.
4495     WGT(S%,I%)=WEIGHT(I%)
4500 '
4505 'If no path exists between the source and destination being examined, then
4510 'enter a zero in the routing directory.
4515     IF (ADJ%(I%)<>0) GOTO 4545
4520       SPM%(S%,I%)=0
4525     GOTO 4605
4530 '
4535 'If the destination node is an adjacent node to the source, then enter the
4540 'destination node in the routing directory.
4545     IF (ADJ%(I%)<>S%) GOTO 4580
4550       SPM%(S%,I%)=I%
4555     GOTO 4605
4560 '
4565 'If the destination node is not adjacent to the source, then retrace the
4570 'path to determine the adjacent node to the source which lies along the
4575 'path.
4580     X%=I%
4585     Y%=X%
4590     X%=ADJ%(X%)
4595     IF (X%<>S%) GOTO 4585
4600     SPM%(S%,I%)=Y%
4605   NEXT I%
4610 NEXT S%
4615 '
4620 'If the call to Dijkstra's Shortest Path First Algotithm was made from the
4625 'reliability algorithm, then return.
4630 IF (A%=4) THEN RETURN
4635 '
4640 '*******************************************************************************
4645 '
4650 'Output Block
4655 '
4660 'Print the total path weight and the routing directory.
4665 PRINT
4670 LPRINT
4675 C$=","
4680 D$=")="
4685 FOR I%=1 TO NON%
4690   FOR J%=1 TO NON%
4695     IF (A%<>1) GOTO 4750
4700       B$="SPM("
4705       E$="    WGT("
4710       IF (WGT(I%,J%)>100000!) GOTO 4730
4715         PRINT USING "\  \##!##\ \##\       \##!##\ \##.##";B$,I%,C$,J%,D$,S
PM%(I%,J%),E$,I%,C$,J%,D$,WGT(I%,J%)
4720         LPRINT USING "\  \##!##\ \##\       \##!##\ \##.##";B$,I%,C$,J%,D$,
SPM%(I%,J%),E$,I%,C$,J%,D$,WGT(I%,J%)
4725         GOTO 4825
4730       F$=")= infinity"
```

```
4735      PRINT USING "\  \##!##\ \##\        \##!##\          \";B$,I%,C$,J%,D$,
SPM%(I%,J%),E$,I%,C$,J%,F$
4740      LPRINT USING "\  \##!##\ \##\        \##!##\          \";B$,I%,C$,J%,D$
,SPM%(I%,J%),E$,I%,C$,J%,F$
4745      GOTO 4825
4750    IF (A%<>2) GOTO 4780
4755      B$="HRPM("
4760      E$="   PROB("
4765      PRINT USING "\   \##!##\ \##\        \##!##\ \#.###";B$,I%,C$,J%,D$,SP
M%(I%,J%),E$,I%,C$,J%,D$,EXP(-WGT(I%,J%))
4770      LPRINT USING "\   \##!##\ \##\        \##!##\ \#.###";B$,I%,C$,J%,D$,S
PM%(I%,J%),E$,I%,C$,J%,D$,EXP(-WGT(I%,J%))
4775      GOTO 4825
4780    B$="LENM("
4785    E$="   LENGTH("
4790    IF (WGT(I%,J%)>100000!) GOTO 4810
4795      PRINT USING "\   \##!##\ \##\        \##!##\ \##";B$,I%,C$,J%,D$,SPM
%(I%,J%),E$,I%,C$,J%,D$,WGT(I%,J%)
4800      LPRINT USING "\   \##!##\ \##\        \##!##\ \##";B$,I%,C$,J%,D$,SP
M%(I%,J%),E$,I%,C$,J%,D$,WGT(I%,J%)
4805      GOTO 4825
4810    F$=")= infinity"
4815    PRINT USING "\   \##!##\ \##\        \##!##\          \";B$,I%,C$,J%,D$
,SPM%(I%,J%),E$,I%,C$,J%,F$
4820      LPRINT USING "\   \##!##\ \##\        \##!##\          \";B$,I%,C$,J%,D
$,SPM%(I%,J%),E$,I%,C$,J%,F$
4825   NEXT J%
4830 NEXT I%
4835 RETURN
4840 '
4845 '**************************************************************************
4850 '
4855 'FORD AND FULKERSON'S MIN-CUT MAX-FLOW ALGORITHM
4860 '
4865 'Purpose of the subroutine:
4870 '
4875 '     This subroutine finds the maximum throughput (or number of link
4880 'independent paths) between all sources and destinations.
4885 '
4890 'Variables used in the subroutine:
4895 '
4900 '     D%    the destination node
4905 '     I%    the intermediate source node, sometimes the source node
4910 '     J%    the intermediate deatination node, sometimes the destination
4915 '           node
4920 '     NON%  the number of nodes
4925 '     S%    the source node
4930 '     W%    the Measure of Effectiveness criterion
4935 '              1  maximum throughput
4940 '              2  link independent paths
4945 '     Z%    the pointer node (intermediate destination node when retracing
4950 '           the path from destination to source)
4955 '
4960 '     ADJ%()    the intermediate source nodes which form the path from
4965 '               source to destination
```

```
4970 '      DIR%()     the direction of flow
4975 '                       -1  flow leaving the intermediate destination node
4980 '                        0  neutral (no flow entering or leaving)
4985 '                        1  flow entering the intermediate destination node
4990 '      LABEL%()  a label on the node
4995 '                      0  unlabeled
5000 '                      1  labeled
5005 '      SCAN%()   a label on the node
5010 '                      0  unscanned
5015 '                      1  scanned
5020 '      TMPFLO()  the temporary flow being pushed along the path from source
5025 '                to destination
5030 '
5035 '      CONNECT()  the connection matrix for the network
5040 '      CUT()      the maximum flow between source and destination
5045 '      FLOW()     the augmented flow in the network
5050 '
5055 '*****************************************************************************
5060 '
5065 'Initialization
5070 '
5075 'This loop initializes the output array to zero.
5080 FOR I%=1 TO NON%
5085   FOR J%=1 TO NON%
5090     CUT(I%,J%)=0
5095   NEXT J%
5100 NEXT I%
5105 '
5110 'Choose the source node.
5115 FOR S%=1 TO NON%
5120 '
5125 'Choose the destination node.
5130   FOR D%=1 TO NON%
5135 '
5140 'If the source node and the destination node are one and the same, then
5145 'choose a new destination node.  Otherwise, continue.
5150     IF (S%=D%) GOTO 6160
5155 '
5160 'This loop initializes the flow along all links to zero.
5165       FOR I%=1 TO NON%
5170         FOR J%=1 TO NON%
5175           FLOW(I%,J%)=0
5180         NEXT J%
5185       NEXT I%
5190 '
5195 'This loop initializes the arrays.
5200       FOR I%=1 TO NON%
5205 '
5210 'Set the adjacent nodes to zero (meaning none found yet).
5215         ADJ%(I%)=0
5220 '
5225 'Set the direction of flow to neutral (meaning no flow into or out of the
5230 'node yet).
5235         DIR%(I%)=0
5240 '
```

