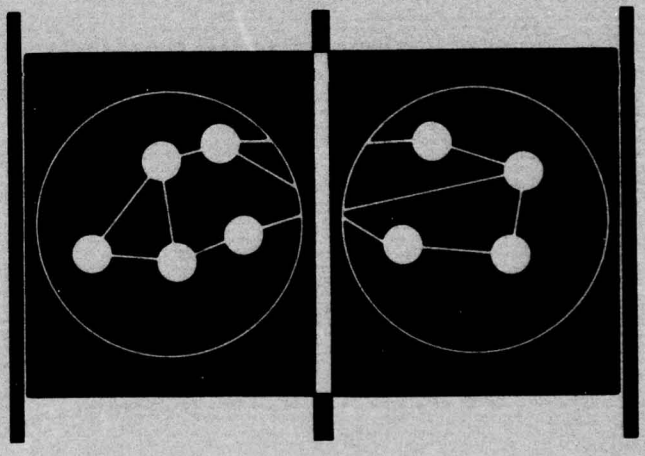
12 B.S.

Local, Regional and Large Scale Integrated Networks

Sixth Semiannual Technical Report

VOLUME 5

COMPUTATIONAL TECHNIQUES



nac

NETWORK ANALYSIS CORPORATION

(9)

Sixth Semiannual Technical Report, no. 6,

(11) February 1976

(12) 132 p.

For the Project

(6)

Local, Regional and Large Scale Integrated Networks.

VOLUME 5.

COMPUTATIONAL TECHNIQUES.

(10)

Principal Investigator: Howard Frank

Co-principal Investigators: Israel Gitman
Richard Van Slyke

Contractor

NETWORK ANALYSIS CORPORATION
Beechwood, Old Tappan Road
Glen Cove, NY 11542
(516) 671-9580

ARPA Order No. 2286

Contract No. DAHC 15-73-C-0135

(15)

Effective Date: 13 October 1972
Expiration Date: 30 June 1977

Sponsored by

Advanced Research Projects Agency
Department of Defense

1473

389161 AB

SIXTH SEMIANNUAL TECHNICAL REPORT

TABLE OF CONTENTS

VOLUME 4
LOCAL AND REGIONAL COMMUNICATION NETWORKS - TECHNOLOGIES AND ARCHITECTURES

VOLUME 5
COMPUTATIONAL TECHNIQUES

SUMMARY OF VOLUME 5

TECHNICAL OBJECTIVES

- Combine quick response of time-sharing system with high computational throughput of batch system.

- Develop efficient methods for the exact reliability analysis of networks with respect to a variety of network reliability measures.

GENERAL METHODOLOGY

- Develop a system for the automatic submission, monitoring, and output retrieval of batch jobs on a large scale scientific computer from an easy to use time-sharing computer using a computer network.

- Use modern techniques of computer science including backtracking and depth first search to develop exact network reliability analysis algorithms that would take advantage of special structure without human intervention.

TECHNICAL RESULTS

- NAC and a group at MIT-DMS headed by Al Vezza developed, tested, evaluated, and applied a system for automatic submission, monitoring, and output retrieval using the resources of the UCLA-CCN IBM 360-91, one of several TENEX sites, and the MIT-DMS PDP-10 on the ARPANET.

- A new class of analytic techniques called probabilistic backtracking was developed and applied to network reliability analysis.

- Exact network reliability analysis algorithms were developed for determining the probability that two specified nodes are connected, the probability that the network is connected, and the expected number of node pairs communicating. These algorithms were developed for nodes only failing, links only failing, and both nodes and links failing.

- Developed a special purpose algorithm for the reliability analysis of acyclic networks for applications to Packet Radio Networks.

- Applied the technique of palm tree generation by depth first search to network reliability analysis to automatically exploit the special structure of realistic communication networks.

- Used algorithms to solve problems of up to 34 failing elements (nodes and links) in less than one minute of PDP-10 CPU time.

DEPARTMENT OF DEFENSE IMPLICATIONS

The importance of reliable communication for the Department of Defense is obvious. Fortunately, the theory and techniques of the reliability of individual elements (channels, switches, etc.) of communications systems is well developed. The studies of this volume address the highly important, difficult, and little understood area of how to efficiently analyze the reliability of an entire communication network as related to the reliability of its elements. Since Department of Defense networks are among the largest in the world, efficient network-wide reliability analysis is particularly important to the military.

IMPLICATIONS FOR FURTHER RESEARCH

Further research is required to analyze:

- Simplex networks (up to now all networks were assumed full duplex).
- Networks with correlated element failure probabilities.
- Special methods for dealing exactly with the reliability of very large networks (e.g., decomposition, partitioning).

TABLE OF CONTENTS

ABSTRACT

PRECEDING PAGE BLANK-NOT FILMED

TABLE OF CONTENTS
FIGURES

VOLUME 5

Chapter 1

SYSTEM FOR AUTOMATIC REMOTE JOB ENTRY

1. SYSTEM FOR AUTOMATIC REMOTE JOB ENTRY

1.1 INTRODUCTION

This chapter describes an effort by Network Analysis Corporation (NAC) and a group at MIT-DMS headed by Al Vezza to develop, test, and evaluate a system for automatic submission of batch jobs and retrieval of output using the resources of the UCLA-CCN IBM/91, TENEX sites, and the MIT-DMS PDP-10 on the ARPA Computer Network ARPANET. The system is designed to provide an interface between batch computers and time-sharing resources on the Network.

1.2 SYSTEM CONSIDERATIONS

The interactive analysis and design of networks requires timely responses from the execution of a variety of network analysis programs. Such programs can consume vast amounts of CPU time, particularly when simulation techniques are used. Although the time-sharing resources for the interactive procedure are appropriate for short analysis programs, the turn around time for programs requiring large amounts of CPU time is far too long for an effective interactive tool. To achieve reasonable turn around time for such programs requires use of a batch computer, and for an effective interactive tool, the operations of connecting to the batch computer, signing the user on, submitting the job and retrieving the output, must be simple and automatic. Previously, procedures for use of batch computers on the ARPANET, as described in the next section, have not satisfied these requirements. We describe below a new procedure which does satisfy the requirements, and which has been effectively used for batch execution in the interactive design process.

1.3 PAST CONFIGURATION

The past procedure for batch programming on the network by users at NAC was:

1. A job file (IBM job control statements and data) was generated at a TENEX site by a user program or text editor. Before submitting the job to UCLA, it was first necessary for the user to read the file into TECO (a text editor) and write it out again. This procedure removed any null characters which could cause read checks on the RJS card reader.

2. The user ran a TENEX subsystem ("(SUBSYS")RJS.SAV) which handles the handshaking with the UCLA-CCN 91. The handshaking includes TELNET connections (to sign the user on to the 91 and report the status of active jobs) and FTP connections (to submit jobs and retrieve output.)

3. If the remote job service of the 91 was responding, the user submited the job by "sending" the job file to the 91. At this point the user has the option to wait for the output to be returned by remaining in this program or to quit this program and do other things. There are commands available to the user to check the status of the job and to have the system notify the user when output is ready to be returned.

4. If the output was not returned by the time the user quit RJS.SAV, the user had to run RJS again to check on the status of the job until the output is available. Depending on the load of the 91, it often took several such requests before that could occur. Another possibility was that the 91 was down before the job was run or went down after the job was submitted and before the output could be retrieved. In this case, the user may turn to other resources rather than going to the trouble of checking the job's status.

5. When the output was finally returned, the user had to use a text editor to view the results and/or list the results on a hardcopy device. When this was done another analysis could be done based on this data.

For a single batch job, the user might tolerate such a procedure. However, in an interactive network design system, the user is interested in a considerable number of possible network configurations and may very well want to run an analysis on a batch computer for many of them. If the user must constantly check the batch computer for job status, much of the time for design has been wasted. With the system described below, the user will make far better use of time and the resources of the ARPANET.

1.4 PRESENT SYSTEM

The present system works as follows:

1. A job file is generated by a user program or a text editor on a time-sharing TENEX computer. This file is complete with job control statements needed to run the job at UCLA and data needed by the program.

2. A special routine, CCNR, is invoked by the user (or the user's program) which performs the following functions:

   a. Receives from the user a name which will identify the information for this batch job. Let 'xxxxx' be such a job identifier.

   b. If invoked directly by the user, CCNR asks for the name of the file which should be submitted to UCLA; else the file name is passed as an argument by the calling program. The routine then reads in the file, removes the null characters, and writes the results on a special file called "xxxxx.FILE". It is this file which will be read by MIT-DMS and submitted to UCLA.

c. The routine generates a short command file which is then FTP'd to a specific file at MIT-DMS. This command file contains the user's login identifier for UCLA, the name of the file to be submitted ("xxxxx.FILE"), where the output will be returned ("xxxxx.PRINT-OUTPUT") and where any interaction between MIT and UCLA will be returned ("xxxxx.FILE-SCRIPT").

d. The routine checks for a successful transfer of the command file, notifies the user and then quits.

3. A DEMON at MIT is activated whenever a command file is sent to the specified file name (see c. above). This DEMON does the necessary bookkeeping at MIT to remember who sent it, what file is to be sent to the 91 and where the output is to be returned. The DEMON then attempts an RJE connection to UCLA, sign the user on, and submit the job. If it is unable to successfully submit the job, it deactivates itself. At a later point, it awakes itself to try again. When successful, it then periodically pokes UCLA to check if the output from the job is available. If the output is available, it fetches it from UCLA and transfers it to the output file designated by the user.

4. The user in the meantime is working independently at the TENEX site. When the output is returned, the user can check in his directory for the PRINT-OUTPUT file or optionally the system can detect it and notifies the user who can use this data for the next analysis or compare the results of previous runs.

1.5 APPLICATION

This system has been implemented on NAC's Network Editor, an interactive network design program. A major part of the program is the Network Reliability Analyzer which studies the reliability of the network by simulation. If the number of samples is small (say 100), the analyzer can run with a reasonable amount of CPU time for a network the size of the ARPANET. With such a small number of samples, the simulation results are not satisfactory. However, for better results the number of samples must be increased tenfold (to 1000), increasing the CPU time to such an extent that it can no longer be used interactively. In this case, the user submits the job to UCLA by simply entering the user's name and the job identifier. The program determines the amount of CPU time required and writes out a file with the necessary IBM job control language and data for submission to the IBM 360-91 using CCNR.

1.6 SYSTEM EXPANSION

The system can be simplified by eliminating the "middle-computer" (MIT's PDP-10) and moving the DEMON and data base to the computer on which the user is running. This would eliminate:

1. The additional operations of interacting with the "middle-computer", and

2. The dependence on that computer for the system to run.

A long range expansion would be to have the system test the load on each batch computer where the user could run the job. This would let the system decide on which site to run the job, and thus decrease turn around time.

Although this effort used the resources of the PDP-10 computer and the IBM model 91, the design can be implemented on other machines to give users the ability to put the resources available to them on the network to the best use.

1.7 CONCLUSION

The System for Automatic Remote Job Entry is a valuable asset to NAC's interactive network design and analysis development. It not only gives users simple and quick access to batch computing, but also increases the power and capability of the interactive system.

VOLUME 5

Chapter 2

BACKTRACKING ALGORITHMS FOR NETWORK RELIABILITY ANALYSIS

2. BACKTRACKING ALGORITHMS FOR NETWORK RELIABILITY ANALYSIS

2.1 INTRODUCTION

Backtracking algorithms are very useful in solving a variety of network-related problems. They provide a framework for efficient manipulation of data with relatively small storage requirements. For example, [HOPCROFT, 1973] gives backtracking algorithms for partitioning a graph into connected components, biconnected components and simple paths, and [READ, 1975] gives backtracking algorithms for listing cycles, paths and spanning trees of a graph. We have devised backtracking algorithms for determining certain reliability measures of a network. The algorithms are useful in analyzing the reliability of many data communications networks.

The reliability measures considered in this chapter are the probability that all nodes can communicate, the probability that all operative nodes can communicate, and the expected fraction of node pairs communicating. Networks are considered in which both nodes and arcs can fail. Exact answers can be obtained; in addition, by using a truncation procedure, approximate answers can be obtained in less time. In the third chapter of this volume, similar techniques are applied to the reliability analysis of acyclic networks. Also, in the third chapter, time saving generalizations of the algorithms in this chapter are given. Finally, in Volume 2 of this report, the reliability techniques of this chapter and the next are applied to the reliability analysis of Packet Radio networks.

There are basically two approaches to network reliability analysis: simulation and analysis. All known analytic methods for network reliability analysis have worst case computation time which grows exponentially in the size of the network considered. Our backtrack methods are analytic methods and are not exceptions to this trend. Hence, they are not recommended for large networks. However, results in [ROSENTHAL, 1974] indicate that network reliability analysis is intrinsically very different. Simulation methods for which computation time grows only slightly faster than

linear with network size have been described in the literature. Simulation techniques are suitable for large networks and are generally more flexible than analytic methods. However, they have the disadvantage that they only give approximate answers, and when a high degree of accuracy is necessary, the running time can grow quite large.

Analytic methods use basic probabilistic laws to reduce or decompose the problem. These methods can be roughly classified as either enumerative or reduction. Enumerative methods enumerate a set of probabilistic events which are mutually exclusive and collectively exhaustive with respect to the measure in question. Our algorithms are examples of enumerative algorithms. Reduction algorithms collapse two or more network components into one network component. The simplest example of network reduction is collapsing two series arcs into one arc.

Two enumerative algorithms for finding the node pair disconnection probability with perfectly reliable nodes are given in [HANSLER, 1974] and [FRATTA, 1973]. Hansler, McAuliffe and Wilkov produce as output a polynomial in P, the constant arc failure probability. Using an APL implementation on an IBM 360-91 computer, their algorithm ran on two 9-node, 12-arc networks in a total of 18 seconds. Fratta and Montanari used a network reduction technique to reduce a 21-node, 26-arc network to an 8-node, 12-arc network. They used a FORTRAN IV implementation on an IBM 360-67 computer. Once the reduction was accomplished, they used their enumerative algorithm on the 8-node, 12-arc network to produce the exact disconnection probability. The total time for the reduction and the enumerative algorithm was 112 seconds. The reduction algorithm most probably took a small percentage of that time.

Reduction techniques have been most successful in finding the probability that a specified pair of nodes can communicate where

parallel, and series arcs can be collapsed into single arcs. [ROSENTHAL, 1974] gives more sophisticated reduction techniques for finding other reliability measures. Rosenthal gives no computational experience; however, it appears that his techniques may be valuable for analyzing sparse networks. Generally, networks can only be reduced so far, so reduction techniques must be used in conjunction with other methods. The one exception is in the case of tree networks.

In [KERSHENBAUM, 1973] a recursive reduction algorithm is given for determining a variety of reliability measures (including all of those mentioned in this Chapter) on tree networks. A 500-node tree was run in 1 1/2 seconds. Algorithms for general networks cannot come close to solving problems of this size.

Simulation methods have been given in [VAN SLYKE, 1972] and [VAN SLYKE, 1975]. They provide a great deal of flexibility in the measures that can be investigated. In addition, they contain powerful sensitivity analysis capabilities. For a given number of samples, the running times increase almost linearly in the number of nodes and arcs. A 9-node, 12-arc network was run using the simulation algorithm on a PDP-10 computer with a FORTRAN IV implementation. The simulation algorithm produced the expected number of node pairs communicating and the probability that all operative nodes can communicate in 54 seconds with a standard deviation of .0046.

We have implemented our algorithms using FORTRAN IV on a PDP-10 computer. The results indicate reduction in running time over existing analytic algorithms. In addition, our algorithms produce reliability measures of interest to network designers, whereas most of the previous work was concentrated on the specified node pair problem. Our algorithms also appear to be much quicker than simulation algorithms for networks with fewer than 20 arcs. A complete summary of computational experience is given in a later section.

2.2 NODE PAIR DISCONNECTION

We will first consider finding the probability that a specified node pair cannot communicate. One minus this value will give us the probability that the specified pair can communicate which is the reliability measure of interest. Henceforth, this node pair will be denoted as (S,T). For the moment, we will assume that nodes are perfectly reliable. All algorithms presented use the same basic approach. The approach is best illustrated through the specified node pair problem which is the simplest. Our algorithm embodies the general idea of [HANSLER, 1974] in a backtracking structure. Their algorithm and our's enumerate a set of "modified cutsets." A modified cutset is the assignment of one of the states, operative, inoperative or free to all arcs in the network in such a way that the inoperative arcs form a cutset with respect to the specified node pair. All modified cutsets are mutually exclusive and collectively exhaustive with respect to the specified node pair being disconnected. Therefore, the sum of their probabilities is the probability that the specified node pair cannot communicate.

