AFIT/GSO/ENS/91D-10

IMPROVING STOCHASTIC COMMUNICATION NETWORK PERFORMANCE : RELIABILITY VS. THROUGHPUT

THESIS

Leonard John Jansen
Captain, USAF

AFIT/GSO/ENS/91D-10

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1991	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE IMPROVING STOCHASTIC COMMUNICATION NETWORK PERFORMANCE : RELIABILITY VS. THROUGHPUT	5. FUNDING NUMBERS
6. AUTHOR(S) Leonard J. Jansen, Captain, USAF	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GSO/ENS/91D-10
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research investigated the measurement and improvement of two performance parameters, expected flow and reliability, for stochastic communication networks. There were three objectives of this research. The first was to measure the reliability of large stochastic networks. This was accomplished through an investigation into the current methodologies in the literature, with a subsequent selection and application of a factoring program developed by Page and Perry. The second objective was to develop a reliability improvement model given that a mathematical reliability expression did not exist. This was accomplished modeling a hueristic by Jain and Gopal, into a linear improvement model. Finally, the third objective was to examine the trade-off between maximizing expected flow and reliability. This was accomplished through generating bounds for the efficient frontier in a modified multicriteria optimization approach. Using the methodologies formulated in this research. the performance parameters of both expected flow and reliability can be measured and subsequent improvements made providing insight into the operational capabalities of stochastic communication networks.

14. SUBJECT TERMS Networks, Reliability, Stochastic Networks, Communication Networks	15. NUMBER OF PAGES 225
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT/GSO/ENS/91D-10

IMPROVING STOCHASTIC COMMUNICATION NETWORK PERFORMANCE : RELIABILITY VS. THROUGHPUT

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Space Operations)

Leonard John Jansen, B.S.
Captain, USAF

December, 1991

Thesis Approval

Student: Captain Leonard Jansen

Section: GSO 91D

Thesis Title: IMPROVING STOCHASTIC COMMUNICATION NETWORK PERFORMANCE : RELIABILITY VS. THROUGHPUT

Defense Date: November 21, 1991

COMMITTEE	NAME/DEPARTMENT	SIGNATURE
Advisor	Dr. Yupo Chan	Yupo Chan
Reader	Captain John Borsi/ENS	[illegible]

Preface

This research investigated the performance measurement and improvement of stochastic communications networks. The methodologies developed were the result of six months of research into network reliability and expected throughput. I would like to thank my thesis advisor, Dr. Yupo Chan, for his technical expertise and guidance. I would also like to thank my reader, Captain John Borsi, for his insights and suggestions. Finally, I want to thank my wife, Penni, for her support and understanding during these last eighteen months.

Leonard John Jansen

Table of Contents

AFIT/GSO/ENS/91D-10

Abstract

This research investigated the measurement and improvement of two performance parameters, expected flow and reliability, for stochastic communication networks. There were three objectives of this research. The first was to measure the reliability of large stochastic networks. This was accomplished through an investigation into the current methodologies in the literature, with a subsequent selection and application of a factoring program developed by Page and Perry (18). The second objective was to develop a reliability improvement model given that a mathematical reliability expression did not exist. This was accomplished modeling a hueristic by Jain and Gopal (15), into a linear improvement model. Finally, the third objective was to examine the trade-off between maximizing expected flow and reliability. This was accomplished through generating bounds for the efficient frontier in a modified multicriteria optimization approach. Using the methodologies formulated in this research, the performance parameters of both expected flow and reliability can be measured and subsequent improvements made providing insight into the operational capabalities of stochastic communication networks.

As with the expected maximum flow calculation, Yim's capacity improvement model will be used, which was examined in the literature review.

3.4 Formulating the Multicriteria Model

Given an objective function for each criteria, expected flow and reliability, a multicriteria optimization model was needed to generate the trade-off region or efficient frontier for the two criteria. Lacking a criterion function for network reliability, two modified approaches to examine the trade-off reqion were investigated in Chapter V.

3.5 Selecting Software

Given the performance measure and improvement models, software was required to implement the models. Previous work by Yim and Gaught used several different off-the-shelf commercial solver packages to include GINO and LP/MIP-83. The General Algebraic Modeling System, *GAMS*, was selected both for its ability to solve linear and nonlinear models on the personal computer and its ease in converting from one model to another. In addition to the *GAMS* solver, Yim's *Formula* program was used for network path generation and which required a Prolog compiler. Finally, integration procedures and input file generation programs were written in Pascal.

3.6 Analyzing Communication Networks

The above methodologies were applied to one small example network and three large realistic networks. The example network was used to demonstrate the application of the models to stochastic networks while the large networks were analyzed with respect to their initial performance and subsequent improved performance. Finally, the trade-off region was examined for the networks. The description of the three large networks along with their respective analysis is presented in Chapter VII.

List of Figures

List of Tables

AFIT/GSO/ENS/91D-10

Abstract

This research investigated the measurement and improvement of two performance parameters, expected flow and reliability, for stochastic communication networks. There were three objectives of this research. The first was to measure the reliability of large stochastic networks. This was accomplished through an investigation into the current methodologies in the literature, with a subsequent selection and application of a factoring program developed by Page and Perry (18). The second objective was to develop a reliability improvement model given that a mathematical reliability expression did not exist. This was accomplished modeling a hueristic by Jain and Gopal (15), into a linear improvement model. Finally, the third objective was to examine the trade-off between maximizing expected flow and reliability. This was accomplished through generating bounds for the efficient frontier in a modified multicriteria optimization approach. Using the methodologies formulated in this research, the performance parameters of both expected flow and reliability can be measured and subsequent improvements made providing insight into the operational capabalities of stochastic communication networks.

Improving Stochastic Communication Network Performance : Reliability vs. Throughput

I. Introduction

The Department of Defense (DoD) is concerned with the survivability of communications networks and the flow of information through the network during crisis situations. (12:2) The improvement of the expected maximum flow or throughput, and reliability of stochastic communications networks can be accomplished through increasing channel capacities or survivabilities within the network. Calculating the maximum expected flow for a large stochastic network is computationally infeasible, but very efficient mathematical programs have been recently developed to calculate the lower bound of the expected maximum flow, which has been demonstrated to be an accurate estimator. In addition, recent algorithms have been developed to calculate the reliability of large stochastic networks. Based on the above programs, the network performance parameters of expected flow and reliability can be measured and investment strategies modeled to improve stochastic network performance.

1.1 Background

A stochastic communications network can be represented by a graph consisting of nodes representing transmitters, receivers, or relay stations, and arcs representing communication channels (land lines, microwave or satellite links) which connect the individual nodes. Under adverse conditions, some arcs or nodes may fail and thus each has an associated probability of survival. This is the basis for the stochastic communication network in contrast to a deterministic network where there are no probabilities of failure.