```
5245 'Make all nodes unlabelled and unscanned.
5250         LABEL%(I%)=0
5255         SCAN%(I%)=0
5260 '
5265 'Set the temporary flow entering and leaving all nodes equal to infinity.
5270         TMPFLO(I%)=9.999999E+37
5275       NEXT I%
5280 '
5285 'Make the source node labelled and unscanned.
5290       LABEL%(S%)=1
5295 '
5300 '***************************************************************************
5305 '
5310 'Label Updating
5315 '
5320 'This outer loop is necessary because the search for the minimum cut is
5325 'very dependent on how the nodes are numbered.
5330       FOR K%=1 TO NON%
5335 '
5340 'This loop chooses an intermediate source node along a path from the source
5345 'node to the destination node.
5350         FOR I%=1 TO NON%
5355 '
5360 'If the node being examined is labelled and unscanned, then continue.
5365           IF (LABEL%(I%)=0) OR (SCAN%(I%)=1) GOTO 6045
5370 '
5375 'This loop chooses an intermediate destination node along a path from
5380 'source to the destination node.
5385             FOR J%=1 TO NON%
5390 '
5395 'If there is no link present, try another node.
5400               IF (CONNECT(I%,J%)=0) GOTO 6040
5405 '
5410 'If the adjacent node being examined is unlabelled, then the link lies
5415 'along a path from source to destination.
5420                 IF (LABEL%(J%)=1) GOTO 6040
5425 '
5430 'If there is flow from the intermediate destination node to the
5435 'intermediate source node, then continue.
5440                   IF (FLOW(J%,I%)<=0) GOTO 5585
5445 '
5450 'Put the intermediate source node on the intermediate destination node's
5455 'adjacent list.
5460                     ADJ%(J%)=I%
5465 '
5470 'Direct the flow out of the intermediate destination node.
5475                     DIR%(J%)=-1
5480 '
5485 'If the temporary flow from the intermediate destination node to the
5490 'intermediate source node is less than flow already along that link, then
5495 'push that amount of temporary flow along that link.  Otherwise, leave the
5500 'flow the same.
5505                     IF (TMPFLO(I%)>=FLOW(J%,I%)) GOTO 5520
5510                       TMPFLO(J%)=TMPFLO(I%)
5515                       GOTO 5535
```

```
5520                    TMPFLO(J%)=FLOW(J%,I%)
5525 '
5530 'Make the intermediate destination node labelled and unscanned.
5535                    LABEL%(J%)=1
5540 '
5545 'Make the intermediate source node labelled and scanned.
5550                    SCAN%(I%)=1
5555 '
5560 'If the capacity for flow through the intermediate destination node is less
5565 'than the capacity for flow through the adjacent link from the intermediate
5570 'source node to the intermediate destination node, then use the capacity
5575 'for flow through the intermediate destination node in place of the
5580 'capacity for flow through the link.
5585                 IF (CONNECT(J%,J%)>=CONNECT(I%,J%)) GOTO 5760
5590 '
5595 'If the capacity of the intermediate destination node is greater than the
5600 'flow along the adjacent link from the intermediate source node to the
5605 'intermediate destination node, then continue.
5610                    IF (CONNECT(J%,J%)<=FLOW(I%,J%)) GOTO 5915
5615 '
5620 'Put the intermediate source node on the intermediate destination node's
5625 'adjacent list.
5630                    ADJ%(J%)=I%
5635 '
5640 'Direct the flow into the intermediate destination node.
5645                    DIR%(J%)=1
5650 '
5655 'If the temporary flow from the intermediate source node to the
5660 'intermediate destination node is less than the amount of unused capacity
5665 'at the intermediate destination node, then push that amount of flow to the
5670 'intermediate destination node.  If the temporary flow is greater than the
5675 'unused capacity at the intermediate destination node, then only push as
5680 'much flow as the node can handle.
5685                    IF (TMPFLO(I%)>=CONNECT(J%,J%)-FLOW(I%,J%)) GOTO 5700
5690                      TMPFLO(J%)=TMPFLO(I%)
5695                      GOTO 5715
5700                    TMPFLO(J%)=CONNECT(J%,J%)-FLOW(I%,J%)
5705 '
5710 'Make the intermediate destination node labelled and unscanned.
5715                    LABEL%(J%)=1
5720 '
5725 'Make the intermediate source node labelled and scanned.
5730                    SCAN%(I%)=1
5735                    GOTO 5915
5740 '
5745 'If the flow from the intermediate source node to the intermediate
5750 'destination node is less than or equal to the capacity of the link, then
5755 'continue.
5760                 IF (CONNECT(I%,J%)<=FLOW(I%,J%)) GOTO 5915
5765 '
5770 'Put the intermediate source node on the intermediate destination node's
5775 'adjacent list.
5780                    ADJ%(J%)=I%
5785 '
5790 'Direct the flow into the intermediate destination node.
```

```
5795                    DIR%(J%)=1
5800 '
5805 'If the temporary flow from the intermediate source node to the
5810 'intermediate destination node is less than the amount of unused capacity
5815 'of the link, then push that amount of flow to the intermediate destination
5820 'node.  If the temporary flow is greater than the unused capacity of the
5825 'link, then only push as much flow as the link can handle.
5830                    IF (TMPFLO(I%)>=CONNECT(I%,J%)-FLOW(I%,J%)) GOTO 5845
5835                      TMPFLO(J%)=TMPFLO(I%)
5840                      GOTO 5860
5845                    TMPFLO(J%)=CONNECT(I%,J%)-FLOW(I%,J%)
5850 '
5855 'Make the intermediate destination node labelled and unscanned.
5860                    LABEL%(J%)=1
5865 '
5870 'Make the intermediate source node labelled and scanned.
5875                    SCAN%(I%)=1
5880 '
5885 '**************************************************************************
5890 '
5895 'Flow Augmentation
5900 '
5905 'If the destination node has been labelled, then augment the flow along the
5910 'path found.
5915                    IF (LABEL%(D%)=0) GOTO 6040
5920 '
5925 'Make the intermediate destination node the destination node.
5930                    Z%=D%
5935 '
5940 'If the flow is directed into the intermediate destination node, then
5945 'increase the flow along the link between the previous intermediate source
5950 'node and the intermediate destination node by the minimum flow along the
5955 'path being examined.
5960                    IF (DIR%(Z%)=1) THEN FLOW(ADJ%(Z%),Z%)=FLOW(ADJ%(Z%
),Z%)+TMPFLO(D%)
5965 '
5970 'If the flow is directed out of the intermediate destination node, then
5975 'decrease the flow along the link between the previous intermediate source
5980 'node and the intermediate destination node by the minimum flow along the
5985 'path being examined.
5990                    IF (DIR%(Z%)=-1) THEN FLOW(ADJ%(Z%),Z%)=FLOW(ADJ%(Z
%),Z%)-TMPFLO(D%)
5995 '
6000 'If the previous intermediate source node is the source node, then repeat
6005 'the process with the inproved flow calculated above.  If the previous
6010 'intermediate source node was not the source node, then make the previous
6015 'intermediate source node the intermediate destination node and continue
6020 'retracing the path to the source node.
6025                    IF (ADJ%(Z%)=S%) GOTO 5200
6030                      Z%=ADJ%(Z%)
6035                      GOTO 5960
6040            NEXT J%
6045          NEXT I%
6050        NEXT K%
6055 '
```