The algorithm maintains a stack of arcs marked operative or inoperative at all times. If the stack contains a cutset of inoperative arcs, the probability associated with the stack is computed and added to a cumulative sum. If the stack does not contain a modified cutset, one is generated by adding to the stack and marking inoperative a set of free arcs. After each modified cutset is considered, the algorithm backtracks. A backtrack consists of altering the arc on top of the stack in some way. If this arc is marked operative, it is taken off the stack and another backtrack is performed. If the arc is marked inoperative and if, when marked operative, a path consisting of operative arcs would exist between S and T, it is taken off the stack and another backtrack is performed. If the arc is marked inoperative and the above condition does not hold, it is marked operative and new modified cutset is found.

Given this basic structure, a number of algorithms could be developed depending on how the arcs to be made inoperative are chosen. Any such algorithm will fit into the following general form:

STEP 0: (Initialization)

Mark all arcs free; create a stack which is initially empty.

STEP 1: (Generate modified cutset)

a. Find a set of free arcs that together with all inoperative arcs will form an S-T cutset.

b. Mark all the arcs found in 1a inoperative and add them to the stack.

c. The stack now contains a cutset; use it to update the S-T disconnection probability.

STEP 2: (Backtrack)

a. If the stack is empty, we are done.

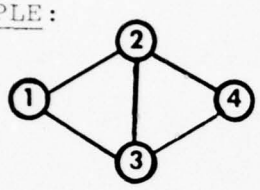
b. Take an arc off the top of the stack.

c. If the arc is inoperative and if when made operative, a path consisting only of operative arcs would exist between S and T, then mark it free and go to 2a.

d. If the arc is inoperative and the condition tested in 2c does not hold, then mark it operative, put it back on the stack and go to Step 1.

e. If the arc is operative, then mark it free and go to 2a.

EXAMPLE:



S = 1
T = 4

12 implies arc 12 is inoperative
$\overline{12}$ implies arc 12 is operative

Examples of possible stack configurations:

12, 13 — 12, 13 are inoperative. All other arcs are free. This is a modified cutset since 12 and 13 form an S-T cutset and they are inoperative. If this were the stack configuration at Step 2 13 would be marked operative.

12, $\overline{13}$,24,34 — 12,24,34 are inoperative; 13 is operative. All all other arcs are free. This is a modified cutset since 24 and 34 form an S-T cutset and they are inoperative. If this were the stack configuration at Step 2, 34 would be taken off the stack, since if it were marked operative, 13 and 34 would form an operative S-T path.

$12,23,\overline{34}$	12, 23 are inoperative; 34 is operative. All other arcs are free. This is not a modified cutset. If this were the stack configuration at Step 2 $\overline{34}$ would be removed from the stack since it is operative.

We have devised two algorithms based on this general algorithm. Algorithm I enumerates a set of modified cutsets similar to the set enumerated by Hansler, McAuliffe and Wilkov. Algorithm II enumerates a set of minimum cardinality modified cutsets with the use of the max-flow, min-cut algorithm.

In Algorithm I, at any given time all operative arcs will form a tree rooted at node S. A modified cutset is formed from any given structure in the stack by marking inoperative all free arcs that connect the tree to the rest of the graph. When backtracking, an inoperative arc will be taken off the stack if it is incident to node T.

DATA STRUCTURE FOR ALGORITHM I:

NN - # of nodes
NA - # of arcs

For each arc A:

Node1(A) and Node2(A) are the two nodes incident to arc A.

PA(A) - failure probability of arc A

$$\text{STATE}(A) = \begin{cases} 1 & \text{if A is free} \\ 2 & \text{if A is inoperative} \\ 3 & \text{if A is operative.} \end{cases}$$

For each node N:

$$\text{LABEL}(N) = \begin{cases} 2 & \text{if N is in the tree} \\ 1 & \text{if N is not in the tree} \end{cases}$$

STACK(K) - K^{th} arc from the bottom of the stack.
TOP - top of stack.
PROB - probability of the configuration on the stack.
FAIL - sum of the probabilities of all modified cutsets generated thus far.

ALGORITHM I:

STEP 0: (Initialization)

LABEL(S) ← 2
LABEL(N) ← 1 for all N ≠ S
STATE(A) ← 1 for all A
TOP ← 0
PROB ← 1
FAIL ← 0

STEP 1: (Find a modified cutset)

a. Find an arc A s.t. STATE(A) = 1 and either [LABEL (NODE1(A)) = 1 and LABEL(NODE2(A)) = 2] or [LABEL(NODE1(A) = 2 and LABEL(NODE2(A)) = 1.]

If no such arc exists go to 1c.

b. STATE(A) ← 2
TOP ← TOP + 1 STACK(TOP) ← A
PROB ← PROB * PA(A)
Go to 1a.

c. FAIL ← FAIL + PROB

STEP 2: (Backtrack)

a. If TOP = 0 stop; FAIL contains the S-T disconnection probability,

otherwise A ← STACK (TOP)

b. If STATE(A) = 2 then:
N ← node incident to A s.t. LABEL (N) = 1
If N = T then:

STATE(A) ← 1
TOP ← TOP - 1
PROB ← PROB/PA(A)
Go to Step 2a.

Otherwise:

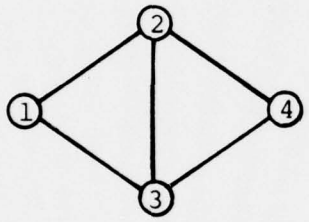
STATE(A) ← 3
PROB ← PROB*(1-PA(A))/PA(A)
LABEL(N) ← 2
Go to Step 1.

c. If STATE(A) = 3 then:

N ← node incident to A that was most recently labeled 2

LABEL(N) ← 1
STATE(A) ← 1
TOP ← TOP - 1
PROB ← PROB/(1 - PA(A))
Go to 2a.

EXAMPLE:



S = 1
T = 4

The sequence of modified cutsets generated by Algorithm I is:

12, 13
12, $\overline{13}$, 32, 34
12, $\overline{13}$, $\overline{32}$, 24, 34
$\overline{12}$, 23, 24, 13
$\overline{12}$, 23, 24, $\overline{13}$, 34
$\overline{12}$, $\overline{23}$, 24, 34

where 12 implies arc 12 is inoperative
and $\overline{12}$ implies arc 12 is operative

This algorithm has a very simple structure and all subprocedures take a small amount of time. The only subprocedure that cannot be done in constant time is finding all arcs incident to a node labeled 1 and a node labeled 2 (the loop between Steps 1a and 1b). In the worst case, all arcs incident to all nodes labeled 2 must be considered. (Presumably all nodes labeled 2 will be kept in a linked list.) This operation then is O(NA). Shortcuts can be incorporated into the algorithm which obviate the necessity of searching through all arcs incident to all nodes labeled 2. Whenever a return is made to Step 1 and no arcs have been switched from operative to free since the last modified cutset was generated, Step 1 can be shortened. A new modified cutset can be generated by marking inoperative all free arcs that join the tree to node T and all free arcs that are incident to the node just added to the tree and nodes not in the tree.

To simplify the description of the algorithm, we update PROB by multiplying or dividing by PA(A). Since PA(A) might be very small or indeed zero, this could be an unstable calculation. Simple procedures could be implemented to eliminate this division. This can be accomplished by storing a vector of probabilities, PROB. PROB(K) will be the probability associated with the first K elements on the stack. PROB need only be updated when the state of the top of the stack is changed or when an arc is added to the stack (e.g., if an inoperative arc, A, were added PROB would be updated by

$$PROB(ITOP) \leftarrow PROB(ITOP - 1) * PA(A)$$

THEOREM:

For any network, Algorithm I generates the S-T disconnection probability in a finite amount of time.

THEOREM:

If NM = the number of modified cutsets enumerated and NA = the number of arcs then Algorithm I is 0(NA * NM).

PROOF:

Any time an arc is made operative, a modified cutset is generated. As was shown earlier, in the worst case, this operation is 0(NA). All operations performed in Step 2 can be done in constant time. Each operation either results in an arc being made operative and thus a new cutset being generated, or an arc being deleted from the stack.///

Algorithm II generates modified cutsets through the use of the Ford-Fulkerson max-flow - min-cut algorithm. Given the current configuration in the stack, a minimum cardinality set of free arcs is found that will form an S-T cutset together with the arcs already inoperative. At any given time during the algorithm, the operative arcs form a forest in the network. When backtracking, it is more difficult to determine whether an inoperative arc should be made operative or free. To determine whether or not an arc would complete an S-T path, nodes that can communicate with node S and nodes that can communicate with node T, are given opposite labels. Thus, an arc which was incident to oppositely labeled nodes would not be marked operative.

DATA STRUCTURE FOR ALGORITHM II:

Same as for Algorithm I except the array LABEL is deleted and the following array added:

For each node N:

$$\text{REACH}(N) = \begin{cases} 0 & \text{if N cannot communicate with S or T} \\ L & \text{if there exists a path of length L - 1 of operative arcs between N and S} \\ -L & \text{if there exists a path of length L - 1 of operative arcs between N and T} \end{cases}$$

ALGORITHM II:

STEP 0: (Initialization)

```
REACH(S)  ← 1
REACH(T)  ← -1
REACH(N)  ← 0        for all N ≠ S or T
STATE(A)  ← 1        for all A
TOP ← 0     ;        PROB ← 1  ;  FAIL ← 0
```

STEP 1: (Perform max-flow - min-cut algorithm)

Use Ford-Fulkerson algorithm to find the min-cut between S and T with all operative arcs at infinite capacity, all inoperative arcs deleted from the graph and all free arcs at capacity 1.

STEP 2: (Add inoperative arcs to stack to generate modified cutset)

a. Mark all free arcs in the min-cut found in Step 1 inoperative and add them to the stack.

For each such arc A, PROB ← PROB * PA(A)

b. FAIL ← FAIL + PROB

STEP 3: (Backtrack)

a. If TOP = 0 stop; FAIL contains the S-T disconnection probability,

otherwise A ← STACK(TOP)

b. If STATE(A) = 2 then:

If REACH(NODE1(A) and REACH(NODE2(A) have opposite signs then:

TOP ← TOP-1
STATE(A) ← 1
PROB ← PROB/PA(A)
Go to 3a.

otherwise:

STATE(A) ← 3
PROB ← PROB * (1-PA(A))/PA(A)
If REACH(NODE1(A)) = REACH(NODE2(A)) = 0 then:
Go to Step 1.

otherwise:

Update REACH for all nodes
Go to Step 1.

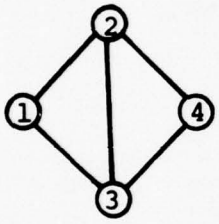
c. If STATE(A) = 3 then:

STATE(A) ← 1
PROB ← PROB/((1-PA(A))
If REACH(NODE1(A)) = REACH(NODE2(A)) = 0 then:
Go to 2a.

Otherwise:

Update REACH for all appropriate nodes.
Go to 2a.

EXAMPLE:



The sequence of modified cutsets generated by Algorithm II is:

12, 13
12, $\overline{13}$, 32, 34
12, $\overline{13}$, $\overline{32}$, 24, 34
$\overline{12}$, 24, 34
$\overline{12}$, 24, $\overline{34}$, 13, 23

Note that one less cutset was generated than in Algorithm I.

Algorithm II enumerates an entirely different partition of the probability space than Algorithm I. The number of events in this partition is smaller than in Algorithm I. Algorithm II

pays for this by the necessity of performing much more work per modified cutset generated. Again, every time an arc is made operative, the algorithm produces a modified cutset. To find this cutset, the max-flow algorithm must be performed. This, however, is an $0(NA*NN^2)$ algorithm. Also, every time a backtrack is performed which switches the state of an arc that communicates with S or T, REACH must be updated. This is accomplished by finding all nodes that are descendants of the node, whose state is changed in the tree of operative arcs emanating from nodes S or T. (The value of REACH(N) gives the level of node N in this tree.) In the worst case for a given node, this operation could be 0(NA). In backtracking past a given modified cutset, the value of REACH for a given node could be changed at most once. Therefore, for each modified cutset, the number of operations required to update REACH would be 0(NA).

THEOREM:

For any given network, Algorithm II generates the S-T disconnection probability in a finite amount of time.

THEOREM:

If NM = the number of modified cutsets enumerated, NN = the number of nodes and NA = the number of arcs, then Algorithm II is $0(NA*NN^2*NM)$.

PROOF:

The proof follows the logic in the equivalent proof for Algorithm I using the facts that the max-flow algorithm is $0(NA*NN^2)$ and updating REACH is 0(NA) for each modified cutset. ///

The results concerning the computational complexity of Algorithm II led us to believe that it would have a much higher running time than Algorithm I. Consequently, we did not code Algorithm II and all extensions in this paper refer to Algorithm I. Algorithm II does have many interesting properties which we hope to explore later.

2.3 NETWORK DISCONNECTION

A measure of the reliability of the entire network is the probability that all nodes can communicate. We chose to compute the probability that the network is disconnected which is one minus this value. Algorithm I extends to this case quite easily. Each modified cutset will disconnect the graph rather than only the specified node pair. (Clearly, any modified cutset which disconnects a specified node pair would also disconnect the graph.) Rather than stopping the growth of the tree when the specified node pair becomes connected, we stop it when it becomes a spanning tree. Spanning trees can easily be recognized by keeping a count on the number of operative arcs. The network disconnection probability can be found using Algorithm I with the following alteration:

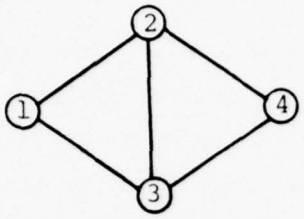
In Step 2b replace:

```
If N = T

     STATE(A) ← 1
     PROB ← PROB/PA(A)
     Go to 2a.
```

with:

```
If # of operative arcs = NN - 2

     STATE(A) ← 1
     PROB ← PROB/PA(A)
     Go to 2a.
```

EXAMPLE:



S = 1

The sequence of modified cutsets generated by Algorithm I with the network disconnection alteration is:

12, 13
12, $\overline{13}$, 32, 34
12, $\overline{13}$, 32, $\overline{34}$, 42
12, $\overline{13}$, $\overline{32}$, 24, 34
$\overline{12}$, 23, 24, 13
$\overline{12}$, 23, 24, $\overline{13}$, 34
$\overline{12}$, 23, $\overline{24}$, 43, 13
$\overline{12}$, $\overline{23}$, 24, 34

2.4 TRUNCATION

The number of modified cutsets grows exponentially with the size of the network. Since our algorithm enumerates these cutsets, its running time will grow exponentially with the size of the network. To accommodate larger problems, we have devised a truncation procedure which will find the disconnection probability within a tolerance supplied by the user. The following procedure assumes that all arc failure probabilities are equal. The idea is that since arc failure probabilities are usually very small (e.g., .02) events with more than a few arcs inoperative have low probability and can be ignored. Given a certain tolerance for the answer, we would like to find a limit on the number of inoperative arcs such that all modified cutsets with more inoperative arcs than the limit could be ignored.

Let:

p = arc failure probability
NA = number of arcs

Then a configuration of the network with exactly k arcs inoperative will have probability:

$$p^k \ (1 - p)^{NA - k};$$

there are C(NA,k), NA things taken k at a time, such events.

Therefore the total probability associated with all events with no more than L arcs inoperative is:

$$\sum_{k=o}^{L} C(NA,k) \ p^k \ (1 - p)^{NA-k}.$$

If TOL is the tolerance given for the answer and LIMIT is the maximum number of inoperative arcs, then LIMIT will equal the smallest L such that:

$$\sum_{k=o}^{L} C(NA,k) \quad p^k \ (1 - p)^{NA - k} \geq 1 - TOL$$

Given this LIMIT, the truncation procedure can be implemented quite easily. A count on the number of inoperative arcs can be kept. Whenever we are adding inoperative arcs to the stack to form a cutset (in Step 1) and the next arc to be added would make the count exceed LIMIT, that arc is added to the stack but marked operative. This addition gives us a new tree. If this new tree is not disconnected from the rest of the graph, we again take one of the free arcs that should be marked inoperative to form a cutset and mark it operative. We keep proceeding in this manner until the tree is cut off from the rest of the graph or until the next arc to be made operative was adjacent to node T (for network disconnection if the number of operative arcs equaled NN - 2). In the latter case, we would backtrack and in the former, we would have a new modified cutset.

DATA STRUCTURE FOR ALGORITHM III: (For network disconnection)

Same as for Algorithm I except:

COUNT1 = number of operative arcs
COUNT2 = number of inoperative arcs
LIMIT = maximum number of inoperative arcs

ALGORITHM III: (For network disconnection)

STEP 0:

```
LABEL(S) ← 2          (S arbitrary)
LABEL(N) ← 1          for all N ≠ S
STATE(A) ← 1          for all A
TOP ← 0
COUNT1 ← 0
COUNT2 ← 0
PROB ← 1
FAIL ← 0
```

STEP 1: (Generate modified cutset)

a. Find an arc A s.t. STATE(A) = 1 and either LABEL(NODE1(A)) = 1 and LABEL(NODE2 (A)) = 2 or LABEL(NODE1(A)) = 2 and LABEL(NODE2(A)) = 1.