To calculate the exact maximum flow through the stochastic network requires enumerating all possible states of the network. Assuming a network with n components (arcs or nodes), each being either up (operational) or down (failed), the total number of states is 2^n. Yim gives an example of a network consisting of 55 arcs which, given a typical personal computer, would require 114.25 years to enumerate all possible states. (23:2) While the exact value for expected maximum flow is computationally infeasible to solve for, Yim developed a mathematical programming model which accurately estimates the expected throughput of the network by calculating a lower bound value. (23:33)

Expanding on his expected throughput model, Yim developed several investment models for improving expected throughput by increasing individual arc capacities within the network. (23:38,40) Subsequently, Gaught developed additional nonlinear investment models for improving expected throughput by increasing the survivabilities of individual arcs within the network.

The performance parameter of expected throughput measures how much information a network can flow under adverse conditions given the probability of paths left to connect the source and sink. The expected flow is thus a weighted average of the flow times the probability of each state of the network. For some operations, however, the amount of flow is less important than the probability that the source and sink are connected. This cannot be derived from the expected flow value but must be calculated through the performance parameter of source to sink reliability.

1.2 Research Problem

It is the purpose of this investigation to develop methodologies to measure and improve the expected throughput and reliability of a stochastic communication network under budgetary limitations. The methodologies should be responsive to the decision maker's inputs, and computional times required should be short enough to facilitate what-if scenarios in an interactive mode.

1.3 Sub-objectives

There are three sub-objectives of this research.

1. The first sub-objective is to investigate methodologies to measure network reliability for large stochastic networks.
2. The second sub-objective is to develop a model to improve the network reliability of large stochastic networks.

3. The third sub-objective is to examine the trade-off between maximizing the two network performance parameters, expected flow and reliability, given a limited budget.

1.4 Assumptions

The following assumptions were made consistent with those used by Yim (23:3) and Gaught (12:3) :

1. Component failures are independent.
2. Components (arcs or nodes) are either up (operational) or down (failed).
3. The flow of information through an arc is restricted to one direction only.
4. Rerouting of the flow is not allowed (for the lower bound formulation).
5. Only a single commodity flows through the network.

1.5 Methodology

The first sub-objective will be met by investigating the current methodologies used to measure or bound network reliability and then select the most appropriate technique for this research. If possible, linear models will be selected over nonlinear models to avoid the problems of local optima and increased computational complexity associated with nonlinear models. In addition, the preferred methodology will operate within the memory and speed limitations of an IBM compatible personal computer.

The second sub-objective will be met by developing a model to improve the reliability of stochastic networks. It should be compatible with the selected methodology used above to measure network reliability. In addition, a mathematical solver package needs to be selected to implement the developed reliability improvement model.

The third sub-objective will be met by developing a methodology using multi-criteria optimization (MCO) to analyze the trade-off between the two criterion objectives of maximizing expected flow and reliability. This approach should produce a trade-off region from which an optimum solution can be obtained by applying the decision makers preference function pertaining to the two criteria.

II. Literature Review

This chapter reviews the literature applicable to single-commodity flows through a stochastic communications network , network reliability, and multi-criteria decision analysis. The following sections will discuss network representation, expected maximum flow, capacity improvements, survivability improvements, network reliability measures, and multi-criteria decision analysis methods.

2.1 Network Representation

A stochastic communication network is a collection of nodes interconnected by directed arcs or paths. The nodes may represent ground stations, repeaters, or satellites while the arcs represent communication links between the nodes. Each arc and node has an associated capacity and survivability where the capacity is the maximum flow the arc can carry while the survivability is the probability that the arc will be up or capable of carrying flow. In a communications network , flow represents the amount of information transmitted in units of bits per second (bps).

In addition to a graphical illustration, networks can be represented in matrix form using either a node-arc incidence matrix or an arc-path incidence matrix. In the node-arc form, the matrix rows correspond to arcs while the columns correspond to nodes. The matrix entries, e_{ik}, are one of the following: a 1. which implies that arc i ends at node k; a -1, indicating i starts at k; or a 0. meaning no connection exists (12:6). The arc-path form again uses rows to represent arcs, but the columns correspond to paths within the network. The matrix entries, a_{ij}, are either a 1 which implies arc i is on path j, or 0 meaning arc i is not on path j (23:7).

2.2 Expected Maximum Flow

To calculate the exact value of the expected maximum flow requires complete enumeration of all possible network states of which there are a total of 2^n. This problem is classified as NP-hard meaning that there currently exists no algorithm that can calculate the expected maximum flow in polynomial time or in other words, that the computations required grow exponentially with the number of components within the stochastic network.

In contrast to the exact value of the expected maximum flow, the lower bound, sometimes referred to as the maximum expected flow, can be efficiently calculated. Yim (23), assuming independent arcs and no rerouting, developed a formulation to calculate the maximum expected flow (lower bound). This formulation uses an arc-path incidence matrix representation a [illegible] [illegible]ximizes the sum of the product of path reliabilities and flows. A description of [illegible] (23) formulation follows.

2.2.1 Lower Bound Formulation The reliability R_j of path A_j within the network can be computed by

$$R_j = \prod_{i \in A_j} p_i$$

Now let f_j be the flow on path j. The sum of the expected flows on all the paths from source to sink is given by

$$\sum_{j=1}^{q} R_j f_j$$

Letting a_{ij} equal 1 if arc i lies on path j, and 0 otherwise, the lower bound formulation is as follows:

$$\text{Max } \sum_{j=1}^{q} R_j f_j$$

s.t

$$\sum_{j=1}^{q} a_{ij} f_j \leq u_i \text{ for } i = 1, 2, \ldots, n$$
$$f_j \geq 0$$

Yim uses the notation u_i to represent the capacity of arc i (23:16-18).

To facilitate the maximum expected flow calculation, Yim (23) developed a computer program, *Formula*, using the artificial intelligence language PROLOG, to generate the network flow paths. An input file was then generated and input into a mathematical programming package for solution. His results were validated against exact solutions generated by Bailey who used simulation and response surface analysis on the same experimental networks (24:18-20). Yim states " Even though the lower bound estimate ignores re-routing, the states considered are prominent enough to adequately represent the expected throughput [illegible] apparently much more accurate than the assumption of no link failure in the upper bound (no failures) formulation." (24:18)

2.3 Capacity Improvements

To improve the expected flow through a stochastic network, either the arc capacities or survivabilities must be increased. In this section , Yim's (23) investment strategy model for increasing throughput by increasing arc capacity will be reviewed while the next section will examine Gaught's (12) investment strategy model using increased arc survivabilities to increase network throughput.

Yim developed a linear programming model which maximizes the lower bound of expected maximum flow given a fixed budget for increasing arc capacities. The model assumes no physical upper limit for arc capacity improvements and equal costs for increasing arc capacities by one unit. The solution obtained from the model identifies which arcs and by how much each should be increased.