```
6060 'Since the flow entering the destination node is the same as the flow
6065 'across the minimum cut (if the destination node was not in the minimum
6070 'cut), then sum the flow entering the destination node to determine the
6075 'maximum throughput between source and destination.
6080        FOR I%=1 TO NON%
6085          CUT(S%,D%)=CUT(S%,D%)+FLOW(I%,D%)
6090        NEXT I%
6095 '
6100 'It is possible that the destination node was part of the minimum cut (and
6105 'this will happen if the node has a very low throughput). If this is so,
6110 'then the minimum cut has no meaning, and the destination node is the
6115 'minimum cut. It is also possible that the minimum cut is valid, but the
6120 'source node is incapable of delivering that amount of flow. In this case,
6125 'the source node is the minimum cut.
6130        IF (W%<>1) GOTO 6160
6135          IF (CUT(S%,D%)=0) GOTO 6160
6140            IF (CONNECT(S%,S%)>=CONNECT(D%,D%)) GOTO 6155
6145              IF (CONNECT(S%,S%)<CUT(S%,D%)) THEN CUT(S%,D%)=CONNECT(S%,S%)
6150              GOTO 6160
6155            IF (CONNECT(D%,D%)<CUT(S%,D%)) THEN CUT(S%,D%)=CONNECT(D%,D%)
6160   NEXT D%
6165 NEXT S%
6170 '
6175 '*************************************************************************
6180 '
6185 'Output Block
6190 '
6195 'Print the maximum throughput.
6200 PRINT
6205 LPRINT
6210 B$="CUT("
6215 C$=","
6220 D$=")="
6225 E$="IND("
6230 FOR I%=1 TO NON%
6235   FOR J%=1 TO NON%
6240     IF (W%=1) THEN PRINT USING "\  \##!##\ \##.#####";B$,I%,C$,J%,D$,CUT(I%
,J%)
6245     IF (W%=1) THEN LPRINT USING "\  \##!##\ \##.#####";B$,I%,C$,J%,D$,CUT(I
%,J%)
6250     IF (W%=2) THEN PRINT USING "\  \##!##\ \##";E$,I%,C$,J%,D$,CUT(I%,J%)
6255     IF (W%=2) THEN LPRINT USING "\  \##!##\ \##";E$,I%,C$,J%,D$,CUT(I%,J%)
6260   NEXT J%
6265 NEXT I%
6270 RETURN
6275 '
6280 '*************************************************************************
6285 '
6290 'RELIABILITY ALGORITHM
6295 '
6300 'Purpose of the subroutine:
6305 '
6310 '     This subroutine calculates the probability that at least one path
6315 'exists between a source and destination using state enumeration. The
6320 'subroutine also calculates the probability that the network is connected.
```

```
6325 '
6330 'Variables used in the subroutine:
6335 '
6340 '     A%      the Measure of Effectiveness criterion for Dijkstra's
6345 '             Shortest Path First Algorithm
6350 '                  1  shortest delay path
6355 '                  2  highest reliable path
6360 '                  3  minimum number of links
6365 '                  4  minimum number of links
6370 '     CONN    the connectivity of the network
6375 '     I%      the source node
6380 '     J%      the destination node
6385 '     K%      the node/link number
6390 '     M%      the number of up links and nodes
6395 '     NOLAN%  the number of links and nodes in the network
6400 '     NON%    the number of nodes in the network
6405 '     SUM%    the number of connected nodes per state
6410 '     TOTAL%  the number of connected nodes before degradation
6415 '
6420 '     DEST%()     the destination node
6425 '     PRB()       the probability of link or node survival
6430 '     SOURCE%()   the source node
6435 '     UPDOWN%()   the state of the link or node
6440 '                      0  down
6445 '                      1  up
6450 '
6455 '     CONNECT()   the connection matrix for the network
6460 '     PROB()      the probability connectiona matrix
6465 '     REL()
6470 '     SPM%()      the routing directory for the network
6475 '
6480 '****************************************************************************
6485 '
6490 'Initilization
6495 '
6500 'Set the number of links and nodes in the network to zero.
6505 NOLAN%=0
6510 '
6515 'This loop determines the number of links and nodes in the network to
6520 'determine the array sizes.
6525 FOR I%=1 TO NON%
6530   FOR J%=1 TO NON%
6535     IF (CONNECT(I%,J%)<>0) THEN NOLAN%=NOLAN%+1
6540   NEXT J%
6545 NEXT I%
6550 DIM DEST%(NOLAN%), PRB(NOLAN%), SOURCE%(NOLAN%), UPDOWN%(NOLAN%)
6555 DIM REL(NON%,NON%), SAV(NON%,NON%)
6560 '
6565 'Set the number of links and nodes in the network to zero.
6570 NOLAN%=0
6575 '
6580 'Set the node/link number to one.
6585 K%=1
6590 '
6595 'This loop numbers all nodes and links.
```