If no such arc exists go to 1d.

b. If COUNT2 = LIMIT go to 1c

otherwise:

```
STATE(A) ← 2
TOP ← TOP + 1; STACK(TOP) ← A
COUNT2 ← COUNT2 + 1
PROB ← PROB * PA(A)
Go to 1a.
```

c. If COUNT1 = NN - 2, go to Step 2

Otherwise:

N ← node incident to A s.t. LABEL (N) = 1
LABEL(N) ← 2
STATE(A) ← 3
TOP ← TOP + 1 ; STACK(TOP) ← A
COUNT1 ← COUNT1 + 1
PROB ← PROB * (1 - PA(A))
Go to 1a.

d. FAIL ← FAIL + PROB

STEP 2: (Backtrack)

a. If TOP = 0 stop; FAIL contains the network disconnection probability

otherwise: A ← STACK(TOP)

b. If STATE(A) = 2 then:

N ← node incident to A s.t. LABEL(N) = 1
If COUNT1 = NN - 2 then:

STATE(A) ← 1
TOP ← TOP - 1
COUNT2 ← COUNT2 - 1
PROB ← PROB/PA(A)
Go to 2a

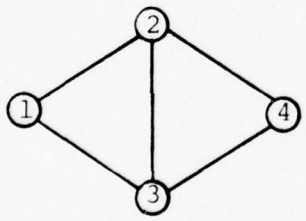
Otherwise:

```
STATE(A)  ← 3
COUNT1    ← COUNT1  + 1
COUNT2    ← COUNT2  - 1
PROB ← PROB * (1 - PA(A))/(PA(A))
Go to Step 1.
```

c. If STATE(A) = 3 then:

```
N ← node incident to A that was most recently labeled 2

LABEL(N)  ← 1
STATE(A)  ← 1
COUNT1    ← COUNT1  - 1
TOP ← TOP - 1
PROB ← PROB/(1 - PA(A))
Go to 2a.
```

EXAMPLE:



S = 1

The sequence of modified cutsets generated by Algorithm III with LIMIT = 2 is:

12, 13
$\overline{12}$, $\overline{23}$, 24, 34

In the new Step 1c, an arc is only made operative if it is incident to both a node labeled 1 and a node labeled 2. Therefore, the operative arcs will again always form a tree. Note that since at most NN - 2 arcs can be marked operative and with the truncation at most LIMIT arcs can be marked inoperative, the stack can never be larger than NN - 2 + LIMIT.

The order of the algorithm in terms of the number of modified cutsets enumerated can no longer be computed easily. Branches of the search tree can be added and deleted without generating a modified cutset. In Algorithm I, whenever an arc was made operative a cutset was generated by an operation that was linear in the number of arcs. Now we may delete and add many arcs only to find that the possible cutsets we were looking for all had more than LIMIT arcs inoperative. One advantage of Algorithm II in this area is that with the truncation, we would know immediately whether or not there were cutsets with less than LIMIT arcs inoperative. The reason we would have this information is that to generate a new cutset, we find the cutset with the minimum number of free arcs marked inoperative. If this minimum cutset were larger than LIMIT, then all cutsets would be larger than LIMIT. (It should be remembered that Algorithm II cannot be extended easily to the network disconnection problem.) Thus, Algorithm II would still be O(NN**2*NA*MC) where MC is the new number of modified cutsets enumerated.

THEOREM:

Algorithm III finds the probability that any network is disconnected within tolerance, TOL, if LIMIT is computed from TOL in the prescribed manner.

2.5 NODE FAILURES

While computations are simpler when only arcs can fail, in reality nodes are also unreliable. When considering the possibility of node failures a question arises as to the definition of network disconnection. The most obvious definition would be, "the network is disconnected any time at least one node cannot communicate with some other node" (ND1). By this definition, a network would be disconnected any time at least one node failed. An alternative definition which is much more useful for the network designer who has no control over node failure rates is, "the network is disconnected any time an operative node cannot communicate with another operative node" (ND2). Thus if a given node is inoperative its ability to communicate with the rest of the graph is irrelevant.

A. PROBABILITY {ND1}:

We will consider ND1 first simply because it is easier to handle. In fact, it reduces to the problem with perfectly reliable nodes.

Let:

ANO = {all nodes operative}
NANO ={not all nodes operative}
PN(N) = failure probability of node N
NN = number of nodes

Then since {ANO} and {NANO} are mutually exclusive, collectively exhaustive events the law of total probability gives us:

$$P\{ND1\} = P\{ND1|ANO\}*P\{ANO\} + P\{ND1|NANO\}*P\{NANO\}$$

P{ND1|ANO} can be found using Algorithm I with the network disconnection option.

$$P\{ANO\} = \prod_{N=1}^{NN} (1-PN(N))$$

$$P\{ND1|NANO\} = 1$$

$$P\{NANO\} = 1- P\{ANO\}$$

Thus with one extra straight forward calculation the graph disconnection problem with node failures reduces to the graph disconnection problem with perfectly reliable nodes.

B. PROBABILITY {S,T CANNOT COMMUNICATE}

The definition of ND2 presents a much more difficult problem for which major modifications to the algorithm are required. First, we will again consider the node pair disconnection problem.

Let:

$S \sim T$ imply S can communicate with T

and $S \not\sim T$ imply S cannot communicate with T

then:

$$P\{S \not\leftrightarrow T\} = P\{S \not\leftrightarrow T \mid S \text{ inop}\} * P\{S \text{ inop}\}$$

$$+P\{S \not\leftrightarrow T \mid S \text{ op}, T \text{ inop}\} * P\{S \text{ op}, T \text{ inop}\}$$

$$+P\{S \not\leftrightarrow T \mid S \text{ op}, T \text{ op}\} * P\{S \text{ op}, T \text{ op}\}$$

and:

$$P\{S \not\leftrightarrow T \mid S \text{ inop}\} = P\{S \not\leftrightarrow T \mid S \text{ op}, T \text{ inop}\} = 1$$

$$P\{S \text{ inop}\} = PN(S)$$

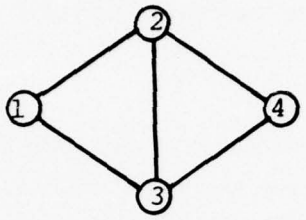
$$P\{S \text{ op}, T \text{ inop}\} = (1-PN(S)) * PN(T)$$

$$P\{S \text{ op}, T \text{ op}\} = (1-PN(S)) * (1-PN(T)).$$

The new version of the algorithm will compute $P\{S \not\leftrightarrow T \mid S \text{ op}, T \text{ op}\}$; i.e., we assume S and T are perfectly reliable and then find the probability that they cannot communicate.

The problem now has been reduced to enumerating a mutually exclusive, collectively exhaustive set of modified cutsets between S and T where nodes other than S and T can also "take part" in the cutset. The most straightforward modification to Algorithm I that would compute the desired probability would be to put nodes as well as arcs on the stack. Nodes are now marked either operative, inoperative, or free. Every time an arc is made operative, the new node added to the tree is placed on the stack and marked inoperative. To disconnect this tree from the rest of the network, all free arcs between operative nodes in the tree and free nodes are added to the stack and marked inoperative. When an inoperative node is encountered in a backtrack, it is switched to operative and a new modified cutset is found in the same manner.

Consider the following example:



S = 1

T = 4

The sequence of modified cutsets generated by the suggested algorithm for the {1,4} node pair disconnection probability is:

12, 13
12, $\overline{13}$, 3
12, $\overline{13}$, $\overline{3}$, 32, 34
12, $\overline{13}$, $\overline{3}$, $\overline{32}$, 2, 34
12, $\overline{13}$, $\overline{3}$, $\overline{32}$, $\overline{2}$, 24, 34
$\overline{12}$, 2, 13
$\overline{12}$, 2, $\overline{13}$, 3
$\overline{12}$, 2, $\overline{13}$, $\overline{3}$, 34
$\overline{12}$, $\overline{2}$, 23, 24, 13
$\overline{12}$, $\overline{2}$, 23, 24, $\overline{13}$, 3
$\overline{12}$, $\overline{2}$, 23, 24, $\overline{13}$, $\overline{3}$, 34
$\overline{12}$, $\overline{2}$, $\overline{23}$, 3, 24
$\overline{12}$, $\overline{2}$, $\overline{23}$, $\overline{3}$, 24, 34

where 1 implies node 1 is inoperative and $\overline{1}$ implies node 1 is operative.

A large savings can be realized by taking advantage of the equivalence between the following two events:

E_1 = {node N inoperative}

E_2 = {node N operative; all arcs between node N and free nodes inoperative}

Notice that in the example the modified cutsets {$\overline{12}$, 2, 13; $\overline{12}$, 2, $\overline{13}$, 3; $\overline{12}$, 2, $\overline{13}$,$\overline{3}$, 34} are the same as {$\overline{12}$,$\overline{2}$, 23, 24, 13; $\overline{12}$,$\overline{2}$, 23, 24, $\overline{13}$, 3; $\overline{12}$, $\overline{2}$, 23, 24, $\overline{13}$, $\overline{3}$, 34} except that 2 in the first set is replaced by $\overline{2}$, 23, 24 in the second set. Recognizing this similarity enables the problem with node failures to be reduced in terms of number of events enumerated to the problem without node failures. All cutsets derived assuming event E_2 are computed in the algorithm for perfectly reliable nodes. By suitable storage of information relating to event E_2 all cutsets assuming event E_1 can be calculated with little extra work. The revised algorithm we propose actually computes all cutsets associated with E_1 and then uses this information to compute the probability of the modified cutsets assuming E_2.

Rather than actually putting nodes on the stack we have created another state for the arcs. Besides being free and inoperative, an arc can be operative in two ways; states 3 and 4 both correspond to the arc being operative. In addition state 3 corresponds to one of the nodes incident to the arc being inoperative. State 4 correponds to both nodes being operative. An extra array, PROBB, is needed which saves the value of FAIL when an arc is put into state 3. When an arc is encountered in state 3 in a backtrack, PROBB is used to update FAIL for that arc in state 4, all arcs incident to it, and free nodes in state 2.

Algorithm IV computes the node pair disconnection probability when both nodes and arcs can fail. No truncation procedure is used.

DATA STRUCTURE FOR ALGORITHM IV:

Same as for Algorithm I except:

STATE will be changed to

$$\text{STATE}(A) = \begin{cases} 1 \text{ if A is free} \\ 2 \text{ if A is inoperative} \\ 3 \text{ if A is operative and one of the nodes incident to it is inoperative} \\ 4 \text{ if A is operative and both of the nodes incident to it are operative} \end{cases}$$

PROBB(K) = the value of FAIL when the K^{th} arc on the stack was placed on the stack.

LABEL will be changed to

$$\text{LABEL}(N) = \begin{cases} 1 \text{ if N is free} \\ 2 \text{ if N is operative} \\ 3 \text{ if N is inoperative} \end{cases}$$

PN(N) = Failure probability of node N

TEMP = Used in updating FAIL when an arc is changed from state 3 to 4.

ALGORITHM IV:

STEP 0: (Initialization)

```
LABEL(S) ← 2
LABEL(N) ← 1          For all N ≠ S
STATE(A) ← 1          For all A
TOP←0
PROB←1
FAIL←0
```

STEP 1:

a. Find an arc A s.t.

STATE(A) = 1 and either LABEL(NODE1(A)) = 1 and LABEL(NODE2(A)) = 2 or LABEL(NODE1(A)) = 2 and LABEL(NODE2(A)) = 1.

If no such A exists Go to 1c

b. STATE(A)←2

TOP←TOP + 1; STACK(TOP)←A
PROB←PROB * PA(A)
Go to 1a

c. FAIL←FAIL + PROB

STEP 2: (Backtrack)

a. If TOP = 0 stop; the probability that S and T do not communicate is PN(S) + (1-PN(S)) * PN(T)
+(1-PN(S)) * (1-PN(T)) * FAIL

Otherwise: A←STACK(TOP)

b. If STATE(A) = 2 then:

N←node incident to A such that LABEL(N) = 1
if N = T then:

STATE(A)←1
TOP←TOP-1
PROB←PROB/PA(A)
GO TO 2a

Otherwise

STATE(A)←3
LABEL(N)←3
PROB←PROB * (1-PA(A)) * PN(N) / PA(A)
PROBB(TOP)←FAIL
GO TO 1a.

c. If STATE(A) = 3 then:

TEMP←(FAIL - PROBB(TOP)) * (1-PN(N))/PN(N)
N←node incident to A such that Label(N) = 3
LABEL(N)←2; STATE(A)←4
PROB←PROB * (1-PN(N)) / PN(N)

[c.1] Find an arc B s.t. STATE(B) = 1 and either NODE1(B) = N and LABEL (NODE2(B)) = 1 or LABEL(NODE1(B)) = 1 and NODE2(B) = N.

If no such B exists go to c.3.

[c.2] STATE(B)←2
TOP←TOP + 1; STACK(TOP)←B
PROB←PROB * PA(B)
TEMP←TEMP * PA(B)
Go to [c.1]

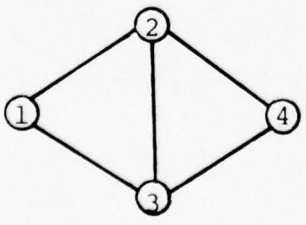
[c.3] FAIL←FAIL + TEMP
Go to 2a.

d. If STATE(A) = 4 then:

N←node incident to A which was most recently made operative

STATE(A)←1; LABEL(N)←1
TOP←TOP-1
PROB←PROB/((1-PA(A))*(1-PN(N)))

EXAMPLE:



S = 1

T = 4

Let:

12 imply STATE (12) = 2
$\overline{12}$ imply STATE (12) = 3
$\hat{12}$ imply STATE (12) = 4

The sequence of modified cutsets generated by Algorithm IV is:

12, 13
12, $\overline{13}$
FAIL is updated for 12, $\hat{13}$, 32, 34,...
12, $\hat{13}$, $\overline{32}$, 34
FAIL is updated for 12, $\hat{13}$, $\hat{32}$, 24, 34,...
$\overline{12}$, 13
$\overline{12}$, $\overline{13}$
FAIL is updated for $\overline{12}$, $\hat{13}$, 34,...
FAIL is updated for $\hat{12}$, 23, 24,...
$\hat{12}$, $\overline{23}$, 24
FAIL is updated for $\hat{12}$, $\hat{23}$, 34,...

It is instructive to compare this sequence of cutsets with those generated in the example for Algorithm I.

Note that division by PN(N) is necessary in Step 2c. The division necessary to update PROB can be handled in the manner prescribed for division by arc failures in Algorithm I. The division necessary in computing TEMP can be eliminated as follows. PROB will never be multiplied by PN(N). Instead it will represent the configuration in the stack not considering inoperative nodes. FAIL will be updated in the same manner in Step 1c. It will be updated for the node failure in Step 2c. To implement this change, the multiplication by PN(N) should be eliminated in Step 2b and FAIL and TEMP computed in Step 2c as follows:

```
TEMP←FAIL - PROBB(TOP)
FAIL←TEMP*PN(N) + PROBB(TOP)
TEMP←TEMP*(1-PN(N))
```

THEOREM:

For any network Algorithm IV generates the S-T disconnection probability is a finite amount of time.

C. PROBABILITY {ND2}

Algorithm IV can be extended to find the probability that at least one pair of operative nodes cannot communicate (ND2). Let us first see what happens when the simple change that was made to Algorithm I, to get the network disconnection probability, is applied to Algorithm IV. That is in Step 2b rather than returning an arc from state 2 to state 1 if it were incident to node T, we return it to state 1 if, when made operative, it would complete a spanning tree of operative arcs. With this alteration, each event enumerated would disconnect node S from at least one other node in the graph. We are interested in the probability that operative nodes cannot communicate. Therefore, the probability of each event enumerated will be multiplied by the probability that at least one node not in the tree is operative. The algorithm will then produce the probability that node S does not communicate with some other operative node. In other words, the algorithm will find the probability that at least one operative node pair cannot communicate given that node S is operative. If $\{H_I\}$, I = 1,...,NN, are mutually exclusive and collectively exhaustive,

$$P\{ND2\} = \sum_{I=1}^{NN} P\{ND2|H_I\} * P\{H_I\}$$

Let: $\{S_1, S_2, \ldots, S_{NN}\}$ be a permutation of the nodes

H_I = {node S_I operative; nodes $S_{I-1}, \ldots, S_1$ inoperative}

Clearly $\{H_I\}_{I=1,\ldots,NN}$ are mutually exclusive and collectively exhaustive.

$$P\{H_I\} = (1-PN(S_I)) \prod_{J=1}^{I-1} PN(S_J)$$

$P\{ND2|H_1\}$ is simply the output of the backtracking algorithm described in the preceding paragraph $P\{ND2|H_I\}_{I>1}$ is the output of that same algorithm performed on the network with nodes $S_1, \ldots, S_{I-1}$ deleted. Algorithm V uses the above analysis to compute P{ND2} with the backtracking algorithm essentially used as a subprocedure.