The model is similar to that of the lower bound formulation, but with the following modifications. An increased arc capacity term, d_i, is included in the flow capacity constraint, and a budget constraint is introduced, where α is the cost of increasing arc i by one unit and β is the total budget available. The model is as follows:

$$\text{Max} \sum_{j=1}^{q} R_j f_j \qquad \text{where } R_j = \prod_i P_i$$

s.t

$$\sum_{j=1}^{q} a_{ij} f_j \leq u_i + d_i$$

$$\sum_{i=1}^{n} \alpha_{ci} d_i \leq \beta$$

$$R_j, u_i, d_i, f_j, \alpha_{ci}, \beta \geq 0 \text{ and } a_{ij} = 0, 1$$

2.4 Survivability Improvements

Gaught (12) developed an investment strategy model to increase the expected maximum flow through the network by increasing the survivabilities of the arcs. He introduced three additional assumptions to those made in the lower bound model; 1) all arc survivabilities had the potential to be increased to one, 2) arc survivabilities will be increased in increments of one tenths, and 3) the costs to increase arc survivabilities were equal and linear.

The objective function maximizes the sum of the expected flows over all paths from source to sink and is

$$\text{Max}\sum_{j=1}^{q} R_j f_j$$

In Gaught's model, however, the path reliability R_j is not a constant but a product of the arc survivabilities, which are dependent on the increased survivability factor X_i. Thus the path reliabilities are nonlinear and are

$$R_j = \prod_i (P_i + .1 \cdot X_i)$$

The constraints consist of a maximum arc survivability constraint, a flow capacity constraint, and a budget constraint. The first constraint ensures the arc survivabilities (original P_i plus increase $.1 \cdot X_i$) are less than or equal to one. The second constraint ensures the arc flow is less than or equal to the arc's capacity u_i. Finally, the third constraint limits the sum of the costs to increase arc survivabilities, α times X_i, to less than or equal to the total budget available, β.

Gaught's (12) nonlinear model is as follows:

$$\text{Max } \sum_{j=1}^{q} R_j f_j \quad \text{where } R_j = \prod_i (P_i + .1 \cdot X_i)$$

s.t.

$$P_i + .1 \cdot X_i \leq 1 \quad i = 1, 2, \ldots n$$
$$\sum_{j=1}^{q} a_{ij} f_j \leq u_i$$
$$\sum_{i=1}^{n} \alpha_{ri} X_i \leq \beta$$
$$X_i, P_i, f_j, \alpha_{ri}, \beta \geq 0 \text{ and } a_{ij} = 0, 1$$

2.5 *Reliability*

Reliability is a useful measure to analyze the performance of networks. The traditional approach used in evaluating network reliability considers some measure of network connectivity, usually source to sink or interconnectivity between k specified nodes. A second approach considers the ability of the network to transmit a required flow from source to sink. Aggarwal (2:184-186) integrates network capacity with reliability to define a network reliability performance index as the probability of successfully transmitting a required flow from source to sink.

Aggarwal's approach (2:184-186) uses a normalized weighting scheme coupled with the traditional $s - t$ reliability methodology. The success states of the network (considering path availability only) are identified and the associated state probabilities are calculated. Aggarwal then multiplies the state probabilities by their normalized weights based on the states maximum flow. The normalized weight is defined as ratio of the state's maximum flow to the original network's maximum flow. The weighted state probabilities are then summed to form the performance index. Aggarwal uses the following notation (2:184):

s, t	source, terminal node
p_x, q_x	reliability, unreliability of branch x
T	set of all states
S, F	set of all success, failure states
S_i	a member of set S
P_{S_i}	probability of system being in state S_i
C_i	capacity of the subnetwork in state S_i
w_i	normalized weight for state S_i
C_{max}	capacity of the network with all branches up
R_{st}	s - t reliability
PI	performance index

For a network where the arcs have only two states, up or down, and the nodes are perfect, the set of all states T, consist of 2^n different states where n is the number of arcs. From this total set, a subset S, needs to be determined corresponding to those system states that have at least one complete path from s to t.

For each state S_i ($S_i \in S$), the following are defined:

α_i - arc i is up

β_i - arc i is down

The probability of state S_i is:

$$P_{S_i} = \prod_{i \in \alpha_i} p_i \prod_{i \in \beta_i} q_i$$

The traditional s - t reliability is:

$$R_{st} = \sum_{(S_i \in S)} P_{S_i}$$

Aggarwal defines the normalized weight w_i as:

$$w_i = \frac{C_i}{C_{max}}$$

Then the weighted reliability or performance index, PI is:

$$PI = \sum_{(S_i \in S)} w_i P_{S_i}$$

Aggarwal's performance index requires the enumeration of all network success states. Due to the binary nature of the arc states, up or down, an efficient algorithm based on boolean algebra variables (literals) has been developed by Brown to determine those success states of the network out of the set of total states. (8:121-124)

2.5.1 Solved Network. The network in Figure 1 was used for an example application of Aggarwal's reliability performance index using Brown's algorithm to generate the network success states.

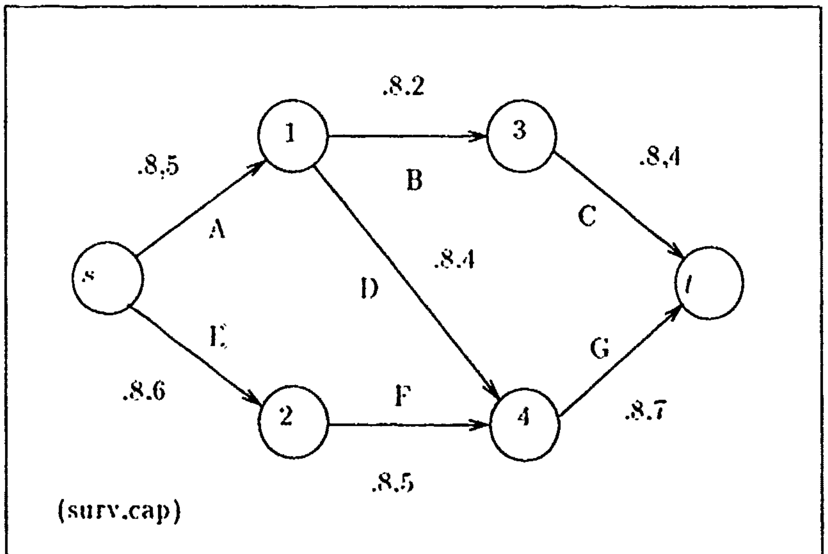


Figure 1. Example Network

The example network consist of seven arcs and three $s - t$ paths, A-B-C, A-D-G, and E-F-G. Using Brown's algorithm (8:121-124), thirty-nine network success states were determined from a possible 128 states and are listed in Table 1.

The probability of each success state was calculated and the $s - t$ or connectedness reliability was obtained by summing the success state probabilities. For the example network, the result was 0.8282. Another reliability measure would be a maximum flow reliability index R_{fc}, where only those success states that have a flow capacity equal to the maximum are considered. For the sample network, only one state, where all arcs are operating, can provide the maximum flow of nine units. The maximum flow reliability index was calculated to be 0.2097.