```
6600 FOR I%=1 TO NON%
6605   FOR J%=1 TO NON%
6610 '
6615 'If the link or node exists, then number it.
6620     IF (CONNECT(I%,J%)=0) GOTO 6685
6625 '
6630 'Store, for quick reference, the probability of survival, the source node,
6635 'and the destination node under its node/link number.
6640       PRB(K%)=PROB(I%,J%)
6645       SOURCE%(K%)=I%
6650       DEST%(K%)=J%
6655 '
6660 'Increment the node/link number.
6665       K%=K%+1
6670 '
6675 'Increment the number of links and nodes.
6680       NOLAN%=NOLAN%+1
6685   NEXT J%
6690 NEXT I%
6695 '
6700 'This loop saves the connection matrix for future use and sets the
6705 'reliability matrix to zero.
6710 FOR I%=1 TO NON%
6715   FOR J%=1 TO NON%
6720     SAV(I%,J%)=CONNECT(I%,J%)
6725     REL(I%,J%)=0
6730   NEXT J%
6735 NEXT I%
6740 '
6745 'Initialize the states of all the nodes and links to down.
6750 FOR I%=1 TO NOLAN%
6755   UPDOWN%(I%)=0
6760 NEXT I%
6765 '
6770 'Make the probability that the network is connected zero.
6775 POC=0
6780 '
6785 'Set the number of links and nodes to zero.
6790 M%=0
6795 '
6800 '****************************************************************************
6805 '
6810 'State Block
6815 '
6820 'Point the node/link to the first element.
6825 K%=1
6830 '
6835 'If the node or link is down, continue.
6840 IF (UPDOWN%(K%)=1) GOTO 7180
6845   M%=M%+1
6850   UPDOWN%(K%)=1
6855 '
6860 'Restore the connection matrix.
6865   FOR I%=1 TO NON%
6870     FOR J%=1 TO NON%
```

```
6875        CONNECT(I%,J%)=SAV(I%,J%)
6880      NEXT J%
6885    NEXT I%
6890 '
6895 'This loop removes all the down nodes and links from the network.
6900    FOR I%=1 TO NOLAN%
6905 '
6910 'If the node or link being examined is down, then remove it from the
6915 'network.
6920      IF (UPDOWN%(I%)=1) GOTO 6965
6925        CONNECT(SOURCE%(I%),DEST%(I%))=0
6930 '
6935 'If the node being examined is down, then remove all of its adjacent links.
6940        IF (SOURCE%(I%)<>DEST%(I%)) GOTO 6965
6945          FOR J%=1 TO NON%
6950            CONNECT(SOURCE%(I%),J%)=0
6955            CONNECT(J%,DEST%(I%))=0
6960          NEXT J%
6965    NEXT I%
6970 '
6975 'Call Dijkstra's Shortest Path First Algorithm to determine the routing
6980 'matrix for the network.  The routing matrix is just a form of the
6985 'reachibility matrix.
6990    GOSUB 3955
6995 '
7000 'Find the product of the probabilities of failures of the down elements and
7005 'the probabilities of successes of the up elements.
7010    T=1
7015    FOR I%=1 TO NOLAN%
7020      IF (UPDOWN%(I%)=0) THEN T=T*(1-PRB(I%))
7025      IF (UPDOWN%(I%)=1) THEN T=T*PRB(I%)
7030    NEXT I%
7035 '
7040 'If a path exists between the source and destination being examined, then
7045 'add in the probability to the total path reliability.
7050    FOR I%=1 TO NON%
7055      FOR J%=1 TO NON%
7060        IF (SPM%(I%,J%)<>0) THEN REL(I%,J%)=REL(I%,J%)+T
7065      NEXT J%
7070    NEXT I%
7075 '
7080 'Set the number of connected node pairs to zero.
7085    SUM%=0
7090 '
7095 'This loop totals the number of connected node pairs.
7100    FOR I%=1 TO NON%
7105      FOR J%=1 TO NON%
7110        IF (SPM%(I%,J%)<>0) THEN SUM%=SUM%+1
7115      NEXT J%
7120    NEXT I%
7125 '
7130 'Calculate the connectivity of the network.
7135    CONN=SUM%/TOTAL%
7140 '
7145 'If the network has as many connected node pairs as the original network,
7150 'then add in the probability to the network reliability.
7155   IF (CONN=1) THEN POC=POC+T
7160 '
7165 'If all links are up, then go to the Output Block.
7170   IF (M%=NOLAN%) GOTO 7230
7175   GOTO 6825
7180 M%=M%-1
7185 UPDOWN%(K%)=0
7190 K%=K%+1
7195 GOTO 6840
7200 '
7205 '****************************************************************************
7210 '
7215 'Output Block
7220 '
7225 'Print the
7230 PRINT
7235 LPRINT
7240 B$="REL("
7245 C$=","
7250 D$=")="
7255 FOR I%=1 TO NON%
7260   FOR J%=1 TO NON%
7265     PRINT USING "\  \##!##\ \#.#####";B$,I%,C$,J%,D$,REL(I%,J%)
7270     LPRINT USING "\  \##!##\ \#.#####";B$,I%,C$,J%,D$,REL(I%,J%)
7275   NEXT J%
7280 NEXT I%
7285 PRINT
7290 LPRINT
7295 E$="The probability the network is connected is "
7300 PRINT USING "\                                              \#.#####";E$,POC
7305 LPRINT USING "\                                             \#.#####";E$,POC
7310 RETURN
7315 '
7320 '****************************************************************************
7325 '
7330 'CHANGE THE CONNECTION MATRIX
7335 '
7340 'Purpose of the subroutine:
7345 '
7350 '     This subroutine calculates the binary connection matrix for the
7355 'network.
7360 '
7365 'Variables used in the subroutine:
7370 '
7375 '     NON%  the number of nodes in the network
7380 '
7385 '     CONNECT()  the connection matrix for the network
7390 '     HOLD()     the connection matrix for the network
7395 '
7400 '****************************************************************************
7405 '
7410 'This loop sets all link and node weights to one.
7415   FOR I%=1 TO NON%
7420     FOR J%=1 TO NON%
```

```
7425 '
7430 'If the link or node exists, then set its weight to one.  Otherwise, leave
7435 'the weight zero.
7440      CONNECT(I%,J%)=0
7445      IF (HOLD(I%,J%)<>0) THEN CONNECT(I%,J%)=1
7450    NEXT J%
7455  NEXT I%
7460 RETURN
7465 '
7470 '***************************************************************************
7475 '
7480 'CONNECTIVITY
7485 '
7490 'Purpose of the subroutine:
7495 '
7500 '     This subroutine determines the before-degradation and
7505 'after-degradation terms for the Connectivity MOE.
7510 '
7515 'Variables used in the subroutine:
7520 '
7525 '    CNP%  the number of connected node pairs
7530 '    NON%  the number of nodes in the network
7535 '
7540 '    SPM%()  the routing directory for routing messages
7545 '
7550 '***************************************************************************
7555 '
7560 'Call Dijkstra's Shortest Path First Algorithm to determine the routing
7565 'matrix for the network.  The routing matrix is just a form of the
7570 'reachability matrix.
7575 GOSUB 3955
7580 '
7585 'Set the number of connected node pairs to zero.
7590 CNP%=0
7595 '
7600 'This loop totals the number of connected node pairs.
7605 FOR I%=1 TO NON%
7610   FOR J%=1 TO NON%
7615     IF (SPM%(I%,J%)<>0) THEN CNP%=CNP%+1
7620   NEXT J%
7625 NEXT I%
7630 RETURN
7635 '
7640 '***************************************************************************
7645 '
7650 'RELIABLE THROUGHPUT
7655 '
7660 'Purpose of the subroutine:
7665 '
7670 '     This subroutine calculates the before-degradation and
7675 'after-degradation terms for the Relative Throughput MOE.
7680 '
7685 'Variables used in the subroutine:
7690 '
7695 '     NON%  the number of nodes in the network
```