DATA STRUCTURE FOR ALGORITHM V:

DISCON = P {ND2} at the end of the algorithm
FAIL = P $\{ND2|H_N\}$ for current value of N
CONDPR = P $\{H_N\}$ for current value of N

ALGORITHM V:

STEP 0: (Initialization)

List the nodes in some desirable order $\{S_1, S_2, \ldots, S_{NN}\}$
N←1
CONDPR←1-PN(S_1)
DISCON←0

STEP 1: (Perform backtracking algorithm)

with S = S_N find FAIL = P{ND2|S operative}

STEP 2: (Check for termination; delete node from network)

DISCON←DISCON + CONDPR * FAIL
Delete S_N and all arcs adjacent to it from the network.
If N = NN stop; DISCON contains P{ND2}.
Otherwise: CONDPR←CONDPR * (1-PN($S_{N=1}$)) * PN(S_N)/(1-PN(S_N))

N←N+1
Go to Step 1

STEP 1, of course, is the only time consuming operation; it is when the backtracking algorithm is performed.

DATA STRUCTURE FOR ALGORITHM V, STEP 1:

Same as for Algorithm IV except,

COUNT1 = number of operative arcs
PROBOUT = probability that all free nodes are inoperative

ALGORITHM V, STEP 1:

Same as for Algorithm IV except,
in STEP 0 add:

$$\text{PROBOUT} \leftarrow \prod_{N \neq S} PN(N).$$

COUNT1←0

STEP 1c becomes:

```
FAIL←FAIL + PROB * (1-PROBOUT)
```

in STEP 2:

Replace a. with

a. If TOP = 0 go to Algorithm V STEP 2 otherwise A←STACK(TOP)

in b change if statement to:

```
If COUNT1  = NN-2
After otherwise add:
COUNT1 ← COUNT1 + 1
PROBOUT←PROBOUT / PN(N)
```

in d. add PROBOUT←PROBOUT * PN(N)

It might appear that the running time of this algorithm would increase dramatically. This is not the case. Since the running time of the backtracking subprocedure grows exponentially, the running time on a graph with one or two nodes deleted is much less than the running time of the algorithm on the original graph. Due to this fact the running time to find P{ND2} is generally less than twice the running time to find P{ND1}.

The choice of the ordering of the nodes is arbitrary. One good method is to pick the node with the highest degree, since on the next iteration the maximum number of arcs will be deleted. Looking toward a possible truncation, a procedure which could be used, is to pick the node with the lowest failure probability. In particular, if some node has 0 failure probability, it should be

chosen as S_1. In this case exactly one call to the backtracking procedure would be necessary since all probabilities subsequently enumerated must be multiplied by $PN(S_1)$.

We stated earlier that division by failure probabilities could be eliminated. In this algorithm another source of such divisions was added in the maintenance of PROBOUT. Again it would be wise to somehow eliminate these divisions. In this case it is slightly more difficult to do so efficiently.

One possibility is to search through all nodes each time a cut-set is generated and compute the product of the failure probabilities of all nodes not in the tree. This process is very time consuming. If any error at all can be tolerated, this search need only take place when a small number of nodes are not in the tree. In all other cases this product can be assumed to be zero.

THEOREM:

For any network Algorithm V computes P{ND2} in a finite amount of time.

D. TRUNCATION

With the addition of node failures, a number of new possibilities arise for truncation. First of all it can never be assumed that node failures and arc failures are equal. Certainly, a widely applicable algorithm should not assume that arc failure probabilities are equal or that node failure probabilities are equal. Thus far a highly effective truncation procedure for the variable probability case or the node failure case has not been developed. The truncation procedure given here produces significant results when most node failure probabilities are zero and arc failure probabilities do not vary widely from arc to arc.

Truncation takes place at two different places in the algorithm. First, a maximum number of inoperative arcs will be found as before. In each iteration of the algorithm when $P\{ND2|H_I\}$ is computed, all combinations of node failures will be considered (i.e., there will be no truncation due to a maximum number of inoperative nodes). At the end of each iteration the probability found will be multiplied by $P\{H_I\}= (1-PN(S_I)) * PN(S_I-1) *...*PN(S_1)$. When this value is sufficiently small the algorithm will be terminated. A certain percentage of the tolerance will be allocated to each of the two truncation procedures. Denote them by Procedure I and Procedure II respectively.

Procedure I: (Allocate .9 TOL; assume variable arc failure probabilities)

a given configuration of the graph will have probability:

$$\prod_{A\ inop} PA(A) \quad \prod_{A'\ op} (1-PA(A')) * P\{\text{Node configuration}\}$$

Number the arcs so that $PA(1) \leq PA(2) \leq \ldots \leq PA(NA)$.

Let:

$$PF(0) = \prod_{A=1}^{NA} (1 - PA(A)),$$

$$PF(0) = P\{\text{no arc fails}\}.$$

Let:

$$PF(1) = \sum_{A=1}^{NA} PA(A) \prod_{A' \neq A} (1-PA(A'))$$

$$PF(1) = P\{\text{exactly 1 arc fails}\}$$

Let:

$$PF(K) = C(NA,K) \prod_{A=1}^{K} PA(A) \prod_{A'=k+1}^{NA} (1-PA(A'))$$

$$K \geq 2$$

$$PF(k) \leq P\{\text{exactly k arcs fail}\}$$

Since we look at all combinations of node failures for each configuration of arc failures no probability will be "lost" to node failures. Therefore, LIMIT, the maximum number of inoperative arcs can be found as follows:

LIMIT ← smallest L such that

$$1-.9\ TOL \leq \sum_{N=0}^{L} PF(N)$$

Procedure II: (Allocate .1*TOL; variable node failure probabilities)

Claim: After finding $P\{ND2|H_M\}$ in algorithm V, if

$\prod_{I=1}^{M} PN(S_I) \leq .1*TOL$ we can stop and DISCON will be within

TOL of P{ND2}.

PROOF:

Let E = P{ND2|no more than LIMIT arcs are inoperative}

then

$$P\{ND2\} - \sum_{J=1}^{NN} P\{E|H_J\}\ P\{H_J\} \leq .9*TOL$$

$$P\{ND2\} - \sum_{J=1}^{M} P\{E|H_J\}\ P\{H_J\} \leq .9*TOL + \sum_{K=M+1}^{NN} P\{E|H_k\}*P\{H_k\}.$$

$$\sum_{K=M+1}^{NN} P\{E|H_k\}\ P\{H_k\}$$

$$= \left(\prod_{I=1}^{M} P\ N(S_I)\right) \sum_{k=M+1}^{NN} P\{E|H_k\}*(1-PN(S_k))* \prod_{H=M+1}^{k-1} PN(S_H)$$

$$\leq \prod_{I=1}^{M} PN(S_I) = .1*TOL,$$

implies

$$P\{ND2\} - \sum_{I=1}^{M} P\{E|H_J\}\ P\{H_J\} \leq TOL \qquad ///$$

Procedure II can be implemented in the obvious manner. Procedure I can be implemented in the same manner the truncation procedure was implemented in Algorithm III. It should be remembered that in Algorithm V, arcs are also marked inoperative in Step 2. Thus, it may be necessary to branch to the truncation procedure from there also.

2.6 EXPECTED FRACTION OF NODE PAIRS COMMUNICATING

The probability that a specified node pair does not communicate is an adequate measure for the specified node pair problem. P {ND1} and P {ND2} do not fully evaluate the reliability of an entire network. These measures essentially assume that a disconnected network is of no use. A disconnected network could be of value to many if not most of its users. The expected fraction of node pairs communicating is a global measure which gives a value to disconnected networks.

Unfortunately, this measure cannot be obtained by a straightforward and efficient extension of our algorithms. For the case where nodes are perfectly reliable, a satisfactory algorithm has been found. However, with node failures the problem appears to be much more difficult.

Assume perfectly reliable nodes, let:

EFC = expected fraction of node pairs communicating
I~J means I can communicate with J
I$\not\sim$J means I cannot communicate with J
NA = number of arcs
NN = number of nodes

$$EFC = \sum_{I=1}^{NN-1} \sum_{J=I+1}^{NN} \frac{P\{I\sim J\}}{C(NN,2)}$$

Although we have an algorithm for computing $P\{I\not\sim J\} = 1 - P\{I\sim J\}$, computing this probability for all pairs {I,J} would be extremely time consuming. In addition it would seem that much more information is obtained than is used each time the backtracking algorithm is run.

The above idea can be modified slightly to obtain an algorithm which requires fewer calls to a backtracking algorithm. Consider

the problem of finding the expected number of pairs that cannot communicate with a specified node, S. Our network disconnection algorithm without node failures enumerates a mutually exclusive set of events, all of which disconnect at least one node from S. Let $\{E_k\}_{k=1,\ldots,KK}$ be all the events enumerated, then:

$P\{S \not\sim J\} = \sum_{k \in K_J} P\{E_k\}$ where K_J is the set of all k's such that if E_k occurs $S \not\sim J$.

Let:

ENC(S) = expected number of nodes not communicating with S, then:

$$ENC(S) = \sum_{J \neq S} P\{S \not\sim J\}$$

$$= \sum_{J \neq S} \sum_{k \in K_J} P\{E_k\}$$

$$= \sum_{k=1}^{KK} N(k,S) * P\{E_k\}$$

Where:

N(k,S)= the number of nodes that cannot communicate with S when event E_k occurs.

(The obvious change to the algorithm would be to multiply $P\{E_k\}$ by N(K,S) each time a cutset was generated.)

Therefore:

$$EFC = \sum_{N=1}^{NN} \frac{(NN-1 - ENC(N))}{2*C(NN,2)}$$

(We must divide by 2 since the probability that each node pair communicates will have been computed twice.)

The fact that we must divide by 2 seems to indicate that a savings could be obtained somewhere. Consider a backtracking procedure which computes the probability that a node pair does

communicate. For a given pair, such an algorithm would somehow be based on enumerating paths rather than cutsets. In terms of network disconnection, such an algorithm would be based on trees rather than cutsets.

Our algorithm for computing EFC will look at connection rather than disconnection. For a given node S it will look at a set of mutually exclusive events in which varying numbers of nodes communicate with S. Thus, we will again consider operative trees but now also consider operative spanning trees.

This type of algorithm can easily be obtained from our graph disconnection algorithm. Instead of backtracking after NN-2 arcs are operative, we backtrack after NN-1 arcs are operative (i.e., after a spanning tree is found). If the probability of each of these events is multiplied by the number of nodes in the tree minus one, the expected number of nodes communicating with node S will have been found. Consider multiplying the probability of each event by the number of node pairs in the tree. The result will then be the expected number of node pairs communicating which also communicate with S, i.e.,

$$\sum_{k=1}^{KK} NN(k,S) * P\{E_k\}$$

$$= \sum_{J=1}^{NN-1} \sum_{I \geq J+1}^{NN} P\{I \sim J \sim S\}$$

(I or J can equal S)

Where:

$NN(k,S)$ = number of node pairs that can communicate and which can also communicate with S when E_k occurs.

Suppose we choose another node $S_1 \neq S$ and can find:

$$\sum_{J \neq NN,S} \sum_{\substack{I \geq J+1 \\ I \neq S}} P\{I \sim J \sim S_1 \neq S\}$$

For a given I and J the events,

$$\{I\sim J\sim S_1 \not\sim S\} \text{ and } \{I\sim J\sim S\}$$

are mutually exclusive. If we continue this procedure for all nodes we will have:

$$EFC = \sum_{K=1}^{NN} \sum_{\substack{J\neq S_m \\ m<K \\ J\neq NN}} \sum_{\substack{I\geq J+1 \\ I\neq S_m \\ m<K}} \frac{P\{I\sim J\sim S_K \not\sim S_{K-1},\ldots,S_o\}}{C(NN,2)}$$

Where S_o is a dummy vacuous node.

The remaining problem is to find:

$$\sum_{\substack{J\neq S_m \\ m<k \\ J\neq NN}} \sum_{\substack{I\geq J+1 \\ I\neq S_m \\ m<K}} P\{I\sim J\sim S_k \not\sim S_{k-1},\ldots,S_o\}$$

i.e., the expected number of node pairs which communicate and which communicate with S_k but which do not communicate with $S_{k-1},\ldots,S_1$. The algorithm just proposed finds this quantity for $I = 1$ ($S_1=S$). To find this quantity for higher values of I we use the same algorithm but we never let nodes $S_{k-1},\ldots, S_1$ become part of the tree. This can be accomplished by backtracking whenever an arc is incident to a node S_m $m<k$ is about to be changed from inoperative to operative.

DATA STRUCTURE FOR ALGORITHM VI:

ENC = expected fraction of node pairs communicating

$$COND(S_n) = \begin{cases} 1 \text{ if } P\{I\sim J\sim S_n\} \text{ has already been computed} \\ 0 \text{ otherwise} \end{cases}$$

ALGORITHM VI:

STEP 0: (Initialization)

Choose an appropriate ordering of the nodes $\{S_1,\ldots,S_{nn}\}$

$n \leftarrow 1$

$COND(S_n) \leftarrow 0$ for all n

$ENC \leftarrow 0$

STEP 1: (Perform backtracking algorithm)

$$FAIL \leftarrow \sum_{\substack{I\neq NN \\ I\neq Sm \\ m<N}} \sum_{\substack{J>I+1 \\ J\neq Sm \\ m<N}} P\{I\sim J\sim S_n \nsim S_{n-1},\ldots,S_o\}$$

STEP 2: (Check for termination; update COND)

$$ENC \leftarrow \frac{FAIL}{C(NN,2)} + ENC$$

if n = NN-1 STOP

Otherwise:

$n \leftarrow n+1$

$COND(S_{n-1}) \leftarrow 1$

Go to STEP 1

Step 1, where the backtracking algorithm is performed, is the time consuming step.

DATA STRUCTURE FOR ALGORITHM VI, STEP 1:

Same as for Algorithm I except add:

PAIRS = Number of node pairs that can communicate in the current configuration in the stack.

COUNT1 = Number of operative nodes

$$COND(N) = \begin{cases} 1 \text{ if N can not be in the tree} \\ 0 \text{ otherwise} \end{cases}$$

ALGORITHM VI, STEP 1:

Same as Algorithm I except:

in Step 0 add:

PAIRS←0

COUNT1←1

STEP 1 (c) becomes

FAIL←FAIL + PROB * PAIRS

in Step 2(a) change:

if TOP = 0 stop to:

if TOP = 0 go to Algorithm VI Step 2

in Step 2(b) change if N=T then to:

if COND(N) = 1 or COUNT1 = NN-1

after otherwise add:

PAIRS←PAIRS + *COUNT1*

COUNT1←COUNT1 + 1

in Step 2(c) add:

COUNT1←COUNT1 - 1

PAIRS←PAIRS - COUNT1

We have not yet coded this algorithm so we cannot give accurate data on its efficiency. It appears, however, that the times should not increase appreciably over the other algorithms. Algorithm VI cannot be easily extended to the case with node failures because of the special trick used in that algorithm. Algorithm VI could be extended to include node failure by considering an inoperative node as a separate event, i.e., not taking advantage of the trick. If this were done, the running time could increase substantially.

THEOREM:

For any network Algorithm VI computes ENC in a finite amount of time.

2.7 COMPUTATIONAL EXPERIENCE AND COMPARISON WITH OTHER ALGORITHMS

We have implemented Algorithm IV, which computes the probability that a specified pair of nodes cannot communicate and Algorithm V, which computes the probability that some pairs of operative nodes cannot communicate. The implementations consider both node and arc failures and include the truncation procedures. When zero node failure probabilities are given, these algorithms reduce to the earlier versions with perfectly reliable nodes. The algorithms were coded in FORTRAN IV on a PDP-10 computer.

The following general conclusions were drawn from testing on a number of networks:

1. Running times for all algorithms increase exponentially with the number of arcs.

2. Exact answers can be obtained for any of the measures in under a minute for networks with 20 or fewer arcs.

3. The truncation procedure is very effective for networks with constant arc failure probabilities and 0 node failure probabilities. With non-zero node failure probabilities, the truncation procedure had little effect.

A. PROBABILITY {ND2}

A series of runs was made on networks with between 10 and 19 arcs. The following data is for the computation of P{ND2} with both non-zero but constant node and arc failure probabilities. (PA(A) = .02 for all A; PN(N) = .001 for all N). Each network was run with a tolerance of .000 and .0001. .0001 is generally between 1 and 10% of the disconnection probability. The true error usually was much smaller than the tolerance.