Given the two above reliability indicators, there still existed a large gap of network flow information between them. Aggarwal's performance indicator filled in this gap where:

$$R_{fc} < PI < R_{st}$$

The PI for the required flows C_R less than C_{max} for the example network are as follows where the definition of w_i is modified as:

$$w_i = \begin{cases} C_i/C_r & \text{if } C_i < C_r \\ 1 & \text{otherwise} \end{cases}$$

Flow Required	PI
$7 < C_R \leq 9$	.5398
$5 < C_R \leq 7$	.6341
$4 < C_R \leq 5$	.7358
$2 < C_R \leq 4$	.7622

The expected maximum flow for the network can also be calculated by summing the state probabilities times the capacity or maximum flow of that state. For the above network, the expected maximum flow was 4.78. Yims lower bound for the example network was 4.69.

Aggarwal's reliability performance index provides valuable insight into the flow capabilities of stochastic networks. The information though comes at a cost, that of computational explosion due the exponential nature of the state space. While Brown has provided an algorithm for determining the network success states, they still number a large fraction of the total states and become very large for any realistic network. The algorithm also requires enumeration of all the source to sink paths. Due to an artificial intelligence based path enumeration program written by Yim (23), the path enumeration part can be generated. The second difficulty with Aggarwal's method is determining the flow capacity for all the success states. While there are numerous efficient algorithms and commercial network packages to solve for the maximum network capacity, again the calculations required for all the states becomes computationally infeasible for large networks.

Aggarwal's approach for examining the flow reliability of a stochastic network has been demonstrated on a small sample network to establish a base from which to improve upon. The network performance information generated is valuable, but improvements in decreasing the computational explosion need to be investigated. Yim has developed an efficient bounding model for the expected maximum flow. Coupling this with an efficient $s-t$ reliability measure will give two performance measures that although not as complete as Aggarwal's measure, can give the decision maker a basis for a better understanding of the network performance.

2.5.2 Alternatives to Complete Enumeration. Provan and Ball have shown that virtually all network reliability problems are NP-hard (19). Complete enumeration of the success states can provide varying insights into the performance of a network including $s-t$ reliability, performance indexes, and expected maximum flow. Several methodologies exist however, based on certain key structures within the network, to measure reliability based on other than states. Alternative methodologies based on states, paths, and arcs will be evaluated in measuring the $s-t$ reliability. See discussions in Chapter IV regarding probable states, disjoint paths, and factoring.

Table 1. Network Success States

State Number	Element States A	B	C	D	E	F	G	State Capacity	State Probability P_{S_i}	Expected Flow
1	0	0	0	0	1	1	1	5	.0008192	.0040960
2	0	0	0	1	1	1	1	5	.0032768	.0163840
3	0	0	1	0	1	1	1	5	.0032768	.0163840
4	0	0	1	1	1	1	1	5	.0131072	.0655360
5	0	1	0	0	1	1	1	5	.0032768	.0163840
6	0	1	0	1	1	1	1	5	.0131072	.0655360
7	0	1	1	0	1	1	1	5	.0131072	.0655360
8	0	1	1	1	1	1	1	5	.0524288	.2621440
9	1	0	0	0	1	1	1	5	.0032768	.0163840
10	1	0	0	1	0	0	1	4	.0008192	.0032768
11	1	0	0	1	0	1	1	4	.0032768	.0131072
12	1	0	0	1	1	0	1	4	.0032768	.0131072
13	1	0	0	1	1	1	1	7	.0131072	.0917504
14	1	0	1	0	1	1	1	5	.0131072	.0655360
15	1	0	1	1	0	0	1	4	0032768	0131072
16	1	0	1	1	0	1	1	4	.0131072	.0524288
17	1	0	1	1	1	0	1	4	.0131072	.0524288
18	1	0	1	1	1	1	1	7	.0524288	.3670016
19	1	1	0	0	1	1	1	5	.0131072	.0655360
20	1	1	0	1	0	0	1	4	.0032768	.0131072
21	1	1	0	1	0	1	1	4	.0131072	.0524288
22	1	1	0	1	1	0	1	4	.0131072	.0524288
23	1	1	0	1	1	1	1	7	0524288	.3670016
24	1	1	1	0	0	0	0	2	.0008192	0016384
25	1	1	1	0	0	0	1	2	.0032768	0065536
26	1	1	1	0	0	1	0	2	.0032768	.0065536
27	1	1	1	0	0	1	1	2	.0131072	.0262144
28	1	1	1	0	1	0	0	2	.0032768	.0065536
29	1	1	1	0	1	0	1	2	.0131072	.0262144
30	1	1	1	0	1	1	0	2	.0131072	0262144
31	1	1	1	0	1	1	1	7	.0524288	.3670016
32	1	1	1	1	0	0	0	2	0032768	.0065536
33	1	1	1	1	0	0	1	5	.0131072	.0655360
34	1	1	1	1	0	1	0	2	.0131072	.0262144
35	1	1	1	1	0	1	1	5	.0524288	.2621440
36	1	1	1	1	1	0	0	2	.0131072	0262144
37	1	1	1	1	1	0	1	5	.0524288	.2621440
38	1	1	1	1	1	1	0	2	0524288	.0262144
39	1	1	1	1	1	1	1	9	.2097152	1.8874368
R_{st}									.8282112	4.780032

2.6 Multicriteria Decision Analysis

Many real-world decisions involve conflicting multiple objectives, attributes, goals, and criteria which can not be effectively reduced into a single aggregate criterion (25:xii). Multi-criteria decision analysis (MCDA) combines the quantifiable aspects of mathematical optimization with the qualitative aspects of a decision makers (DM) preferences to arrive at a best compromise solution for the multiple objective optimization problem (MOP) (20:5-8).

Due to the conflicting nature of the objectives, several solutions, called efficient solutions, are obtained. An efficient solution is one where an improvement in any other objective would degrade one or more of the other objectives (11:1270). The efficient frontier consist of all the efficient solutions. This set is also known as the nondominated,noninferior,or pareto optimal solutions (21:98). The best compromise solution is that efficient solution which maximizes the DM's value or preference function, assuming more of each objective is preferred to less (11:1270).

In addition to an optimization methodology, some scheme to obtain and formulate the preference function is required. MCDA methods can be classified into three categories based on the timing of obtaining the DM's preference structure relative to the optimization (11:1271):

1. prior to the optimization (priori)
2. after the optimization (posteriori)
3. during or in sequence with (progressive)

The first approach requires interviews with the DM prior to optimization to formulate the preference function. The determination of the function may require a prohibitive amount of time and effort accompanied by the DM's difficulty in expressing the required information (11:1271)

The second approach generates all or most of the efficient solutions, presenting them to the DM for a most preferred selection. There are three problems associated with this approach; first, the material presented to the DM may be difficult to understand; second, many real problems are too large to solve for most of the efficient solutions; and third, the number of efficient solutions presented is too large for the DM to analyze effectively (11:1273). These methods may, however, be useful in generating efficient inputs for the third approach discussed below (21:98).