```
7700 '     RTR  the sum of the link and node probabilities
7705 '
7710 '     PROB()  the probability connection matrix
7715 '
7720 '*****************************************************************************
7725 'Set the sum of the link and node probabilities to zero.
7730 RTR=0
7735 '
7740 'This loop sums the link and node probabilities.
7745 FOR I%=1 TO NON%
7750   FOR J%=1 TO NON%
7755     RTR=RTR+PROB(I%,J%)
7760   NEXT J%
7765 NEXT I%
7770 RETURN
7775 '
7780 '*****************************************************************************
7785 '
7790 'NETWORK RELIABILITY
7795 '
7800 'Purpose of the subroutine:
7805 '
7810 '     This subroutine calculates the before-degradation and
7815 'after-degradation terms for the Network Reliability MOE.
7820 '
7825 'Variables used in the subroutine:
7830 '
7835 '     CNP%  the number of connected node pairs
7840 '     PDCT  the product of the number of connected node pairs and the sum
7845 '           of the link and node probabilities
7850 '     RTR   the sum of the link and node probabilities
7855 '
7860 '*****************************************************************************
7865 '
7870 'Call the Connectivity subroutine.
7875 GOSUB 7575
7880 '
7885 'Call the Reliable Throughput subroutine.
7890 GOSUB 7730
7895 '
7900 'Calculate the Network Reliability term.
7905 PDCT=CNP%*RTR
7910 RETURN
```

## Appendix C: A GRAFTHY User's Manual

## C.0 Introduction to the User's Manual

Since GRAFTHY was intended to be user friendly for non-technical users and flexible enough for users to play "what if" games, the weights input to the program can either correspond to a relative delay on the links and nodes or to an actual bit rate on the links and nodes. For example, if there are two links in the computer network and the second link is twice as slow as the first link, then the first and second link weights could be 1 and 2 respectively. Weights of 2 and 4, 4 and 8, etc. would also work. Therefore the user is not required to know what the actual bit rates are but only the relative delay between them.

However, if the bit rates for the network under test are known, then the bit rates must be converted to the proper weights. To convert the bit rates on the nodes and links to weights, a standard should be chosen. Then each bit rate should be divided into the standard to determine their weights. This can be expressed as:

$$\mathrm{WEIGHT} = \frac{R}{\mathrm{STD}} \tag{2.1}$$

where

| | |
|---|---|
| WEIGHT | is the weight of the node or link in the model, |
| R | is the bit rate of the node or link in the network, |
| STD | is the standard bit rate chosen. |

For example, if there are three links in the computer network which have bit rates of 56 Kbps, 19.2 Kbps, and 9600 bps respectively, and the standard is 56 Kbps, then the link weights are 1, 2.917, and 5.714 respectively.

This user's manual is valid for both the FORTRAN GRAFTHY and the BASIC GRAFTHY. However, please note that the BASIC GRAFTHY requires a printer attached to the personal computer.

## C.1 Inputs to GRAFTHY

There are three different blocks of inputs to GRAFTHY. The first input block determines which MOE's are needed by the user. The second input block reads in the network to be examined. The third input block is called the event block, and its purpose is to read in changes to the network. All three of these blocks are discussed below.

### C.1.1 Requesting Different Measures of Effectiveness

Once the program has been loaded into memory and the command has been given to start execution, GRAFTHY will respond with:

The following is a list of Measures of Effectiveness (MOE's) which this algorithm will calculate, and all the MOE's will be calculated unless the algorithm is told otherwise. To prevent a MOE from being calculated, just type in a negative response when prompted. The default answer is yes (the MOE is needed).

Is the Shortest Delay Path MOE needed?

GRAFTHY will then pause, waiting for user input. If the user does not want the MOE calculated, then a response of n, N, no, or NO will prevent the MOE from being calculated. Any other input (including only a carriage return) will cause that MOE to be calculated for the network.

The above discussion also holds true for the next eight MOE questions that GRAFTHY will ask the user. They are:

Is the Highest Reliable Path MOE needed?

Is the Reachability MOE needed?

Is the Maximum Flow MOE needed?

Is the Number of Link Independent Paths MOE needed?

Is the Source to Destination Reliability MOE needed?

Is the Connectivity MOE needed?

Is the Reliable Throughput needed?

Is the Network Reliability MOE needed?

After the user responds to each MOE question, an echo print occurs reminding the user which MOE's were requested for that run.

## C.1.2 Network Input

After all the MOE questions have been answered, the network under test is then entered into GRAFTHY. The program will ask:

How many nodes in the network?

GRAFTHY will then pause, waiting for user input. The number of nodes in the network must be a non-negative integer value since it is not logical to consider either a negative number of nodes or a fractional number of nodes.

Once the number of nodes has been input to the program, an echo print will occur. GRAFTHY will then display:

This is the input block for all nodes. Input the node, its node weight, and its probability of survival. Input 0,0,0 when finished.

GRAFTHY will then pause, waiting for the user to input the node number, its weight, and its probability of survival. The node number will be an integer between one and the number of nodes in the network inclusive. The node weight will be a real value greater than zero corresponding to the delay at that node. In GRAFTHY, a zero weight does not correspond to an infinite bit rate (no delay), but to the link or node being inoperative. The probability of node survival will be a fraction between 0 and 0.999. Probabilities of exactly one are not allowed in GRAFTHY, but no loss of generality occurs since real-world elements are not error free.

After each user response, an echo print will occur. Once all nodes, their weights, and their probabilities of survival have been input to GRAFTHY, the user should type 0,0,0 to terminate the node input. GRAFTHY will then respond with:

This is the input block for all bidirectional links. Input the two nodes, the link weight, and the link probability of survival. Input 0,0,0,0 when finished.

Again, GRAFTHY will pause, waiting for user input. Since bidirectional links allow communication in either direction between the two nodes that the link connects, the order that the node numbers are input does not matter. For example, the link 1,2 is the same as the link 2,1. The rules for node number input, weight input, and probability of survival input are the same as those listed above.

After each user response, an echo print will occur. Once all the bidirectional links, their weights, and their probabilities of survival have been input to GRAFTHY, the user should type 0,0,0,0 to terminate the bidirectional link input. GRAFTHY will then display:

This is the input block for all directed links. Input the source node, the destination node, the link weight, and the link probability of survival. Input 0,0,0,0 when finished.

GRAFTHY will then pause, waiting for the user to input the source node number, the destination node number, the link weight, and the probability of link survival in that order. The rules for node number input, weight input, and probability of survival input are the same as those listed above.

After each user response, an echo print will occur. Once all the directed links, their weights, and their probabilities of survival have been input to GRAFTHY, the user should type 0,0,0,0 to terminate the directed link input.

## C.1.3 Entering Changes to the Network

After the program has calculated and output the before-degradation terms for all the MOE's requested, changes to the network may be entered into GRAFTHY if the user so desires. GRAFTHY will display:

```
Should the program continue (default yes)?
```

GRAFTHY will then pause, waiting for user input. If the user does not want to continue, then a response of n, N, no, or NO will terminate the program. Any other input (including only a carriage return) will cause the program to continue. If the user should terminate the program at the first event, the user should expect no network MOE's to be output since these are ratios of after-degradation terms to before-degradation terms.

If the user has chosen to continue the program, GRAFTHY will then display:

```
     This is the event block for the program. At this point, the weights and
probabilities of the nodes and links can be changed, or nodes and links may be
removed from the network entirely. Input the degradated nodes and links only.
All other nodes and links will remain the same.

     This is the event block for the nodes. Input the node, the new node
weight, and the new probability of node survival. Input 0,0,0 when finished.
```

GRAFTHY will then pause, waiting for the user to input changes to the network. Only changes to the links and nodes should be input to GRAFTHY. All other link and node weights and probabilities of survival will remain the same. The input to the event block behaves identically to the input to the input block and needs no further explanation.

The above discussion also holds true for the next two input events that GRAFTHY will ask the user. They are:

```
     This is the event block for all bidirectional links. Input the two nodes,
the new link weight, and the new probability of link survival. Input 0,0,0,0
when finished.

     This is the event block for all directed links. Input the source node,
the destination node, the new link weight, and the new probability of link
survival. Input 0,0,0,0 when finished.
```

Once all the changes have been input to the program, the after-degradation terms will be calculated and output for all the MOE's requested. The program will then return to the event block, allowing the user to make additional changes to the network or terminate ths program.

## C.2 Outpute from GRAFTHY

All outpute from GRAFTHY will be weights corresponding to the delay on the nodss or links. If the actual bit ratee for the nodes and links were unknown, and only the relative delays were known, then the output weighte are the weighte relative to the input wsights. If the bit ratee were known for the network under teet, then the output weights can be converted directly to bit rates for ths network.

Nine different output MOE's are poesible with GRAFTHY. They are: the shortest delay path outpute, the highest reliable path outpute, the reachability outputs, the maximum throughput outputs, the number of link independent path outputs, the reliability outputs, the connectivity outputs, the reliabile throughput outpute, and the nstwork reliability outpute. Theee GRAFTHY outputs are diecussed below.

Note that the firet pointer of each output array is the eource node, and the second pointer is the destination node. The outputs for the routing matrices are adjacent nodes to the eource nodee along paths to the destinations, depending on the criterion uesd (shortest delay path, higheet reliable path, etc.).

### C.2.1 Shortest Delay Path Outputs

The output from Dijketra'e Shortest Path First Algorithm will be a weight corresponding to the dslay between the given source and destination taking into account proceeeing delay at each of the nodes along the path. The SDP matrix is the routing matrix for the shortest delay path criterion. The WGT matrix gives the total weight from the given source to the given destination.

If the bit ratee for the nodee and links were known originally, then to convert the output weight to a delay in seconds, the meeeage length should be divided by the standard and multiplied by the shortest delay path weight. This can be expressed as

$$D = \frac{ML}{STD} * SDPW \qquad (3.1)$$

where

D is the delay from eource to destination in seconds,
ML is the message length in bits,
STD is the standard bit rate chosen in bps,
SDPW ie the shortest delay path weight.

For example, if weight 1 (the standard) correeponds to a bit rate of 56 Kbps, the meeeage length is 1 Kbits, and the ehorteet delay path weight is 7, then the delay from the source to destination is 0.125 eeconds.

## C.2.2 Highest Reliable Path Outputs

The HRPM matrix is the routing matrix for the highest reliable path criterion. The PROB matrix gives the higheet rsliable path probability from the given source to the given deetination taking into account the probability of node survival at each node along the path, including the source and deetination node.

## C.2.3 Reachability Outputs

The LENM matrix is the routing matrix for the minimum number of links criterion. The LENGTH matrix gives the minimum number of links that a message must traverse when traveling from the given source to the given destination.

## C.2.4 Maximum Throughput Outputs

The output from Ford and Fulkereon's Min-Cut Max-Flow Algorithm will be a weight corresponding to the throughput between the given source and destination node taking into account the processing delay at each of the nodes, including the source and the deetination nodee. The CUT matrix gives the maximum throughput betwsen the given eource and the given destination.

If the bit ratee for ths nodes and links were known originally, then to convert the output weight to a throughput in bits per second, the standard should be multiplied by a maximum-flow weight. In other worde,

$$T = STD * MFW \tag{3.2}$$

where

T is the maximum throughput between a given source and destination in bps,
STD is the standard chosen in bps,
MFW is the maximum-flow weight.

For example, if weight 1 (the standard) corresponds to a bit rate of 9600 bps, and the maximum-flow weight is 1.20, then the maximum throughput from source to destination is 11.52 Kbps.

## C.2.5 Number of Link Independent Path Outputs

Ths IND matrix gives the number of link independent paths that a message could traverse when traveling from the given source to the given destination.

## C.2.6 Reliability Outputs

The REL matrix gives the probability that at least one path exists between the given source and destination nodes. The probability that the network is connected gives the probability that every connected node pair before degradation is still connectsd after degradation.

## C.2.7 Connectivity Outputs

The connectivity of the network gives the number of communicating node pairs after degradation divided by the number of communicating node pairs befors degradation.

## C.2.8 Reliabls Throughput Outputs

The reliable throughput of the network gives the sum of the link and node probabilities after degradation divided by the sum of the link and node probabilities before degradation.

### C.2.9 Network Reliability Outputs

The network reliability gives the product of the connectivity of the network and the reliable throughput of the network.

### C.2.10 Example Outputs

Examples of the above-mentioned MOE outputs can be found in Appendix D.

## Appendix D: An Example GRAFTHY Run

The example network shown in Figure D.1 was used to produce the following GRAFTHY output. The example run serves a dual purpose: to show a typical GRAFTHY output and to provide a demonstration the user can execute to become familiar with GRAFTHY and its procedures.

Should the user want to duplicate the example run, then all the MOE's should be requested. The example network shown in Figure D.1 is a three-node network with one bidirectional link and two directed links. For the initial input to the program, the node inputs should be:

```
1, 1, 0.95 (follow each line of input with a carriage return)
2, 1, 0.95
3, 2, 0.9
0, 0, 0
```

The bidirectional link inputs should be:

```
1, 2, 3, 0.7
0, 0, 0, 0
```

The directed link inputs should be:

```
2, 3, 4, 0.6
3, 1, 6, 0.8
0, 0, 0, 0
```

For the first event, the node inputs should be:

```
1, 1.5, 0.95
2, 1.5, 0.95
3, 2.5, 0.85
0, 0, 0
```

The bidirectional link input should be:

```
0, 0, 0, 0
```

The directed link input should be:

```
2, 3, 0, 0
3, 1, 6, 0.7
0, 0, 0, 0
```

For the second event, ths node inputs should be:

```
1, 3, 0.8
2, 0, 0
3, 4, 0.7
0, 0, 0
```

The bidirectional link input should be:

```
0, 0, 0, 0
```

The directed link inputs should be:

```
3, 1, 6, 0.5
0, 0, 0, 0
```

The approximate times for calculating and printing each block of MOE's on the Z-100 desktop computer for the example network are: six seconds for SPM and WGT, seven seconds for HRPM and PROB, nine seconds for LENM and LENGTH, 27 seconds for CUT, 26 seconds for IND, six minutes and 28 seconds for REL and the probability that the network is connected, and nine seconds for the remaining MOE's.

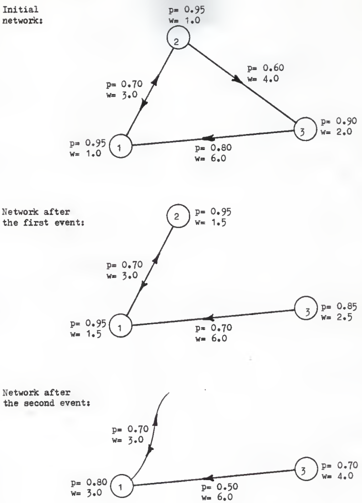

Figure D.1 An example network with two example degradation events.

```
SDP= 1
HRP= 1
RCH= 1
MCMF= 1
NOLIP= 1
SDRL= 1
CON= 1
RT= 1
NR= 1

The number of nodes in the network is 3

Node Input:
 1   1.00  0.950
 2   1.00  0.950
 3   2.00  0.900

Bidirectional Link Input:
 1   2   3.00  0.700

Directed Link Input:
 2   3   4.00  0.600
 3   1   6.00  0.800

SPM( 1, 1)=  0        WGT( 1, 1)=  1.00
SPM( 1, 2)=  2        WGT( 1, 2)=  5.00
SPM( 1, 3)=  2        WGT( 1, 3)= 11.00
SPM( 2, 1)=  1        WGT( 2, 1)=  5.00
SPM( 2, 2)=  0        WGT( 2, 2)=  1.00
SPM( 2, 3)=  3        WGT( 2, 3)=  7.00
SPM( 3, 1)=  1        WGT( 3, 1)=  9.00
SPM( 3, 2)=  1        WGT( 3, 2)= 13.00
SPM( 3, 3)=  0        WGT( 3, 3)=  2.00

HRPM( 1, 1)=  0       PROB( 1, 1)= 0.950
HRPM( 1, 2)=  2       PROB( 1, 2)= 0.632
HRPM( 1, 3)=  2       PROB( 1, 3)= 0.341
HRPM( 2, 1)=  1       PROB( 2, 1)= 0.632
HRPM( 2, 2)=  0       PROB( 2, 2)= 0.950
HRPM( 2, 3)=  3       PROB( 2, 3)= 0.513
HRPM( 3, 1)=  1       PROB( 3, 1)= 0.684
HRPM( 3, 2)=  1       PROB( 3, 2)= 0.455
HRPM( 3, 3)=  0       PROB( 3, 3)= 0.900

LENM( 1, 1)=  0       LENGTH( 1, 1)=  0
LENM( 1, 2)=  2       LENGTH( 1, 2)=  1
LENM( 1, 3)=  2       LENGTH( 1, 3)=  2
LENM( 2, 1)=  1       LENGTH( 2, 1)=  1
LENM( 2, 2)=  0       LENGTH( 2, 2)=  0
LENM( 2, 3)=  3       LENGTH( 2, 3)=  1
LENM( 3, 1)=  1       LENGTH( 3, 1)=  1
LENM( 3, 2)=  1       LENGTH( 3, 2)=  2
LENM( 3, 3)=  0       LENGTH( 3, 3)=  0
```

```
CUT( 1, 1)=  0.00000
CUT( 1, 2)=  0.33333
CUT( 1, 3)=  0.25000
CUT( 2, 1)=  0.50000
CUT( 2, 2)=  0.00000
CUT( 2, 3)=  0.25000
CUT( 3, 1)=  0.16667
CUT( 3, 2)=  0.16667
CUT( 3, 3)=  0.00000

IND( 1, 1)=  0
IND( 1, 2)=  1
IND( 1, 3)=  1
IND( 2, 1)=  2
IND( 2, 2)=  0
IND( 2, 3)=  1
IND( 3, 1)=  1
IND( 3, 2)=  1
IND( 3, 3)=  0

REL( 1, 1)= 0.00000
REL( 1, 2)= 0.63175
REL( 1, 3)= 0.34114
REL( 2, 1)= 0.74871
REL( 2, 2)= 0.00000
REL( 2, 3)= 0.51300
REL( 3, 1)= 0.68400
REL( 3, 2)= 0.45486
REL( 3, 3)= 0.00000

The probability the network is connected is 0.27292

Node Changes:
 1   1.50  0.950
 2   1.50  0.950
 3   2.50  0.850

Bidirectional Link Changes:

Directed Link Changes:
 2   3   0.00  0.000
 3   1   6.00  0.700

SPM( 1, 1)=  0      WGT( 1, 1)=  1.50
SPM( 1, 2)=  2      WGT( 1, 2)=  4.00
SPM( 1, 3)=  0      WGT( 1, 3)= infinity
SPM( 2, 1)=  1      WGT( 2, 1)=  4.00
SPM( 2, 2)=  0      WGT( 2, 2)=  1.50
SPM( 2, 3)=  0      WGT( 2, 3)= infinity
SPM( 3, 1)=  1      WGT( 3, 1)= 10.00
SPM( 3, 2)=  1      WGT( 3, 2)= 12.50
SPM( 3, 3)=  0      WGT( 3, 3)=  2.50
```