CPU Times in Seconds

NN	NA	.000 = TOL	.0001 = TOL
8	10	4.06	3.76
9	12	6.45	5.69
10	15	15.49	12.18
13	17	26.21	22.81
15	19	55.01	48.00

The times include the time to read in the data.

With 0-node failure probabilities the 15-node, 19-arc network ran in 37.65 seconds with .000 tolerance and 14.45 seconds with .0001 tolerance. A network with 19 nodes and 23 arcs ran in 37.08 seconds with .0001 tolerance. Notice that the effect of the truncation is much more pronounced when node failure probabilities are zero.

B. SPECIFIED NODE PAIR

A series of runs was made on networks with between 12 and 28 arcs. The following data is for the computation of the probability that a specified node pair cannot communicate. Node failures were zero and arc failures were constant (PN(N) = 0 for all N; PA(A) = .02 for all A). The networks were run with tolerances of .00, .0001 and .001. As before, the true error is usually much smaller than the tolerance.

CPU Times in Seconds

NN	NA	.000 = TOL	.0001 = TOL	.001 = TOL
9	12	4.36	4.20	4.34*
10	15	7.36	4.71	3.94
13	17	8.77	6.66	5.97
15	19	17.22	9.34	6.87
19	23	71.64	16.69	17.16*
24	28	not run	not run	45.36

*In these cases there was no reduction in time between .0001 tolerance and .001 tolerance. This occurs when LIMIT has the same value for TOL = .0001 or TOL = .001.

A 10-node, 19-arc network with .00 tolerance ran in 26.82 seconds. Comparing this time with the time for the 15-node, 19-arc network (17.22 seconds) seems to indicate that denser networks require higher running times per number of arcs.

C. COMPARISON WITH OTHER ALGORITHMS

To compare our algorithm with the results given in [HANSLER, 1974] we added the capability of producing a failure probability polynomial in P, the constant arc failure probability. Our algorithm produced the node pair disconnection probability for the two Hansler, McAuliffe and Wilkov test networks in 11.20 seconds. As was stated in the introduction, their algorithm required 18 seconds. However, it should be noted that different computers and languages were used.

To compare our algorithm with simulation results, we computed the probability that all operative nodes can communicate on the same 9-node, 12-arc network run using the simulation algorithm. The running time was 6.45 seconds. This is a marked improvement over the simulation time of 54 seconds; however, the simulation algorithm also computed the expected fraction of node pairs communicating.

VOLUME 5

Chapter 2

REFERENCES

REFERENCES

[FRATTA, 1973] Fratta, L., and U. Montanari, "A Boolean Algebra Method for Computing the Terminal Reliability in a Communication Network," IEEE Transactions on Circuit Theory, Vol. CT-20, No. 3, May 1973.

[HANSLER, 1974] Hansler, E., G. McAuliffe, and R. Wilkov, "Exact Calculation of Computer Network Reliability," Networks, Vol. 4, No. 2, 1974, pp. 95-112.

[HOPCROFT, 1973] Hopcroft, J., and R. Tarjan, "Efficient Algorithms for Graph Manipulation," Communications of the ACM, Vol. 16, No. 6, June 1973.

[KERSHENBAUM,1973] Kershenbaum, A., and R. Van Slyke, "Recursive Analysis of Network Reliability," Networks, Vol. 3, No. 1, 1973, pp. 81-94.

[READ, 1975] Read, R., and R. Tarjan, "Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees," Networks, Vol. 5, No. 3, 1975, pp. 237-252.

[ROSENTHAL, 1974] Rosenthal, A., Computing Reliability of Complex Systems, Ph.D dissertation, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1974.

[VAN SLYKE, 1972] Van Slyke, R., and H. Frank, "Network Reliability Analysis: Part I," Networks, Vol. 1, No. 3, 1972, pp. 279-290.

REFERENCES (cont'd)

[VAN SLYKE, 1975] Van Slyke, R., H. Frank, and A. Kershenbaum, "Network Reliability Analysis: Part II," R. E. Barlow (ed.), Reliability and Fault Tree Analysis, SIAM, 1975, pp. 619-650.

VOLUME 5

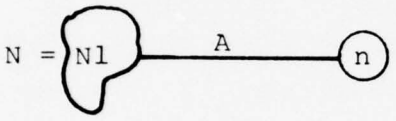
Chapter 2

APPENDICES

APPENDIX A: NETWORK REDUCTION

In this appendix, some techniques are given for reducing networks with nodes of degree 1, series arcs, and parallel arcs. Except for the parallel arc technique, these methods are specific to the reliability measure being computed.

A. Nodes of Degree 1



1. Network Disconnection (0 node failure probabilities)

Let:

ND = {network disconnected}

$N1$ = {N - A, n} ,

then:

$$P\{ND\} = P\{ND|A\ inop\}P\{A\ inop\} + P\{ND|A\ op\}P\{A_{op}\}$$

with

$$P\{ND|A\ inop\} = 1$$

$$P\{A\ inop\} = P_A$$

$$P\{A\ op\} = 1 - P_A$$

Therefore:

$$P\{ND\} = P_A + (1 - P_A)\, P\{N1D\}$$

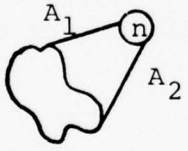
Thus, we have reduced the problem to finding $P\{N1D\}$

2. Node Pair Disconnection

If n is a member of the specified node pair, nothing should be done since our backtracking algorithm treats this case as simply as possible.

If n is not a member of the specified node pair, then A and n have no relevance to the problem so the backtracking algorithm can be performed on N1. Again the network has been reduced by one node and one arc.

B. Nodes of Degree 2

Network = 

1. Network Disconnection (0 node failure probabilities)

$$P\{ND\} = P\{ND|A_1, A_2 \text{ inop}\}P\{A_1, A_2 \text{ inop}\}$$

$$+ P\{ND|A_1 \text{ or } A_2 \text{ op}\}\, P\{A_1 \text{ or } A_2 \text{ op}\}$$

with

$$P\{ND|A_1, A_2 \text{ inop}\} = 1$$

$$P\{A_1, A_2 \text{ inop}\} = P_{A1}\, P_{A2}$$

If A_1 or A_2 are operative, n can communicate with at least one other node. Therefore, we need not consider explicitly n not communicating with another node. The only problem of relevance is whether the path A_1, n, A_2 is operative. Thus, we can replace this path with a single arc, A. A will fail if A_1 or A_2 fail.

$$P\{A \text{ inop} | A_1 \text{ or } A_2 \text{ op}\} = \frac{P\{A_1 \text{ or } A_2 \text{ fail and } A_1 \text{ or } A_2 \text{ op}\}}{P\{A_1 \text{ or } A_2 \text{ op}\}}$$

$$= \frac{P_{A1}\ (1 - P_{A2}) + P_{A2}\ (1 - P_{A1})}{1 - P_{A1}\ P_{A2}} = \frac{P_{A1} + P_{A2} - 2P_{A1}\ P_{A2}}{1 - P_{A1}\ P_{A2}}$$

If $P_{A1} = P_{A2} = P$ we have

$$\frac{2P - 2P^2}{1 - P^2} = \frac{2P}{1 + P} .$$

Let:

$$N1 = \{N - A_1,\ n,\ A_2 + A\} \quad \text{then}$$

$$P\{ND\} = P_{A1}\ P_{A2} + (1 - P_{A1}\ P_{A2})\ P\{N1D\}$$

where

$$P_A = \frac{P_{A1} + P_{A2} - 2P_{A1}\ P_{A2}}{1 - P_{A1}\ P_{A2}}$$

N1 has one less arc and one less node than N .

2. Node Pair Disconnection

If n is a member of the specified node pair there is no simplification that can be implemented.

If n is an intermediate node, then the only relevant question as far as the specified node pair is concerned is whether or not the path A_1, n, A_2 is operative. Therefore, we can replace this path by a single arc which fails any time at least one of A_1, n or A_2 fail. That is:

$$P_A = 1 - (1 - P_{A1})\ (1 - P_n)\ (1 - P_{A2})$$

C. Multiple Arcs

After deleting nodes and arcs in the first two parts or even in the initial data, there may be multiple arcs. These can easily be aggregated for any of the reliability measures as follows:

where

$$P_A = P_{A1}\ P_{A2}.$$

APPENDIX B: SAMPLE RUN

```
ST

ENTER # OF NODES AND # OF ARCS:<4 5

ENTER 0 FOR GRAPH DISCONNECTION PROB AND 1 FOR
NODE PAIR DISCONNECTION PROB:<0

ENTER THE TOLERANCE FOR THE ANSWER:<0.00

ENTER DEGREE OF EACH NODE AND NODE FAILURE  PROBABILITY
<2 .01 3 .01 3 .01 2 .01

ENTER NODES ADJACIENT TO NODE  1 AND FAILURE  PROBABILITY
OF THE CORRESPONDING ARC
<2 .02 3 .02

ENTER NODES ADJACIENT TO NODE  2 AND FAILURE  PROBABILITY
OF THE CORRESPONDING ARC
<1 .02 3 .02 4 .02

ENTER NODES ADJACIENT TO NODE  3 AND FAILURE  PROBABILITY
OF THE CORRESPONDING ARC
<1 .02 2 .02 4 .02

ENTER NODES ADJACIENT TO NODE  4 AND FAILURE  PROBABILITY
OF THE CORRESPONDING ARC
<2 .02 3 .02


NUMBER OF NODES:      4

NUMBER OF ARCS:      5

MAX NUMBER OF INOP ARCS PER CUT:      5

THE GRAPH DISCONNECTION PROBABILITY HAS BEEN FOUND

DISCONNECTION PROBABILITY: 0.00168711

TOLERANCE: 0.00000000


CPU TIME: 2.17  ELAPSED TIME: 1:9.74
NO EXECUTION ERRORS DETECTED

EXIT.
↑C
```

```
ST

ENTER # OF NODES AND # OF ARCS:<4 5

ENTER 0 FOR GRAPH DISCONNECTION PROB AND 1 FOR
NODE PAIR DISCONNECTION PROB:<1

ENTER THE TOLERANCE FOR THE ANSWER:<0.001

ENTER DEGREE OF EACH NODE AND NODE FAILURE  PROBABILITY
<2 .0 3 .0 3 .0 2 .0

ENTER NODES ADJACIENT TO NODE  1 AND FAILURE  PROBABILITY
OF THE CORRESPONDING ARC
<2 .02 3 .02

ENTER NODES ADJACIENT TO NODE  3 AND FAILURE  PROBABILITY
OF THE CORRESPONDING ARC
<1 .02 3 .02 4 .02

ENTER NODES ADJACIENT TO NODE  3 AND FAILURE  PROBABILITY
OF THE CORRESPONDING ARC
<1 .02 2 .02 4 .02

ENTER NODES ADJACIENT TO NODE  4 AND FAILURE  PROBABILITY
OF THE CORRESPONDING ARC
<2 .02 3 .02

ENTER THE 2 NODES WHOSE DISCONNECTION  PROB IS SOUGHT:
<1 4


NUMBER OF NODES:      4

NUMBER OF ARCS:      5

MAX NUMBER OF INOP ARCS PER CUT:      2

NODE PAIR DISCONNECTION PROBABILITY HAS BEEN FOUND
BETWEEN NODES  1 AND  4

DISCONNECTION PROBABILITY: 0.00078416

TOLERANCE: 0.00100000


CPU TIME: 2.33  ELAPSED TIME: 1:13.68
NO EXECUTION ERRORS DETECTED

EXIT.
↑C
@
```

VOLUME 5

Chapter 3

EXACT ALGORITHMS FOR NETWORK RELIABILITY ANALYSIS

3. EXACT ALGORITHMS FOR NETWORK RELIABILITY ANALYSIS

3.1 INTRODUCTION

In much of the work NAC and other network designers undertake, simulation is used to calculate the effect of variation of network design parameters. In such simulation studies, the reliability of a network must be calculated over and over again. Consequently, fast reliability analysis is mandatory. In addition, it is desirable to get exact reliability values for each sample so as to reduce the variance of the estimates. We have continued development efforts on fast exact algorithms to support large simulation studies. Generally speaking, it is possible to develop fast exact methods for small to moderately sized problems but for large problems the running times are prohibitive. In our packet radio reliability studies three types of routing strategies were considered: tree, restricted, and adaptive. Each strategy gave rise to a different network structure. A tree network was used to model tree routing. In [KERSHENBAUM, 1973], reliability algorithms which have linear time bounds were used for tree networks. These algorithms were directly applied to the packet radio analysis [NAC, 1976]. A directed acyclic network was used to model restricted routing. For this model we developed an algorithm whose running time increases exponentially with the number of nodes on a level and linearly with the number of levels. An undirected network was used to model adaptive routing. For this model we used a general reliability algorithm with some special features incorporated for networks in which only nodes can fail. The running time of this algorithm increases exponentially with the number of nodes in the network.

In this Chapter, we give a description of the algorithm for acyclic networks and an algorithm for undirected networks which extends those of Chapter 2. The tree algorithms have been adequately described in the literature. Both algorithms make use of probabilistic backtracking.

3.2 ACYCLIC NETWORK RELIABILITY ALGORITHM

3.2.1 The Network Structure and Its Relation to the Algorithm

The use of a restricted routing strategy in packet radio networks gives rise to a highly structured network reliability problem. First of all, as is the case for packet radio networks in general, all communication must proceed through the root node and only nodes can fail. We assume nodes fail independently with a known failure probability. Secondly, the limitations of restricted routing imply that the network is directed and acyclic. The nodes in the network can be partitioned into levels. The level of a node corresponds to the number of "hops" between that node and the root node. Links in the network are directed from nodes on one level to nodes on the the next lower numbered level. Level 0 contains exactly one node, the root node (see Figure 3.1).

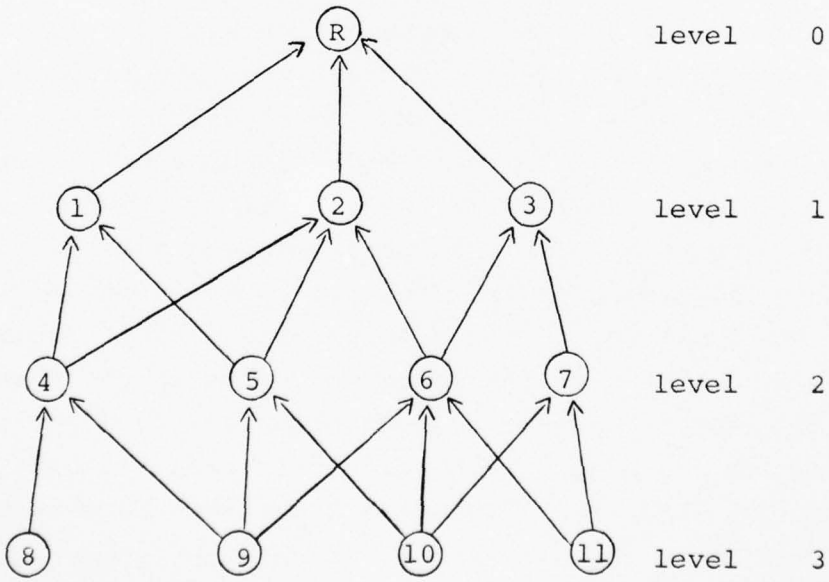
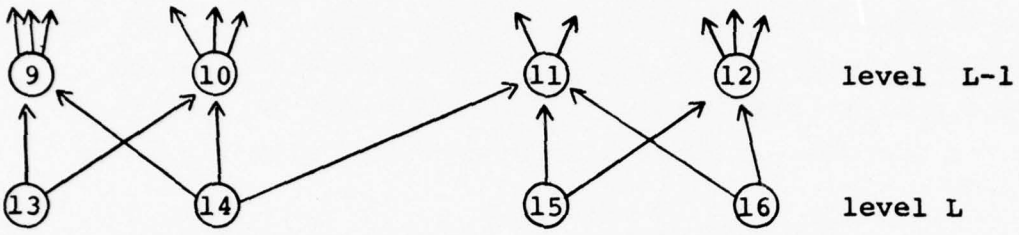


FIGURE 3.1: NETWORK STRUCTURE FOR RESTRICTED ROUTING

All paths from a node on level L to the root node contain exactly L links. In addition each of these paths intersect levels L-1 through 1 exactly once. Since all communication must go through the root at least one of these paths must be operative in order for the node on level L to communicate with any other node.

The algorithm we have devised is based on the following simple observation. The ability of a node on level L to communicate with the root can be completely determined by the ability of at least one of the nodes adjacent to it on level L-1 to communicate with the root. Therefore, reliability information for level L-1 is sufficient to determine the reliability of level L. The algorithm analyzes the reliability of each level in increasing numerical order. To compute the reliability of each level it uses only information from the previous level.