In the third approach, the DM interacts with the optimization process providing preferences in an iterative manner. This approach is also known as an interactive multiple objective optimization. In 1982, Zeleny stated "Some approaches, although promising and potentially very useful, are not yet sufficiently developed and theoretically grounded to warrant a serious review (for example, interactive programming approaches)." (25:xii) In 1988, The CONDOR report stated "Interactive software will require more realistic representation of decision maker preference that reflect lack of surety and preference changes that arise as interaction proceeds." (10:624) In 1991, however, Shin and Ravindran suggest that the interactive methods are considered "promising" and comparative studies indicate superiority over the other two approaches (21:98).

Shin and Ravindran (21) divide the interactive methods into the following categories: feasible region reduction methods; feasible direction methods; criterion weight space methods; tradeoff cutting plane methods; Lagrange multiplier methods; visual interactive methods; branch-and-bound methods: relaxation methods; and other important methods. Each will be briefly reviewed referencing the Shin and Ravindran Survey (21) which is the most current and extensive survey on the subject.

2.6.1 Feasible Region Reduction Methods. Each iteration consist of a calculation phase, a decision phase, and a feasible region reduction phase. In the calculation phase, an *ideal solution* is obtained from the nearest efficient solution with respect to given weights. The DM's responses in the decision phase are used to create additional constraints which reduce the feasible region. The iterations continue until the DM feels the current solution is the best compromise solution (21:100).

2.6.2 Feasible Direction Method. This is an iterative method which starts with a feasible solution and proceeds to a more preferred solution using a direction and step-size. based on the DM's preferences. The pioneering method is the GDF procedure by Geoffrion which is a modified Frank Wolfe method. Several extensions have been developed to improve the direction finding step or the imbedded line search (21:101,102).

2.6.3 Criterion Weight Space Method. This method combines the objectives into a single objective by using weighting. A set of tradeoffs associated with the current single weighted objective is presented to the DM whose responses are used to create additional constraints on the weights and generate a new efficient solution. The iterations end when the DM is satisfied with the current solution (21:102,104).

2.6.4 Tradeoff Cutting Plane Method. This approach is a variation of the feasible direction method. Cutting planes are used iteratively to reduce the feasible region eliminating the need for a line search. Several modifications have been developed to reduce the DM's burden in the interactive process (21:104).

2.6.5 Lagrange Multiplier Methods. These methods use the Lagrange multipliers of the constraints, resulting from the maximization of one objective relative to the bound set of objectives, to form tradeoff functions. Interactions with the DM are then used to generate shadow prices of the bound objectives and thus define a surrogate worth function (21:104,106).

2.6.6 Visual Interactive Methods. These methods use graphic-aided interactive approaches as an extension of the GDF method. A graphic of a subset of the efficient frontier is generated each iteration for interaction with the DM. This allows the DM to control the efficient frontier through the interactive process. Shi states that with advanced personal computers, this approach is becoming popular (21:106).

2.6.7 Branch-and-Bound Method. This method divides the objective space into subsets where each subset is a branch and further branching can occur if it has promise. An ideal solution is determined at each branch and is used to form an upper bound for that branch. Each solution is compared to the incumbent solution through interaction with the DM. A branch is fathomed if dominated by the incumbent solution (21:107).

2.6.8 Relaxation Method. This method generates a master LP problem equivalent to the original problem and is solved iteratively by relaxation of some constraints. At each iteration, bounds of the objective functions are presented to the DM for interaction (21:107).

2.6.9 Other Methods. Other lesser used but important methods to consider include sequential methods, scalarizing function methods, fuzzy methods, and statistical methods (21:107,108).

The effectiveness of the methods depends on problem structure and the characteristics of the methodology selected (21:108). The characteristics to consider are interaction style, solution approach, applicability, and mathematical programming required (21:109). In addition, the methods should be evaluated with respect to the DM's cognitive burden, ease of actual use, effectiveness in the decision-making process, and the ability to handle DM inconsistencies (21:110).

Buchanan and Daellenbach (9) performed a comparative evaluation of four interactive methods in 1987. They concluded that, "DM's seem to prefer solution methods where they are in control, and where they are allowed to backtrack and change previous inconsistent decisions." (9:358) In addition, they suggest that the DM can reasonably demand the information presented be in a high quality graphics form (9:358).

2.7 Summary

Methodologies exist to calculate the maximum expected flow (lower bound) and reliability for stochastic communication networks. Separate investment strategy models have been developed to improve the network flow by increasing either arc capacities or survivabilities. In addition, the development of an artificial intelligence program to enumerate the paths within the stochastic network decreases the time and effort required for the above calculations. Finally, numerous interactive multiple criteria decision analysis tools are available for application to the problem of analyzing trade-offs in improving the performance of stochastic communications networks.

III. Methodology

An overall research plan was accomplished in order to fulfill the research objectives outlined in Chapter I. The research plan consist of the following steps 1) understanding the problem, 2) formulating the reliability performance measure 3) formulating the reliability improvement model, 4) formulating the multicriteria model, 5) selecting software to implement and integrate the models, and 6) analyze communication networks.

3.1 Understanding the Problem

The performance of stochastic networks can be evaluated using several parameters. This research concentrated on two specific performance parameters, source to sink expected maximum flow and source to sink reliability. Both required methodologies to measure and improve each parameter, after which tradeoff methodologies were applied. For this research, the tradeoff was between improving network expected maximum flow through increased arc capacities and survivabilities, and increasing network reliability through increased arc survivabilities.

The calculation of the two performance parameters for large stochastic networks was not an easy task due to the exponential nature of the calculations. In general, the calculation of the exact performance values is NP-Hard meaning there exists no known algorithm that can solve for the value in polynomial time. Faced with the NP nature of the problem, three general types of methodologies have been applied by researchers in calculating the performance parameters. The first is simulation, which models the network and given a large number of samples will calculate a statistically significant value that approximates the exact value. To get an accurate approximation though requires a very large number of samples with a correspondingly large run time. The simulation approach was not considered for this research, although previous simulation work done by Bailey (5) was used as reference for the network analysis portion of this research. The second approach has been to bound the exact value, thus calculating an upper and lower bound for the exact performance measure value. This approach reduces the number of states considered and thus may be computionally feasible given the bounds are sufficiently tight to be useful as a performance measure. The third approach is to increase the efficiency of nonpolynomial algorithms to the point that the required run time and memory requirements are sufficiently reduced so as to meet the time and memory constraints of the problem to be solved. The last two of the above three methodologies will be examined with respect to network expected maximum flow and reliability performance models.

3.2 Formulating the Reliability Performance Model

The calculation of the reliability performance measure can be accomplished through four general techniques. The first using complete state enumeration becomes computationally infeasible for even medium size networks but can provide valuable insights when applied to small models. The other three techniques consist of the most probable states, disjoint products, and factoring. Each will be investigated for its applicability to the research problem and presented in Chapter IV.