```
HRPM( 1, 1)=  0    PROB( 1, 1)= 0.950
HRPM( 1, 2)=  2    PROB( 1, 2)= 0.632
HRPM( 1, 3)=  0    PROB( 1, 3)= 0.000
HRPM( 2, 1)=  1    PROB( 2, 1)= 0.632
HRPM( 2, 2)=  0    PROB( 2, 2)= 0.950
HRPM( 2, 3)=  0    PROB( 2, 3)= 0.000
HRPM( 3, 1)=  1    PROB( 3, 1)= 0.565
HRPM( 3, 2)=  1    PROB( 3, 2)= 0.376
HRPM( 3, 3)=  0    PROB( 3, 3)= 0.850

LENM( 1, 1)=  0    LENGTH( 1, 1)=  0
LENM( 1, 2)=  2    LENGTH( 1, 2)=  1
LENM( 1, 3)=  0    LENGTH( 1, 3)= infinity
LENM( 2, 1)=  1    LENGTH( 2, 1)=  1
LENM( 2, 2)=  0    LENGTH( 2, 2)=  0
LENM( 2, 3)=  0    LENGTH( 2, 3)= infinity
LENM( 3, 1)=  1    LENGTH( 3, 1)=  1
LENM( 3, 2)=  1    LENGTH( 3, 2)=  2
LENM( 3, 3)=  0    LENGTH( 3, 3)=  0

CUT( 1, 1)=  0.00000
CUT( 1, 2)=  0.66667
CUT( 1, 3)=  0.00000
CUT( 2, 1)=  0.66667
CUT( 2, 2)=  0.00000
CUT( 2, 3)=  0.00000
CUT( 3, 1)=  0.16667
CUT( 3, 2)=  0.16667
CUT( 3, 3)=  0.00000

IND( 1, 1)=  0
IND( 1, 2)=  1
IND( 1, 3)=  0
IND( 2, 1)=  1
IND( 2, 2)=  0
IND( 2, 3)=  0
IND( 3, 1)=  1
IND( 3, 2)=  1
IND( 3, 3)=  0

REL( 1, 1)= 0.00000
REL( 1, 2)= 0.63175
REL( 1, 3)= 0.00000
REL( 2, 1)= 0.63175
REL( 2, 2)= 0.00000
REL( 2, 3)= 0.00000
REL( 3, 1)= 0.56525
REL( 3, 2)= 0.37589
REL( 3, 3)= 0.00000

The probability the network is connected is 0.00000

The connectivity of the network is .6666667
The reliable throughput of the network is .8660713

The network reliability of the network is .5773809

Node Changes:
 1   3.00  0.800
 2   0.00  0.000
 3   4.00  0.700

Bidirectional Link Changes:

Directed Link Changes:
 3   1   6.00  0.500

SPM( 1, 1)=  0     WGT( 1, 1)=  3.00
SPM( 1, 2)=  0     WGT( 1, 2)= infinity
SPM( 1, 3)=  0     WGT( 1, 3)= infinity
SPM( 2, 1)=  0     WGT( 2, 1)= infinity
SPM( 2, 2)=  0     WGT( 2, 2)=  0.00
SPM( 2, 3)=  0     WGT( 2, 3)= infinity
SPM( 3, 1)=  1     WGT( 3, 1)= 13.00
SPM( 3, 2)=  0     WGT( 3, 2)= infinity
SPM( 3, 3)=  0     WGT( 3, 3)=  4.00

HRPM( 1, 1)=  0    PROB( 1, 1)= 0.800
HRPM( 1, 2)=  0    PROB( 1, 2)= 0.000
HRPM( 1, 3)=  0    PROB( 1, 3)= 0.000
HRPM( 2, 1)=  0    PROB( 2, 1)= 0.000
HRPM( 2, 2)=  0    PROB( 2, 2)= 0.000
HRPM( 2, 3)=  0    PROB( 2, 3)= 0.000
HRPM( 3, 1)=  1    PROB( 3, 1)= 0.280
HRPM( 3, 2)=  0    PROB( 3, 2)= 0.000
HRPM( 3, 3)=  0    PROB( 3, 3)= 0.700

LENM( 1, 1)=  0    LENGTH( 1, 1)=  0
LENM( 1, 2)=  0    LENGTH( 1, 2)= infinity
LENM( 1, 3)=  0    LENGTH( 1, 3)= infinity
LENM( 2, 1)=  0    LENGTH( 2, 1)= infinity
LENM( 2, 2)=  0    LENGTH( 2, 2)=  0
LENM( 2, 3)=  0    LENGTH( 2, 3)= infinity
LENM( 3, 1)=  1    LENGTH( 3, 1)=  1
LENM( 3, 2)=  0    LENGTH( 3, 2)= infinity
LENM( 3, 3)=  0    LENGTH( 3, 3)=  0

CUT( 1, 1)=  0.00000
CUT( 1, 2)=  0.00000
CUT( 1, 3)=  0.00000
CUT( 2, 1)=  0.00000
CUT( 2, 2)=  0.00000
CUT( 2, 3)=  0.00000
CUT( 3, 1)=  0.16667
CUT( 3, 2)=  0.00000
CUT( 3, 3)=  0.00000
```

```
IND( 1, 1)=  0
IND( 1, 2)=  0
IND( 1, 3)=  0
IND( 2, 1)=  0
IND( 2, 2)=  0
IND( 2, 3)=  0
IND( 3, 1)=  1
IND( 3, 2)=  0
IND( 3, 3)=  0

REL( 1, 1)= 0.00000
REL( 1, 2)= 0.00000
REL( 1, 3)= 0.00000
REL( 2, 1)= 0.00000
REL( 2, 2)= 0.00000
REL( 2, 3)= 0.00000
REL( 3, 1)= 0.28000
REL( 3, 2)= 0.00000
REL( 3, 3)= 0.00000

The probability the network is connected is 0.00000

The connectivity of the network is .1666667

The reliable throughput of the network is .3571428

The network reliability of the network is 5.952381E-02
```

# A COMPUTER NETWORK SIMULATION UTILIZING GRAPH THEORY TO CALCULATE MEASURES OF EFFECTIVENESS

by

RUSSELL DEAN THOMAS

B.S., Kansas State University, 1983

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCES

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

## ABSTRACT

This paper contains a description of a computer network simulation program which utilizes graph theory to calculate the following Measures of Effectiveness (MOE's): Shortest Delay Path, Highest Reliable Path, Reachability, Maximum Throughput, Number of Link Independent Paths, Reliability, Connectivity, and two independently developed MOE's, Reliable Throughput and Network Reliability.