The question that must be answered in formalizing an algorithm is exactly what form should the reliability information of a level be put in. That is, what information concerning level L-1 is necessary in order to compute the reliability of level L. Given the state of a random network, the state of level L-1 can be simply given by the subset of nodes on that level that can communicate with the root. This subset on level L-1 will give rise to a different type of subset on level L. That is, all nodes on level L that are adjacent to nodes in the subset on level L-1. By definition all nodes in the subset on level L-1 can communicate with the root. The nodes in the subset on level L do not necessarily communicate with the root since some or all of them may be inoperative. A precise definition of the subset on level L is: the set of all nodes of level L that can communicate with the root assuming all nodes on level L are operative. We shall refer to this subset as a communicating subset. Note that the communicating subset on level L is a function of the node states on levels L-1 through 0 and is independent of the node states on level L (see Figure 3.2).



nodes on level L-1 that can comm. with root	communicating subset on level L
[9,10]	[13,14]
[10,11]	[13,14,15,16]
[11,12]	[13,14,15]

FIGURE 3.2: COMMUNICATION SUBSET AT LEVEL L

The objective of an algorithm is to get the reliability information for level L-1 into a form which will easily yield the reliability information for level L. Our algorithm partitions the probabilistic event space on level L-1 according to the communicating subsets yielded on level L. This is a valid partition for level L-1 since these subsets are independent of the nodes states on level L. The algorithm then analyzes each communicating subset on level L separately to yield all possible communicating subsets on level L+1; it continues in this manner until all levels have been considered.

The algorithm associates with each communicating subset the nodes in the subset, the subset probability and the values of various reliability measures. The communicating subsets represent disjoint

events. That is, given a random state of the network, there will be a unique communicating subset associated with each level. Since these events are disjoint, the reliability measures of the communicating subset on the final levels can be summed to yield the values of the network reliability measures.

At each iteration the algorithm must convert the information concerning a communicating subset on level L into information concerning a communicating subset on level L+1. To perform this analysis, the algorithm must consider all combinations of node states for the subset on level L. Each combination will yield a communicating subset on level L+1. Given the node states in the subset on level L and the subset probability and reliability measure values, the probability and reliability measure values for the communicating subset on level L+1 can be updated. In our algorithm a probabilistic backtracking procedure is used to perform these probabilistic updates. In this context the amount of computational work performed is reduced since all node state combinations need not be considered explicitly.

3.2.2 The Algorithm

The superstructure of our algorithm is given below in ALGOL notation. The algorithm iterates through all levels. At each level it converts the communicating subset list for level L, SSLSTL to the communicating subset list for level L+1, SSLST2. To perform this conversion it considers individually each communicating subset, SS1, on SSLST1 and finds all communicating subsets, SS2, on level L+1 that are achievable from SS1. SSLST1 is initialized with the only possible communicating subset on level 0, the root node.

```
PROC:   RELACYC
   Comment:  Procedure to determine the reliability of an acyclic
             network.

   BEGIN
      SSLST1←[R]
      FOR L←0 to NUMLEV-1 Do
         BEGIN
            SSLST2←Ø
            For each SS1 on SSLST1 Do
               BEGIN
                  Comment:  Find subset on level L+1 associated with
                                                  a state of SS1
                  NEWSS (SS1, SS2, IND)
                  While IND=true Do
                     Comment:  IND=False when all poss comm subsets
                               on level L+1 have been found
                     BEGIN
                        IF SS2 IS ON SSLST2 Then
                           Comment:  Update reliability values
                           UPDATE (SS2, SSLST2)
```

```
                        ELSE
                           Comment:  Put SS2 on SSLST2 and initialize
                                     its reliability values
                           INIT (SS2, SSLST2)
                        Comment:  Find another SS2
                        NEWSS (SS1, SS2, IND)
                  END
               END
            SSLST1←SSLST2
         END
      Compute network reliability measures from values in SSLST1
END RELACYC
```

SUBPROCEDURES

```
PROC:  NEWSS (SS1, SS2, IND)
   Comment:  NEWSS uses a probabilistic backtracking scheme to
     find communicating subsets on level L+1 assuming SS1 is on
     level L.  Each time it is called it returns another communica-
     ting subset, SS2.  In addition it finds the necessary proba-
     bilistic information associated with this subset.  If no more
     communicating subsets are possible on level L+1 starting with
     SS1 on level L it returns with IND=False.

PROC:  UPDATE (SS2, SSLST2)
   Comment:  UPDATE updates the reliability measures for SS2.

PROC:  INIT (SS2, SSLST2)
   Comment:  INIT initializes the reliability measures for SS2
     and puts SS2 on SSLST2.
```

Three subprocedures were used in the algorithm. UPDATE and INIT involved probabilistic updating and set manipulation. We will not give the details of the set manipulation but we will explain the probabilistic updating necessary for the reliability measures of interest. NEWSS performs the state enumeration of SS1. It uses a probabilistic backtracking scheme. In the next section we give the details of this procedure and in the last section the details of the probabilistic updating.

3.2.3 Probabilistic Backtracking Procedures

To illustrate the work that must be done by NEWSS we will consider a slightly simplified version of the problem. In the next section we will extend this procedure to the other reliability measures. Suppose we wish to find the probability of each possible communicating subset on level L+1 given a starting subset on level L and its probability. The straightforward approach would be simply to enumerate all combinations of node states. Given the state of all nodes, the resulting subset on level L+1 and its probability could be easily found (see Figure 3.3).

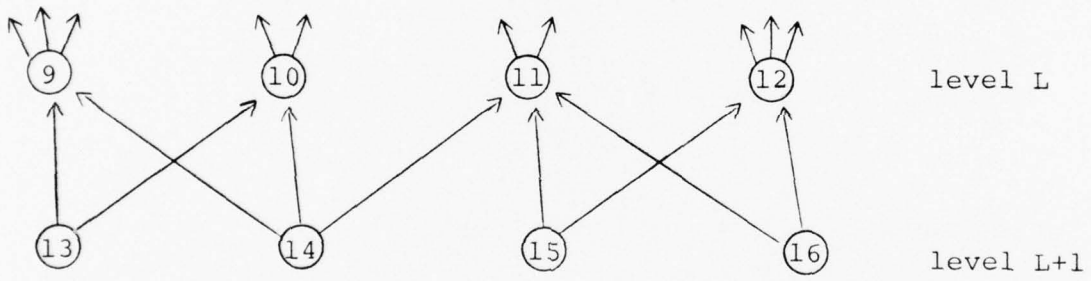


FIGURE 3.3: COMMUNICATION SUBSET ON LEVEL L+1

Let:

a implies a operative

$\bar{a}$ implies a inoperative

p = node fail prob

[9,10,11] = communicating subset on level L

PSS = prob [[9,10,11] is communicating]

For each state of [9, 10, 11,] the set of nodes on level L+1 adjacent to operative nodes in [9, 10, 11] will be the communicating subset on level L+1.

Node State on L	Prob	Comm. Subset on L+1
9,10,11	$(1-P)^3$	[13,14,15,16]
9,10,$\overline{11}$	$(1-P)^2P$	[13,14]
9,$\overline{10}$,11	$(1-P)^2P$	[13,14,15,16]
9,$\overline{10}$,$\overline{11}$,	$(1-P)P^2$	[13,14]
$\overline{9}$,10,11	$(1-P)^2P$	[13,14,15,16]
$\overline{9}$,10,$\overline{11}$	$(1-P)P^2$	[13,14]
$\overline{9}$,$\overline{10}$,11	$(1-P)P^2$	[14,15,16]
$\overline{9}$,$\overline{10}$,$\overline{11}$	P^3	ø

Comm. Subset on L+1	Prob.
[13,14,15,16]	PSS $((1-P)^3 + (1-P)^2 P + (1-P)^2P)$
[14,15,16]	PSS $((1-P)P^2)$
[13,14]	PSS$((1-P)^2P + (1-P)P^2 + (1-P)P^2)$
ø	PSS P^3

The communicating subset probabilities for level L+1 are obtained by summing the probabilities of all node states which yield that subset and then multiplying the sum by PSS. Notice that the number of communicating subsets on level L+1 is much less than the total number possible.

To increase efficiency, we use a probabilistic backtracking scheme to generate all possible subsets on level L+1 and their probabilities. The scheme adds nodes from the subset on level L to a stack. When a node is placed on the stack it is marked operative. Nodes are added to the stack until the addition of no new node would increase the number of nodes in the subset of level L+1. It then updates the probability of the communicating subset generated on level L+1 and backtracks. A backtrack consists of changing the state of the node on top of the stack from operative to inoperative. If the node is already inoperative it takes it off the stack and backtracks again. After backtracking it again adds nodes to the stack. In this manner it generates a set of disjoint probabilistic events with which the probabilities of communicating subsets on level L+1 can be updated.

To complete a description of the algorithm a choice rule must be given for which node to add to the stack. Our algorithm chooses the node which is adjacent to the most nodes not in the communicating subset generated on level L+1. The output from the probabilistic backtracking procedure is shown in Figure 3.4.

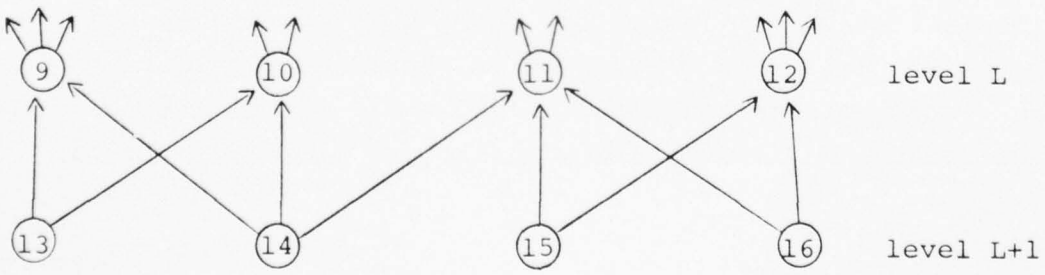


FIGURE 3.4: OUTPUT FROM PROBABILISTIC BACKTRACKING PROCEDURE

Assume the same notation and conditions as Figure 3.4.

The disjoint probabilistic events output by the backtracking algorithm are:

$11,9$
$11,\overline{9},10$
$11,\overline{9},\overline{10}$
$\overline{11},9$
$\overline{11},\overline{9},10$
$\overline{11},\overline{9},\overline{10}$

The probability of each communicating subset can be obtained in the same manner that it was obtained in Figure 3.3. In this case, the 6 disjoint events listed above replace the total state space enumeration. For this particular example the reduction was not too great (6 events instead of 8). However, for denser and larger networks the reduction would be much greater.

The procedure used in our description of the algorithm, NEWSS, returns one subset at a time. Using this backtracking procedure before each backtrack the subroutine NEWSS would return the subset associated with the stack configuration. In example 2.3B NEWSS would return a subset 6 times, once for each of the 6 different stack configuration. Note that it returns the same subset more than once. This does not create a problem since the stack configurations are disjoint and their probabilities can be added. On the 7^{th} call NEWSS would return with IND = false.

3.2.4 Probabilistic Updating Formulae

The reliability measures reconsidered are the probability of all operative nodes can communicate, POCR, the expected number of node pairs communicating with the root, EPCR, the expected number of nodes communicating with the root, ENCR, and for each node n, the probability that n can communicate with the root, PER(n). Our algorithm computes these measures by considering a larger set of nodes at each iteration. That is, for each level L it computes the measures over the subnetwork consisting of all nodes on levels L-1 through 0. The first three measures restricted to these subsets can be defined as: the probability that all operative nodes on levels L-1 through 0 can communicate with the root, POCR(L), the expected number of node pairs on levels L-1 through 0 that can communicate with the root, EPCR(L), and the expected number of nodes on levels L-1 through 0 that can communicate with the root, ENCR(L).

At each iteration the algorithm partitions the probabilistic event space restricted to the subset of nodes on level L-1 through 0. This partition groups all possible states of the nodes on levels L-1 through 0 according to the communicating subset they produce on level L. As was stated earlier this is a valid partition of the probability space restricted to levels L-1 through 0.

Define

NNS(L) = number of possible communicating subsets on level L

Impose an indexing on all communicating subsets so that

[SS(1,L), SS(2,L),...,SS(NSS(L),L)] = set of all possible communicating subsets in level L.

Let

PSS(I,L) = prob [SS(I,L) is the communicating subset on level L]

A value for each of the first three reliability measures is associated with each subset. This value is a conditional probability times the probability of the conditioning event.

POCR(I,L) = prob [all op. nodes on levels L-1 through 0 can comm. with root | SS(I,L) is the comm. subset on level L] * PSS(I,L)

EPCR(I,L) = exp [# nodes on levels L-1 through 0 comm. with root | SS(I,L) is the comm. subset on level L] * PSS(I,L)

In the previous section we showed how to find a set of disjoint node states which covered all possible states of a given subset on level L. In addition each state uniquely determines the communicating subset on level L+1. Below we show how to update the reliability measures for communicating subsets on level L+1 given the measures for communicating subsets on level L and the disjoint node states.

For each communicating subset on level L let

NST(I,L) = # of disjoint node states produced from SS(I,L) by the backtracking algorithm.

ST(J,I,L) = J-th state produced from SS(I,L)

PST(J,I,L) = prob [ST(J,I,L)]

NOS(J,I,L) = # of operative nodes in ST(J,I,L)

NNS(I,L) = # of nodes on level L not in SS(I,L)

Assume all reliability measures on L+1 have been initialized to 0. For each subset on level L the probabilistic backtracking algorithm is applied. The following are the update formulae for use when each node state is output:

Suppose SS(K, L+1) is the subset on level L+1 generated from SS(I,L) by ST(J,I,L) then the update formulae for all reliability measures are:

1. PSS(K, L+1)

   PSS(K,L+1) = PSS(K,L+1) + PSS(I,L) * PST(J,I,L)

2. PER(n)

   For each node n in SS(K,L+1)

   PER (n) = PER(n) + PSS(J,L) * PST(J,I,L)

3. POCR(K,L+1)

   POCR(K,L+1) = POCR(K,L+1) + POCR(I,L)
   PSS(I,L) * PST(J,I,L) * $P^{NNS(K,L+1)}$

4. ENCR(K,L+1)

   let ENCRL(J,I,L) = exp [#nodes on level L comm. with root when in state J of subset I]

   ENCRL(J,I,L) = NOS(J,I,L) + (1-P) * NNS(I,L)

   ENCR(K,L+1) = ENCR(K,L+1) + PST(J,I,K) *
   (ENCR(I,L) + ENCRL(J,I,K) * PSS (I,L))

5. EPCR(K,L+1)

let EPCRL(J,I,L) = exp [# node pairs on level L comm. with root when in state J of subset I]

$$EPCRL(J,I,L) = C(NOS(J,I,L),2) + \sum_{h=2}^{NNS(I,L)} C(h,2) * (1-P)^{h} P^{NNS(I,L)-h} * C(NNS(I,L,),h)$$

$$= C(NOS(J,I,L),2) + C(NNS(I,L),2) * (1-P)^{2}/2$$

where C(m,n) is m things taken n at a time

EPCR(K,L+1) = EPCR(K,L+1) +

PST(J,I,L) * (EPCR(I,L) + ENCR(I,L) *

ENCRL(J,I,L) + EPCR(J,I,L) * PSS(I,L))

After all updates have been performed, this procedure will have summed over all communicating subsets on level L and all node states in each subset. The sum for a given communicating subset on level L+1 can include terms from many subsets on level L. All of the terms can be summed because the backtracking procedure produces disjoint events and the communicating subsets on a level form a partition.

After the reliability measures have been computed for the last level, the network reliability measures can be computed. Essentially, the calculations are carried out one level farther. This imaginary level contains no nodes and consequently has one possible communicating subset, the null set. The output from the probabilistic backtracking routine for each level will be one node state. This state will have all nodes in the subset

free and its probability will be one. The same formulae can be used to update the measures for the one communicating subset on the imaginary level. The reliability measures associated with it will be the network reliability measures.

3.3 GENERAL NETWORK RELIABILITY ALGORITHM

3.3.1 Introduction

In this section we present an exact algorithm which extends those of Chapter 2 for the most general and consequently the most difficult network reliability problem. The model is an undirected network in which nodes and links can fail with independent known failure probabilities. This model applies as it is to the ARPANET. Assuming links are perfectly reliable it applies to packet radio networks using adaptive routing.

The network designer is primarily interested in measures of global network reliability. For example, the probability that all operating nodes can communicate and the expected fraction of node pairs communicating. In the presence of a centralized root node a designer could also be interested in the probability that all operating nodes can communicate with the root node and the expected fraction of node pairs communicating through the root. Much of the earlier work in this area has been concentrated on finding the probability that a specified pair of nodes can communicate in networks with perfectly reliable nodes. This measure was of interest in measuring the reliability of complex systems which were modeled as networks. With the inclusion of the possibility of node failures this measure can be useful to individual network users. When computed for all possible pairs of users it can also be useful to network designers.