The expected maximum flow performance measure will be calculated using Yim's lower bound formulation which calculates a sufficiently tight lower bound using path enumeration as discussed in the literature review.

3.3 Formulating the Reliability and Capacity Improvement Models

Once the performance measure models were selected, methodologies to allow investment for improving the measures needed to be developed. Gaught (12) had developed an investment strategy which increased the lower bound of the expected maximum flow by increasing arc survivabilities. While this had a side effect of increasing network reliability, it did not maximize network reliability. The most desired improvement model would be one that maximized a reliability mathematical expression. Due to the reliability methodology selected, a mathematical expression was not generated for use as a reliability improvement objective function. In addition, a reliability expression for large networks would result in an exceedingly long nonlinear expression requiring large amounts of memory to store and optimize, if at all possible. Thus a linear heuristic methodology by Jain and Gopal (15) was used which did not require a reliability expression to maximize the network reliability. The methodology orders the network arcs with respect to their potential contribution to the overall network reliability. In addition to not requiring a reliability expression, the methodology was linear in nature, greatly reducing the computational complexity and avoiding the local optimum problems associated with Gaught's nonlinear improvement model. Jain and Gopal's method will be described in Chapter IV.

As with the expected maximum flow calculation, Yim's capacity improvement model will be used, which was examined in the literature review.

3.4 Formulating the Multicriteria Model

Given an objective function for each criteria, expected flow and reliability, a multicriteria optimization model was needed to generate the trade-off region or efficient frontier for the two criteria. Lacking a criterion function for network reliability, two modified approaches to examine the trade-off reqion were investigated in Chapter V.

3.5 Selecting Software

Given the performance measure and improvement models, software was required to implement the models. Previous work by Yim and Gaught used several different off-the-shelf commercial solver packages to include GINO and LP/MIP-83. The General Algebraic Modeling System, *GAMS*, was selected both for its ability to solve linear and nonlinear models on the personal computer and its ease in converting from one model to another. In addition to the *GAMS* solver, Yim's *Formula* program was used for network path generation and which required a Prolog compiler. Finally, integration procedures and input file generation programs were written in Pascal.

3.6 Analyzing Communication Networks

The above methodologies were applied to one small example network and three large realistic networks. The example network was used to demonstrate the application of the models to stochastic networks while the large networks were analyzed with respect to their initial performance and subsequent improved performance. Finally, the trade-off region was examined for the networks. The description of the three large networks along with their respective analysis is presented in Chapter VII.

IV. Reliability

4.1 Introduction

Several techniques exist to measure the reliability of stochastic networks. The three major techniques appearing in the literature were examined to determine which was best suited for this research. The first technique examined was that of disjoint products where the network paths are used to generate disjoint terms for which the sum of all terms is the exact network reliability value. The second technique involved the network's most probable states where only those states that contribute the most to the network reliability are used, thus generating an upper and lower bound to the exact reliability value. Finally, the recursive technique of factoring in conjunction with reduction techniques was examined in the calculation of the exact reliability value. The three techniques were applied to Figure 2 for illustrative as well as evaluative purposes.

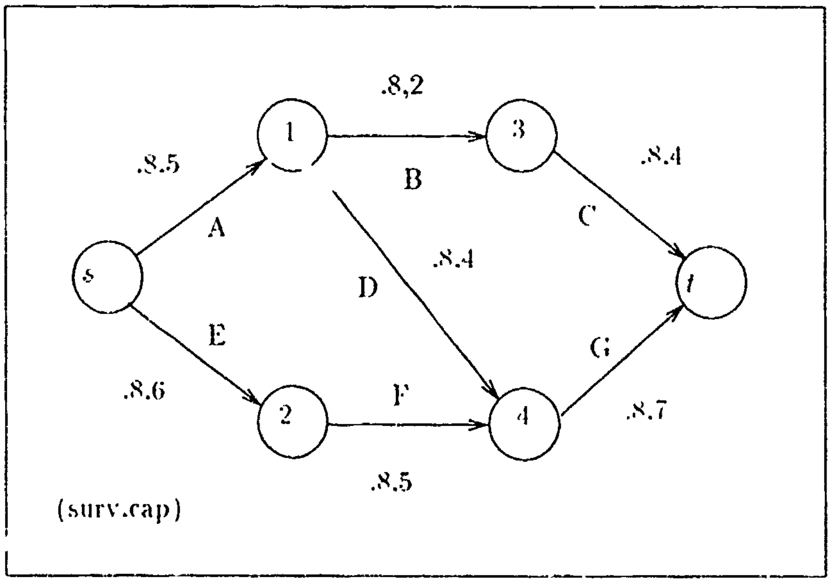


Figure 2. Example Network

4.2 Disjoint Product Algorithms

Given the m success paths $P_1, P_2, \cdots, P_m$ are known, the system success can be described as:

$$S = P_1 \cup P_2 \cup \cdots \cup P_m$$

If the paths are made disjoint (mutually exclusive), then the $s - t$ reliability can be described by:

$$R_{st} = Pr(S) = Pr(P_1 \cup P_2 \cup \cdots \cup P_m)$$

Aggarwal (3:83-85) presents an algorithm that makes the success paths disjoint thus allowing for the exact calculation of the $s - t$ reliability. Abraham (1:58-61) presents an improved algorithm that provides a reliability expression where all paths are made disjoint. Abraham's algorithm will be applied to the same example network used in Chapter II. The paths are first ordered by the number of arcs within the paths. In this case all three paths have three arcs so the first to be considered will be path A-B-C. Since it is the first path to be considered, it does not need to be made disjoint with others, thus generates only one product term, $p_a p_b p_c$. The second path considered is E-F-G. Three product terms are generated to make it disjoint with the first path. These are $q_a p_e p_f p_g$, $p_a q_b p_e p_f p_g$, and $p_a p_b q_c p_e p_f p_g$. Finally the third path, A-D-G, was made disjoint with the previous two, generating four additional terms, $p_a q_b p_d q_e p_g$, $p_a q_b p_d p_e q_f p_g$, $p_a p_b q_c p_d q_e p_g$ and $p_a p_b q_c p_d p_e q_f p_g$. The total reliability expression is as follows:

$$R_{st} = p_a p_b p_c + q_a p_e p_f p_g + p_a q_b p_e p_f p_g + p_a p_b q_c p_e p_f p_g + p_a q_b p_d q_e p_g + p_a q_b p_d p_e q_f p_g + p_a p_b q_c p_d q_e p_g + p_a p_b q_c p_d p_e q_f p_g$$

The resulting reliability from the above eight term reliability expression was .8282, which agrees with the results obtained by complete enumeration of the 39 success states in Chapter II. In his paper, Abraham presents the results of his algorithm on a network consisting of 12 arcs and 24 paths. The resulting reliability expression contains 71 terms compared to 4096 total states for the network. While this algorithm generates the exact reliability value, it can also be used to calculate a lower bound by utilizing those terms with the least number of complemented (q_i) terms. For the example network used in this paper, the first four terms using only one complemented arc will produce a lower bound reliability of .761856 which is within 10 percent of the exact value. In Abrahams network, using those terms with up to two complemented arcs results in 14 terms producing a lower bound of .8369 which is again within ten percent of the exact reliability value.

Given the same number of terms used, the closeness of the lower bound to the exact value will decrease with lower arc survivabilities. For the example network used in this paper, if all arcs had a survivability of .6 rather than .8, the lower bound would be .385344 compared to an exact value of .4738176. This results in a lower bound within 20 percent compared to the previous ten percent result when arc survivabilities were .8.

Recent work by Locks (17) has made further efficiency improvements to the disjoint product technique by using rapid inversions in place of search operations. For Abrahams network of 24 paths, Locks generates 60 disjoint product terms compared to Abraham's 71. Even further improvements have been made by Heidtmann (13) who, for the same network, generated only 41 disjoint product terms. While Heidtmann's algortithm is the most efficient, the number of disjoint product terms still grows exponentially with the size of the network. Thus for large networks, a reliability expression may contain thousands of terms with each term containing tens of products, creating an extremely large nonlinear expression.

4.3 Most Probable State Enumeration

For large networks, calculating the exact reliability value becomes computationally prohibitive. Rather than enumerating all states, Li and Sylvester (16:1105-1110) consider only the m most probable states to compute lower and upper bounds on network performance measures, to include $s-t$ reliability. As shown in Chapter II, the probability of state k is given by the product of the arc success probabilities, p_i, and the arc failure probability q_i which is defined as $1 - p_i$.

Li and Sylvester rename the arcs with an R_i term where $R_i = q_i/p_i$. With the above renaming. the state probabilities can be calculated by

$$P_{S_k} = \prod_{i=1}^{n} p_i(q_i/p_i)^{T_i(S_k)}$$

or

$$P_{S_k} = \left(\prod_{i=1}^{n} p_i\right)\left(\prod_{j=1}^{n} R_j^{T_j(S_k)}\right)$$

where

$$T_i(S_k) = \begin{cases} 0 & \text{if arc } i \text{ operates in state } S_k \\ 1 & \text{otherwise} \end{cases}$$

The most probable state, S_1 is when all arcs are operational. and the next most probable states are those with only one arc failure. Li and Sylvestor (16) developed an algorithm called *Order* which by using R_i where $R_1 \geq R_2 \geq \cdots \geq R_n$, orders the m most probable states.

Li and Sylvester (16:1106,1107) also include a method to estimate the number of states required to achieve a specified coverage of the state space where f_L is defined to be the probability associated with the states considering up to L arc failures per state:

$$f_L = \sum_{k=0}^{L} \begin{pmatrix} n \\ k \end{pmatrix} p^{n-k} q^k$$

This equation assumes that the p values and thus the resulting q values are the same for all arcs. If arc survival probabilities are different. selecting a p value equal to the lowest arc survivability value will generate a lower bound on the state space covered.

Given the m most probable states, The lower and upper bounds for s--t reliability can be calculated where Li and Sylvestor use a reliability or connectivity performance measure $C(S_k)$ where

$$C(S_k) = \begin{cases} 1 & \text{if arc } i \text{ the network is connected in } S_k \\ 0 & \text{if not} \end{cases}$$

and thus

$$C_L(m) = \sum_{k=1}^{m} P(S_k)C(S_k)$$

$$C_U(m) = \sum_{k=1}^{m} P(S_k)C(S_k) + (1 - \sum_{k=1}^{m} P(S_k))$$

In other words, the lower bound is the sum of all the connected m states and the upper bound is one minus the sum of all the disconnected m states. The upper and lower bound converge on the exact reliability value. When all possible states are considered, the upper and lower bounds equal the exact reliability value. For the example network with arc survivabilities equal to .8, the upper and lower bounds were computed considering only states with up to two arc failures. The lower bound is .7995392 and the upper bound is .934464.

For large networks (50 arcs), considering the most probable states up to 3 arc failures, 20,875 states will be used in the calculations. With arc survivabilities of .9, the covered state probability is 24 percent. In other words, the uncertainty between the lower and upper bounds is 76 percent. If arc survivabilities are lowered to .8 and up to five arc failures are allowed per state, over 2 million states will be used with a resulting uncertainty of 95 percent. This demonstrates the usefullness of this approach is limited to networks with arc survivabilities greater than .9.

4.4 *Factoring*

The factoring theorem of reliability states that the reliability of a binomial system S can be decomposed with respect to the probability of the two possible states (up or down) for a selected arc :

$$R(S) = p_i R(S|i \text{ up}) + q_i R(S|i \text{ down})$$

The decomposed reliability expression can be applied recursively with reduction techniques applied within the recursions. Eventually the network is reduced to a simple structure for which the reliability can be easily calculated. The complexity of factoring algorithms is dependent on the selection rules for which arcs are to be factored (22).

Page and Perry (18) have developed a recursive computer program, ***dirprog***, based on factoring in conjunction with reduction techniques that efficiently solves the $s - t$ reliability for large directed stochastic networks. See Appendix F for a Pascal source code listing.

Network reduction techniques require only polynomial time yet by reducing the size of the network, thereby reducing the state space, tend to reduce the exponential growth of a factoring algorithm's backtrack search structure (22:272). Page and Perry summarize the network reduction techniques used as follows (18:558,559):

1. Remove arcs directed into the source or out of the sink. They are irrelevant.
2. Remove dead-end and false-start nodes. A dead-end node is a node other than the sink having no arc directed outward from the node. A false-start node is a node other than the source having no arc directed into the node.
3. If there is a single arc out of the source or into the sink, contract the arc (making the adjacent node into the new source or sink). The reliability of the original network is the reliability of the reduced network multiplied by the survivability of the contracted arc. This is the only reduction that generates a multiplying factor
4. If there is a single arc directed into or out of a node other than the source or sink, then the anti-parallel arc (if it exists) can be removed.

5. Series arc reductions. Two arcs with a common node for a head and tail with survivabilities p_1 and p_2 can be replaced by a single arc with survivability p_1p_2.
6. Parallel arc reduction. Two parallel arcs (arcs joining the same two nodes and having the same orientation) with survivabilities p_1 nd p_2 can be replaced by a single edge having the same orientation as the original edges and having a survivability of $p_1 + p_2 - p_1p_2$.

For the example network, *dirprog* was run with the following results. A network reliability of .8282 was calculated, requiring 3 source/sink reductions, 4 chain vertex reductions, and 1 factoring. The computation time required was less than .01 seconds.

4.5 Evaluation

The disjoint product technique can be used to solve for the exact reliability value but for large networks. is limited by the amount of memory available to store the disjoint products. A lower bound approach can be used to limit the number of disjoint products but the effectiveness of this approach is dependent on the arc survivabilites with high arc survivabilities lending to increased efficiency. Lower arc survivabilities, as is seen in this research, would not benefit as much from this bounding technique. In addition, a lower bound without an associated upper bound leaves a window of uncertainty from the lower bound up to a reliability value of one for where the exact reliability value lies.