The earliest work on the specified node pair problem recognized the mechanism by which series and parallel links could be combined. Other work on this problem enumerated all paths or cutsets and then either used these to obtain an approximate answer [JENSEN, 1969] or performed other operations to convert the paths or cutsets to a set of disjoint probabilistic events whose

probabilities could be added [FRATTA, 1973]. More recent algorithms have directly produced a set of disjoint probabilistic events whose probabilities could be added [HANSLER, 1974].

The problem of obtaining the more general reliability measures has not been considered until recently. In [VAN SLYKE, 1972] and [VAN SLYKE, 1975] simulation techniques are given. They can be used to obtain all of the global measures and efficiently produce these measures for a parametrization of the failure probabilities. In [ROSENTHAL, 1974] generalizations of series-parallel reductions are given which apply to the global measures. In addition he shows how these can be combined with algorithms for dividing a network into triconnected components to produce additional savings.

Our approach to the problem is to directly enumerate a set of disjoint probabilistic events in such a way that the expectation or probabilities can be combined in a convenient manner. We feel this approach is superior to the one which first enumerates all paths, trees or cuts and then forms disjoint events. In such an approach, the running time of both the enumeration and the combination into disjoint events grows exponentially with the size of the network. We have generalized probabilistic enumeration of the last chapter so that we can factor on larger events and thus reduce the size of the enumeration tree. Finally, we are able to save certain state information so that redundant enumerations can be eliminated. We hope that by generalizing techniques for saving state information series-parallel type reductions can be done implicitly.

3.3.2 Probabilistic Backtracking

Backtracking is a technique that has been used extensively to solve a variety of enumeration problems. Recently, it has been applied to network problems. For example, [REED, 1975] shows how it can be used to enumerate efficiently all trees or cycles in a network. Suppose we propose an enumeration problem such as enumerating all subsets of a set with a desired property. We examine elements in a prescribed order. When an element is examined, we decide whether or not to include it in the subset under construction. When the subset has the desired property we list it. Afterwards we change our decision about the last element and begin adding new elements until the subset again has the desired property. If changing our decision on an element cannot produce a subset with the desired property, we back up to the previous element. If this element has been considered both in and out, we back up again. If it has only been considered in one state, we change our decision on it and proceed as before. When the process terminates all subsets have been enumerated. This technique can be applied to many problems with a variety of possible rules for ordering the enumeration. We have found this technique very useful for determining the probability of a random occurrence. In the probabilistic context, backtracking proceeds by adding probabilistic events to a stack. When the probability of the random occurrence, given the intersection of all events on the stack, is known it is multiplied by the probability of the intersection of the events on the stack. The resulting probability is added into a cumulative sum. The event on top of the stack is then complemented and new events are added until the probability of the random occurrence is again known. After an event and its complement have both been considered we back up to the previous event and examine it. When this process terminates the cumulative sum contains the probability of the random occurrence.

Below we have given the general version of a probabilistic backtracking algorithm in ALGOL notation. This version can theoretically compute the probability of any random occurrence. To define a specific algorithm a choice rule for specifying the event to add to the stack must be given. In addition a method for at some point recognizing the probability of the random occurrence given the configuration on the stack must be available. The finiteness of the algorithm is dependent upon the choice rule and the ability to eventually recognize the probability of the random occurrence given the configuration on the stack. The set of events enumerated is mutually exclusive and collectively exhaustive. This fact together with the law of total probability guarantees the correctness of the algorithm.

PROBABILISTIC BACKTRACKING ALGORITHM

DATA STRUCTURE

PSUM = Sum of probabilities

STACK = Event stack

STACK ⇐ E implies E added to STACK

E ⇐ STACK implies E taken off STACK

STATE(E) = True if E has not yet been complemented, false otherwise

ALGORITHM

```
PROB(A)
    Comment:  Procedure to compute the probability of A
    BEGIN
       PSUM←0
       STATE (Ø)←true
       STACK ⇐ Ø
       WHILE E ¬= Ø DO
          BEGIN
             WHILE prob [A|STACK] is unknown DO
                BEGIN
                   Find event E to add to stack
                   STATE(E)← true
                   STACK ⇐ E
                END
             PSUM←PSUM + prob[A|STACK] * prob[STACK]
             E = STACK
             WHILE ¬ STATE(E) DO E ⇐ STACK
             IF E ¬ = Ø DO
                BEGIN
                   STATE(E)←False
                   STACK ⇐ E
          (        END
          END
    END
 RETURN(PSUM)
END PROB
```

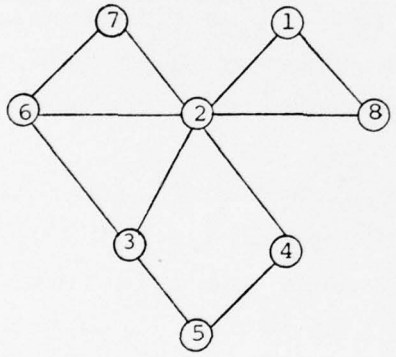
3.3.3 Depth First Search

Within the context of probabilistic backtracking many algorithms are possible for network reliability problems. We generate probabilistic events while searching through the network. Consequently, the type of events we generate are closely related to our search technique.

Depth first search has been shown to be a very useful method for exploring a network to examine its structural properties. For example [HOPCROFT, 1973] and [TARJAN, 1972] use this technique to divide a network into biconnected components and into triconnected components. To perform a depth first search we initially choose some arbitrary start node. Then we select a link adjacent to it and traverse that link to a new node. We continue in this manner by always traversing a link adjacent to the node most recently discovered which still has unexplored adjacent links. To implement this algorithm we maintain a stack. Each time a new node is explored it is added to the stack. After all links adjacent to one node have been explored that node is taken off the stack. The unexamined links adjacent to the next node on the stack are then examined.

Depth first search induces a graphical structure called a palm tree. Many of the desirable properties of depth first search can be observed from the structure of the palm tree. The links of a palm tree are directed and can be divided into two categories, tree links and fronds. Tree links form a directed tree rooted at the depth first search start node. They are the links from the original network along which each node was first discovered during the search. Each is directed into the node that was newly discovered when that link was used. Fronds are all other links from the original network. They are directed away from the node at which they were first examined. We define node v to be an ancestor

of node u if there exists a directed path consisting of tree links from node v to node u. Node u is a descendant of node v if node v is an ancestor of node u. Node u is the father of node v if the tree link into v originates from u. An important property of palm trees is that each frond is directed from a node to one of its ancestors. We associate a palm tree number with each node. This number corresponds to the order in which the node was discovered in the depth first search. For example the depth first search start node would have palm tree number 1 and the last node discovered would have palm tree number NN, where NN is the number of nodes in the network.

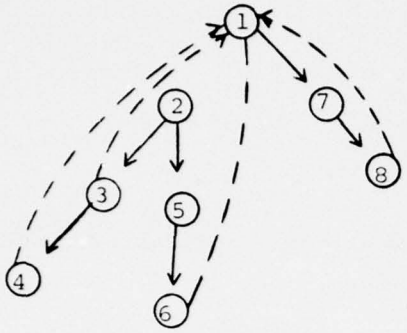


NETWORK

START NODE: 2

FIGURE 3.5: EXAMPLE OF PALM TREE

EXAMPLE 2.3:A (Cont'd.)



Palm tree with nodes renumbered according to palm tree numbers

Node	Palm Tree Number
2	1
3	2
6	3
7	4
5	5
4	6
1	7
8	8

v→w denotes a tree link
v-→w denotes a frond

We present below in ALGOL notation a depth first search algorithm. In its present form the only purpose of the algorithm is to generate a palm tree.

DEPTH FIRST SEARCH ALGORITHM

DATA STRUCTURE

Network Representation

NEXT(0,n) = First node on adjacency list of n

$$NEXT(u,n) = \begin{cases} w & \text{if } w \text{ is node } n \text{ adjacency list of } n \text{ following } u \\ \emptyset & \text{if } u \text{ is the last node in the adjacency list of } n \end{cases}$$

PTNUM(n) = palm tree number of u
PSTAC = palm tree stack
FATH(n) = father of u in palm tree

ALGORITHM

```
PTGEN
   Comment:  Procedure to generate a palm tree using depth first search

   BEGIN
      PTNUM(n)←∞for all n
      FATH(n)←0 for all n
      PTSTAC⇐(0,S)
      i←0
      WHILE PTSTAC is not empty DO
         BEGIN
            (w,u) ⇐PTSTAC
            w←NEXT (w,u)
            WHILE w ¬ = ∅ & PTNUM(w)<∞ DO
               BEGIN
                  IF PTNUM(w)<PTNUM(u) & FATH(u) ¬ = w
                     THEN construct u-→w in P.T.
                  w←NEXT(w,u)
               END
            IF w ¬ = ∅ THEN
               BEGIN
                  Comment: w is unnumbered
                  PTNUM(w)← i←i + 1
                  FATH(w)←u
                  Construct u→w in P.T.
                  PTSTAC ⇐(w,u)
                  PTSTAC ⇐(0,w)
               END
         END
   END
END PTGEN
```

3.3.4 Description of Algorithms

The algorithm we present in Sections 3.3.4 and 3.3.5 computes the probability that all nodes can communicate. Nodes are assumed to be perfectly reliable. In Section 3.3.6, we show how to include node failures and how to compute the more complex reliability measures.

Our algorithm uses depth first search to generate events for the probabilistic backtracking procedure. Each time a node is encountered in the depth first search a new event is added to the event stack. Events are of the form:

E = [at least one link in W(v) is operative]

where

W(v) = [u→v]∪F(v)

where

F(v) = [all fronds pointing out of v].

We say that v is the node associated with event E.

In other words, the event states that either the tree link into v or at least one of these fronds out of v is operative. Given that all nodes previously considered can communicate, this event implies that node v can communicate with them also. Events are added to stack until the depth first search terminates. At that time NN-1 events will be on the stack. Each node except the start node will be associated with exactly one event. This stack configuration implies that the network is connected.

When the algorithm backtracks it complements the top event. A complemented event states that for some v all links in the set W(v) are inoperative. If the stack configuration with the complemented top event implies the network is disconnected, the algorithm removes the event and considers the new top event. If this event is inoperative it is removed from the stack, otherwise it is complemented. When an event is made inoperative the links in the set W(v) are deleted from the network. The operative events left on the stack will correspond to a subpalm tree of the last spanning palm tree generated. A depth first search of the nodes not in the subpalm tree is performed. This search starts at the highest numbered node in the subpalm tree. When the search is terminated, the new palm tree generated contains the old subpalm tree and preserves its old palm tree numbers. Again, events are added to the stack during the depth first search. When the search terminates the stack will contain exactly NN-1 operative events and the configuration on the stack will again imply that the network is connected.

This procedure can be looked at in a slightly different light. A stack containing i operative events implies that the nodes numbered 1 through i+1 can communicate with each other. Thus, we can consider them one super node. Event i+1 implies that the node numbered i+2 is joined to the super node. Each time a new node is discovered it is added to the super node. When the search terminates the super node contains all nodes in the network.

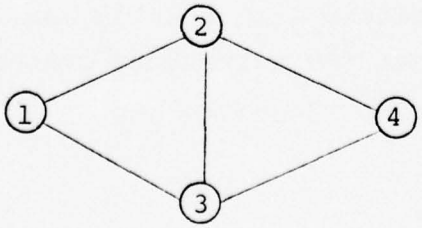


FIGURE 3.7: EXAMPLE OF ALGORITHM

Let [u→v, v-→w(1),...,v-→w(m)] represent the event: either u→v operative or at least one of v-→w(1),...,v-→w(m) operative.
Let E(k) represent the kth event on the stack, if it is operative

$\overline{E}(k)$ represent the kth event on the stack, if it is inoperative.

E(1) = [1→2]

E(2) = [2→3, 3-→1]

E(3) = [3→4, 4-→1]

backtrack: E(1), E(2), $\overline{E}$(3) node r isolated

backtrack: E(1), $\overline{E}$(2)

delete (2,3) and (3,1) from network

E(1) = [1→2]

$\overline{E}$(2) = ¬[2→3, 3-→1]

E(3) = [2→4]

E(4) = [4→3]

backtrack: E(1), $\overline{E}$(2), E(3), $\overline{E}$(4) node 3 isolated

backtrack: E(1), $\overline{E}$(2), $\overline{E}$(3) nodes 3 and 4 isolated

backtrack: $\overline{E}$(1)

delete (1,2) from network

$\overline{E}(1) = \urcorner[1 \rightarrow 2]$

$E(2) = [1 \rightarrow 3]$

$E(3) = [3 \rightarrow 2]$

$E(4) = [2 \rightarrow 4,\ 4 \rightarrow 3]$

backtrack: $\overline{E}(1)$, $E(2)$, $E(3)$, $\overline{E}(4)$ node 4 isolated

backtrack: $\overline{E}(1)$, $E(2)$, $\overline{E}(3)$

delete (3,2) from network

$\overline{E}(1) = \urcorner[1 \rightarrow 2)$

$E(2) = [1 \rightarrow 3]$

$\overline{E}(3) = \urcorner[3 \rightarrow 2]$

$E(4) = [3 \rightarrow 4]$

$E(5) = [4 \rightarrow 2]$

backtrack: $\overline{E}(1)$, $E(2)$, $\overline{E}(3)$, $E(4)$, $\overline{E}(5)$ node 2 isolated

backtrack: $\overline{E}(1)$, $E(2)$, $\overline{E}(3)$, $\overline{E}(4)$ nodes 2,4 isolated

backtrack: $\overline{E}(1)$, $\overline{E}(2)$ nodes 2,3,4 isolated

Since there are no more operative events on the stack we are done.

3.3.5 The Algorithm

When writing this procedure as a formal algorithm, two key questions must be answered:

1. When will complementing the event on top of the stack imply that the network is disconnected?

2. When a backtrack is completed, how can a new palm tree be efficiently generated from the subpalm tree left after the backtrack?

When backtracking to an operative event we can make it inoperative and generate a new palm tree if by complementing it we do not disconnect the network. If when complemented the event does imply disconnection, the operative event can simply be removed from the stack. Thus the answer to question 1 tells us whether an inoperative event encountered in the backtracking should be complemented or whether it should be removed from the stack.

To answer this question we associate a number, LOWPT(n) with each event. This number is similar to a number used in [TARJAN, 1972] to recognize cut points. The number is a function of the first palm tree generated after the event containing n is put on the stack. It is defined as:

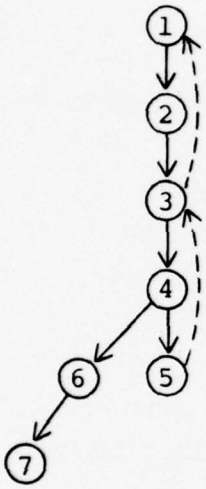
$$LOWPT(n) = \min [PTNUM(n), PTNUM(u) \text{ S.t. } u \in V(n)]$$

where

$V(n) = [u|$ there is a path in the palm tree from n to u consisting of one or more tree links and exactly one frond$]$

If LOWPT(n) < PTNUM(n) then there is a frond from a proper descendant of n to an ancestor of n in the palm tree. When the event containing n is complemented, all of n's descendants will be out of the subpalm tree and all of its ancestors will be in the subpalm tree. Thus, if LOWPT(n) < PTNUM(n) there will exist a path from a node in the subpalm tree to a descendant of n. This path will allow the subpalm tree to be grown into a spanning palm tree.

EXAMPLE OF USE OF LOWPT



For convenience, let the node numbers given in the palm tree be the palm tree numbers, then:

LOWPT(1) = 1
LOWPT(2) = 1
LOWPT(3) = 3
LOWPT(4) = 3
LOWPT(5) = 5
LOWPT(6) = 6
LOWPT(7) = 7

Suppose the last 4 events on the event stack are:

[3→4]
[4→5, 5→3]
[4→6]
[6→1]

When backtracking we would take the last 3 events off the stack. Note that in all cases if any one of them were marked inoperative some nodes would be isolated. This is indicated by the fact that for these events LOWPT(n)¬<n when [3↑4] is reached LOWPT(4)<4 and we mark this event inoperative. In this case it is possible to obtain a connected network with this event inoperative.

LOWPT can be easily computed during a depth first search. In the context of our backtracking algorithm some care must be taken to ensure that LOWPT values associated with each event are the values corresponding to the spanning palm tree generated immediately after the event was placed in the stack. In particular, this implies that the LOWPT values associated with the subpalm tree left after backtracking need not be recomputed.

Question 2 must be answered so that a new palm tree can be generated from the subpalm tree left after a backtrack. In our palm tree generation algorithm PTSTAC contains the information necessary to complete the generation of a palm tree at any stage in the algorithm. Ideally, we would like to have the PTSTAC corresponding to the subpalm tree left after a backtrack. We have been unable to find an efficient method for recreating this stack. Instead our algorithm maintains a stack containing all nodes currently in the subpalm tree ordered by palm tree number. A backtrack PTSTAC is then set equal to this stack and a new palm tree is generated. This process requires that all nodes left in the subpalm tree must be examined after every backtrack.

CONNECTION PROBABILITY ALGORITHM

Comments

1. Since events are placed on the stack as nodes are encountered in the search, all fronds pointing out of a node must be found when that node is first encountered in the search.

2. The statement: WHILE[(E¬=Ø) & [(LOWPT(u)>=PTNUM(u)) |¬FLAGA(u,w))]] DO

is a specific example of the general condition for backtracking in this type of algorithm.

E = ∅ implied no more events are on the stack so the algorithm terminates.

LOWPT(u)> = PTNUM(u) implies an operative event can be backtracked past

¬FLAGA(u,w) implies the event is inoperative so it can be backtracked past

DATA STRUCTURE

FLAGA(u,w) = true if (u,w) is in the network
false if (u,w) is inoperative

Event Representation
if E = [at least one of $u \rightarrow w$, $w \rightarrow v_1, \ldots, w \rightarrow (v_m$ is operative]

then

ENOD(E) = w
ENSTAC(E) contains $u, v_1, \ldots, v_m$ in any order
ESTAC is the stack of events

notice that u is not distinguished from the v_i's.

ALGORITHM

```
CONNECT
   Comment: procedure to compute the probability that a network with
            randomly failing links is connected

BEGIN
   INITIAL  comment:  initialization routine
   WHILE ESTAC is not empty DO
      BEGIN
         Comment:  compute a new disjoint connectivity event by
                   generating a new palm tree
```

```
      NEWPT
      PSUM←PSUM + prob[ESTAC]
      Comment:  backtrack
      E ⇐ ESTAC
      u ←ENOD(E)
      w ⇐ ENSTAC(E)
      WHILE [(E¬=Ø) & [(LOWPT(u)>= PTNUM(u))|(¬FLAGA(u,w))]] DO

          BEGIN
             Comment:  delete E from event stack
             DELEVENT (E,u,w)
             Comment:  now consider a new top event
             E ⇐ ESTAC
             IF E¬= Ø THEN
                  BEGIN
                     u←ENOD(E)
                     w⇐ENSTAC(E)
                  END
          END
        IF E¬= Ø THEN
           BEGIN
         Comment:  take complement of E and put it back on stack
         COMPEVENT(E,u,w)
         PTSTAC←ENSTAC  Comment:  PTSTAC is replaced by ENSTAC so
                                  that a new P.T. can be generated
                                  from the old sub P.T.
           END
      END
   RETURN (PSUM)
END CONNECT
```

SUBPROCEDURES

INITIAL: performs initialization

NEWPT: performs a depth first search starting with subpalm tree on PTSTAC. While doing so it updates ESTAC.

DELEVENT(E,u,w): deletes event E from ESTAC and performs necessary updates on network state variables.

COMPEVENT(E,u,w): complements event E and puts it back on ESTAC.

3.3.6 Generalizations and Improvements

3.3.6.1 Network Decomposition

It is possible and, in fact, common to encounter palm tree configurations which can be probabilistically decomposed. An example of the general structure, which we shall refer to as a split in a palm tree is shown in Figure 3.8.

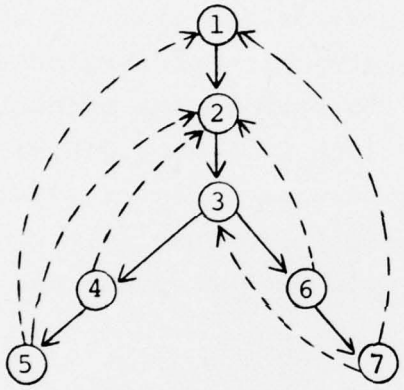


FIGURE 3.8: DECOMPOSED PALM TREE STRUCTURE

Given that nodes 1, 2 and 3 can communicate, the communication ability of nodes 4 and 5 and of nodes 6 and 7 are independent. To compute the connection probability, we can compute separately the probability that 4 and 5 can communicate with 1, 2 and 3 and then the probability that 6 and 7 can communicate with 1, 2 and 3. The connection probability given that 1, 2 and 3 can communicate will be the product of these two terms.

We have been able to implement this idea efficiently in our algorithm. Before 6 and 7 are reached in the depth first search we compute the connection probability of 4 and 5. Essentially we assume 4 and 5 are the last two nodes in the network. When

backtracking brings us to node 3 we notice that 6 and 7 have not yet been explored and thus identify the split structure. Using this implementation eliminates the need for a separate palm tree stack. Rather, the depth first search is guided by the event stack. In particular, this eliminates the difficulty described in question 2 of Section 3.3.4.

3.3.6.2 Updating Connection Probability

In the version of our algorithm given in this report, we look upon the probabilistic backtracking process as a method for generating disjoint events. To compute the connection probability, we sum the probability of each of these events. Another point of view is that the stack represents a set of events we have conditioned upon. To compute the connection probability given the stack configuration, we condition upon another event and use the law of total probability:

STACK = [intersection of events on stack]
CON = [net connected]
E = [conditioning event]

$$\text{prob}[CON|STACK] = \text{prob}[CON|STACK \cap E] * [E|STACK] + \text{prob}[CON|STACK \cap \bar{E}] * \text{prob}[\bar{E}|STACK]$$

Of course, E is simply the next event we will add to the stack. Looking at the process in this manner leads to another method for updating the connection probability. This probability will eventually be updated to be the connection probability given the configuration on the stack up to that level. At termination of the backtracking the probability associated with the bottom of the stack will be the connection probability.

We have found this update method superior for our algorithm. In particular, it is very compatible with our method for processing palm tree split.

3.3.6.3 Other Reliability Measures and Node Failures

This procedure can be extended easily to node failures. We look at the event:

E = [n operative and at least one link in W(n) operative] and its complement,

E= [n inoperative or all links in W(n) inoperative].

When considering node failures, we define a connected network as one in which all operative nodes can communicate. Our algorithm computes the probability that all operative nodes can communicate with the start node. This can be extended to the network connection measure in a manner described in Chapter 2 of this volume. The sequence of palm trees generated with node failures is exactly the same as the one without node failures. When it is found that marking an event inoperative will imply the network is disconnected, we cannot simply take the event stack. We must update the connection probability with the event inoperative by multiplying by the probability that all descendant nodes are inoperative. This requires that we keep track of the probability that all the descendant nodes are inoperative. This is straightforward in the context of depth first search.

The algorithm implicitly treats the complement of an event, E(n) as two events:

The algorithm implicitly treats the complement of an event, E(n) as two events:

[n operative and all links in W(n) inoperative] and [n inoperative].

If another event containing n is put on a stack, it will set n operative and thus eliminate the possibility of n inoperative. On the other hand, if n were cut off from the rest of the network and if it were operative the network would be disconnected. In such a case we can eliminate the possibility of n operative. The algorithm does not multiply in the probability of an inoperative event at its position in the stack. Rather, if an operative event is placed on the stack containing that node it then multiplies in the probability that the links in the inoperative event are inoperative. If the node is cut off from the palm tree, it multiplies in the probability that the node is inoperative.

We have included in the algorithm the capability of computing the expected fraction of node pairs communicating through the root node. Below we give the formulae for updating the expected number of node pairs communicating through the root and the expected number of nodes communicating with the root. We do not break inoperative events up as we did in computing the connection probability. Consequently, we may have to multiply by conditional probabilities which could involve division. The event probabilities given below are the conditional probabilities of the events given the configuration on the stack.

M = Number of independent subpalm trees splitting off from the palm tree at event E(K)

$E(J,K+1)$ = first event put on stack when Jth subpalm tree is reexplored

$E(J,\bar{K}+1)$ = complement of $E(J,K+1)$

$EP(J,K+1)$ = exp [Number of pairs comm. with root in Jth subpalm tree $|E(J,K+1)]$

$EP(J,\bar{K}+1)$ = exp [Number of pairs comm. with root in Jth subpalm tree $|E(J,\bar{K}+1)]$

$EN(J,K+1)$ = exp [Number of nodes comm. with root in Jth subpalm tree $|E(J,K+1)]$

$EN(J,\bar{K}+1)$ = exp [Number of nodes comm. with root in Jth subpalm tree $|E(\bar{J},\bar{K}+1]$

EP(K) = exp [Number of pairs comm. with root in subpalm tree starting with $E(K)|E(K)]$

EN(K) = exp [Number of nodes comm. with root in subpalm tree starting with $E(K)|E(K)]$

prob $[E(O,K)] = 1$, prob $[E(O,\bar{K})] = 0$

if E(K) represents an operative event $EN(O,K+1) = 1$

if E(K) represents an inoperative event $EN(O,K+1) = 0$

$$EP(K) = \sum_{J=1}^{M} (EP(J,K+1)\ \text{prob}\ [E(J,K+1)] + EP(J,\bar{K}+1)\ \text{prob}\ [E(J,\bar{K}+1)])$$

$$+ \sum_{J=0}^{M-1} [(EN(J,K+1)\ \text{prob}\ [E(J,K+1)] + EN(J,\bar{K}+1)\ \text{prob}\ [J,\bar{K}+1])$$

$$\sum_{I=J+1}^{M} (EN(I,K+1)\ \text{prob}\ [E(I,K+1)] +$$

$$EN(I,\bar{K}+1)\ \text{prob}\ [E(I,\bar{K}+1])]$$

$$EN(K) = \sum_{J=o}^{M} (EN(J,K+1)\ \text{prob}\ [E(J,K+1)] + EN(J,\bar{K}+1)$$

$$\text{prob}\ [E(J,\bar{K}+1)])$$

For updating when the palm tree does not split these formulae apply with M=1. We have incorporated these formulae recursively into our algorithm.

We have also included the capability of computing the probability that each node can communicate with the root. By running the algorithm with each node as a start node, we can use these probabilities to compute the expected fraction of node pairs communicating. Of course, these probabilities are of value in themselves. For this measure, we do not split up the inoperative events. We keep track of a stack probability. Whenever a new event is added to the stack we add the exact probability into a sum for the node associated with that event. At termination, the sum for each node represents the probability that it can communicate with the root.

3.3.6.4 Perfectly Reliable Links

The algorithm will compute reliability measures for networks with perfectly reliable links. However, it spends a great deal

of effort looking at event combinations that would not be possible with perfectly reliable links. To make the algorithm more efficient for this case, if the failure probability associated with the link failures in an event is 0, the link failures are left out of the complemented events. This implies that a node can never be associated with an operative event on the stack while it is associated with an inoperative event.

Another saving can be made when processing splits in the palm tree. With link failures, when the event from which a split started was made inoperative, the probability space could no longer be divided. If, however, only the node is made inoperative, this division is still valid. The reason is that the split occurs when the nodes in the palm tree above the split collectively form a cut node in the network. When the node from which the split is started is operative and connected to the rest of the palm tree, the palm tree forms a cut node. When that node is inoperative the rest of the palm tree forms a cut node. When that node is operative and not connected to the rest of the palm tree, the rest of the palm tree does not necessarily form a cut node. This is illustrated in Figure 3.9.

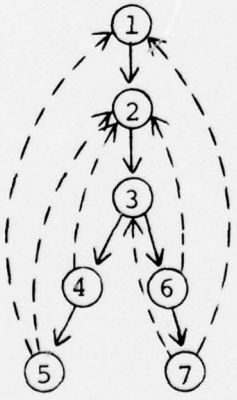


FIGURE 3.9: AN EXAMPLE OF A PALM TREE WHICH DOES NOT FORM A CUT NODE

With node 3 and link 2→3 operative all paths from 4 or 5 to 6 or 7 pass through 1, 2 or 3.

With node 3 inoperative all paths from 4 or 5 to 6 or 7 pass through 1 or 2.

With 2→3 inoperative there is a path (e.g., [2,4,3,6]) from 2 to 6 that does not pass through 1 or 2.

VOLUME 5

Chapter 3

REFERENCES

REFERENCES

[FRATTA, 1973] Fratta, L., and U. Montanari, "A Boolean Algebra Method for Computing the Terminal Reliability in a Communication Network," IEEE Transactions on Circuit Theory, Vol. CT-20, No. 3, May 1973, pp. 203-211.

[HANSLER, 1974] Hansler, E., G. McAuliffe, and R. Wilkov, "Exact Calculation of Computer Network Reliability," Networks, Vol. 4, No. 2, 1974, pp. 95-112.

[HOPCROFT, 1973] Hopcroft, J., and R. Tarjan, "Dividing a Network into Triconnected Components," SIAM Journal on Computing, Vol. 2, No. 3, September 1973, pp. 135-158.

[JENSEN, 1969] Jensen, P.A., and M. Bellmore, "An Algorithm to Determine the Reliability of a Complex System," IEEE Transactions on Reliability, Vol. R-18, No. 4, November 1969, pp. 169-174.

[KERSHENBAUM, 1973] Kershenbaum, A., and R. Van Slyke, "Recursive Analysis of Network Reliability," Networks, Vol. 3, No. 1, 1973, pp. 81-94.

[NAC, 1976] Network Analysis Corporation, "Reliability Considerations in Packet Radio Networks," in Recent Advances in Ground Packet Radio Systems, Sixth Semiannual Technical Report, Vol. 2, Ch. 4, February 1976, available from Defense Communication Center, Arlington, Va.

REFERENCES (cont'd)

[READ, 1975] Read, R., and R. Tarjan, "Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees," Networks, Vol. 5, No. 3, 1975, pp. 237-252.

[ROSENTHAL, 1974] Rosenthal, A., Computing Reliability of Complex Systems, Ph.D Dissertation, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1974.

[TARJAN, 1972] Tarjan, R., "Depth First Search and Linear Graph Algorithms," SIAM Journal on Computing, Vol. 1, No. 2, 1972, pp. 146-160.

[VAN SLYKE, 1972] Van Slyke, R., and H. Frank, "Network Reliability Analysis: Part I," Networks, Vol. 1, No. 3, 1972, pp. 279-290.

[VAN SLYKE, 1975] Van Slyke, R., H. Frank, and A. Kershenbaum, "Network Reliability Analysis: Part II," R.E. Barlow (ed.), Reliability and Fault Tree Analysis, SIAM, 1975, pp. 619-650.

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) NETWORK ANALYSIS CORPORATION BEECHWOOD, OLD TAPPAN ROAD GLEN COVE, NEW YORK 11542	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP None

3. REPORT TITLE
SIXTH SEMIANNUAL TECHNICAL REPORT, FEBRUARY 1976, FOR THE PROJECT, "LOCAL, REGIONAL AND LARGE SCALE INTEGRATED NETWORKS," (VOLUMES 1 THROUGH 5)

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
SIXTH SEMINANNUAL REPORT, FEBRUARY 1976

5. AUTHOR(S) (Last name, first name, initial)
NETWORK ANALYSIS CORPORATION
BEECHWOOD, OLD TAPPAN ROAD
GLEN COVE, NEW YORK 11542

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
FEBRUARY 1976	880	140

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DAHC-15-73-C0135 b. PROJECT NO. c. ARPA ORDER NO. 2286 d.	SEMIANNUAL REPORT 6 (5 VOLUMES) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) ---

10. AVAILABILITY/LIMITATION NOTICES

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

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency, Department of Defense

13. ABSTRACT This technical report presents significant accomplishments in the development of computational techniques for network analysis and design, and the following cost/performance tradeoffs for data and computer communication networks: cost-effectiveness and economies of scale previously demonstrated for medium size packet switching networks (e.g., ARPANET) extend to large scale networks (e.g., 10 Mbs throughput). The most cost-effective architecture for large scale networks is a multi-level hierarchical structure. The cost of hierarchical networks is not very sensitive to number of backbone nodes near the optimum; the local access cost is approximately 50% of the total, and is the most sensitive component. For AUTODIN II, Phase I, the minimum cost backbone network configuration is a hybrid terrestrial-satellite net with six (out of eight nodes) ground stations, resulting in 25% savings in backbone line cost compared to a terrestrial net only. Packet Radio local access is a cost-effective alternative to conventional techniques, providing a Packet Radio Unit can be manufactured for $20K-$30K. Significant results are reported on Packet Radio network communication protocols, throughput-delay-reliability analysis, and stability control.

14. KEY WORDS Computer networks, communication networks, terrestrial and satellite networks, packet radio networks, throughput, cost, delay, blocking, ARPA Computer Network, store-and-forward, packet switching.

UNCLASSIFIED
Security Classification