The most probable state enumeration method can be used to calculate a lower and upper bound for $s - t$ reliability. While Li and Sylvester have an efficient algorithm to order the most probable states, the effectiveness is again dependent on the arc survivabilities. In addition to determining the most probable states, an efficient means to separate those states into connected and non-connected states is required for the reliability bounding calculations. It is also seen that the efficiency for calculating the bounds decreases with lower arc survivabilities where as shown, arc survivabilities below .9 greatly increase the computations required to make this method computationally feasible for large networks. Generally, these methods are applicable when applied to reliability problems where the arc reliabilities are .9 or greater. For large network survivability problems where arc survivabilities are often less than .9, such as in this research, the most probable state method is not applicable.

The factoring technique in conjunction with reduction methods is very effective in calculating the reliability of large directed networks encountered in this research. The networks lend themselves to available reduction methods used by Page and Perry and the recursive nature of their program enables large networks to be run on personal computers up to a point where the MS DOS stack limit of 64K is reached. For larger networks, a modified version of *dirprog* running under a VMS operating system was written.

4.6 Reliability Improvement Model

The network reliability improvement model is based on a linear heuristic method developed by Jain and Gopal (15). Their method calculates an important index, II_j, and thus a ranking for each arc in the network. The arcs with the highest important index are the most vital, contributing the most to the network reliability and thus are assigned the highest reliability. The important index is based on the following two observations by Jain and Gopal (15).

1. Low cardinality $s-t$ paths contribute more to network reliability than high cardinality paths and so are more important.
2. The higher the frequency of occurrence of an arc in a particular cardinality path set, the more important it is.

The importance index II_j is defined as:

$$II_j = \sum_i \frac{f_{ij}}{NP_i C_i^2}$$

where the following notation applies

- f_{ij} = frequency of occurrence of arc j in $s-t$ paths of cardinality C_i

- NP_i = number of $s - t$ paths of cardinality C_i

Jain and Gopal (15) outline the following four steps to determine the important indices:

1. Generate all $s - t$ paths.
2. Arrange all paths in groups in order of increasing cardinality. Count the number of paths (NP_i) in each cardinality (C_i) group.
3. Find, for all j, the frequency of occurrence f_{ij} of arc j in each group of paths.
4. Determine II_j for all j.

Using *Formula* to generate the paths through a depth first search process. the important indices can be calculated using the above procedure. A linear mathematical model was then developed to improve the arc survivabilities with the highest index, subject to a maximum arc survivability of one and a finite improvement budget.

The objective function maximizes the sum of the important indices over all arcs X_j given each arc has an initial survivability P_j. The two type of constraints ensure that each arc has a survivability upper bound of one and that the total amount spent on arc survivability improvements. the sum of increasing arc survivabilities by .1. α_j. is less than or equal to the total budget available β. The model is as follows:

$$\text{Max} \sum II_j(P_j + .1 \cdot X_j)$$

s.t.

$$P_j + .1 \cdot X_j \leq 1 \qquad j = 1.2.\ldots.n$$

$$\sum_{j=1}^{n} \alpha_j X_j \leq \beta$$

$$X_j \geq 0$$

A Pascal program was written to calculate the important indices for a stochastic network (see Appendix G).

V. Multicriteria Optimization

The traditional multicriteria optimization (MCO) problem requires two or more criteria or objectives (the Y space), both of which utilize a common set of alternatives from the X space. For the problem of stochastic network performance improvement, there were two criterion objectives examined in this research. The first maximized the expected throughput or flow while the second maximized the network reliability or connectedness. The alternative set X consisted of arc survivability improvements and arc capacity improvements. This chapter describes a MCO model given both criterion functions . For large networks though. the network reliability in this research was calculated using a recursive technique which, in turn, did not result in a reliability objective function. Subsequently, two modified approaches were developed to analyze the tradeoffs between the two criterion objectives.

5.1 MCO Model

The MCO model consists of two criterion functions, CF_1 and CF_2. The first represents the lower bound for the network expected flow where the lower bound is maximized by increasing both the path reliability, R_j through arc survivability improvements. X_i, and the path flow. f_j through arc capacity improvements. d_i. The first criterion function CF_1. is equal to Gaught's objective function for his nonlinear models and is

$$\text{Max } \textstyle\sum_{j=1}^{q} R_j f_j \text{ where } R_j = \prod_i (P_i + .1 \cdot X_i)$$

The second criterion function CF_2. also nonlinear. maximizes the network reliability. For the example network. Figure 3. the second criterion function is as calculated in Chapter IV using Aggarwal's disjoint path method and is

$$
\begin{aligned}
CF_2 = \quad & (P_1 + X_1)(P_2 + X_2)(P_3 + X_3) \\
+ \quad & (1 - (P_1 + X_1))(P_5 + X_5)(P_6 + X_6)(P_7 + X_7) \\
+ \quad & (P_1 + X_1)(1 - (P_2 + X_2))(P_5 + X_5)(P_6 + X_6)(P_7 + X_7) \\
+ \quad & (P_1 + X_1)(P_2 + X_2)(1 - (P_3 + X3))(P_5 + X_5)(P_6 + X_6)(P_7 + X_7) \\
+ \quad & (P_1 + X_1)(1 - (P_2 + X_2))(P_4 + X_4)(1 - (P_5 + X_5))(P_7 + X_7) \\
+ \quad & (P_1 + X_1)(1 - (P_2 + X_2))(P_4 + X_4)(P_5 + X_5)(1 - (P_6 + X_6))(P_7 + X_7) \\
+ \quad & (P_1 + X_1)(P_2 + X_2)(1 - (P_3 + X_3))(P_4 + X_4)(1 - (P_5 + X_5))(P_7 + X_7) \\
+ \quad & (P_1 + X_1)(P_2 + X_2)(1 - (P_3 + X_3))(P_4 + X_4)(P_5 + X_5)(1 - (P_6 + X_6)) \\
& (P_7 + X_7)
\end{aligned}
$$

There are three types of constraints in the model. They are:

1. Constraints to ensure arc survivabilities are ≤ 1.
2. Flow capacity constraints for each path j.
3. A budget constraint limiting the amount spent on both arc capacity and survivability improvements to a total budget amount, β.

The first type of constraints. one for each arc, were used to ensure arc reliabilities had an upper bound of 1. The constraints are

$$P_i + .1 \cdot X_i \leq 1$$

The second type. flow capacity constraints, were formulated by letting a_{ij} equal 1 if arc i lies on path j. and 0 otherwise. Using this definition, the constraints for flow capacity. one for each path. are

$$\textstyle\sum_{j=1}^{7} a_{ij} f_j \leq u_i + d_i$$

Where u_i is the capacity of arc i. and d_i is the amount of capacity improvement to arc i.

The third type; the budget constraint is

$$\sum_{i=1}^{n}(\alpha_{ci}d_i + \alpha_{ri}X_i) \leq \beta$$

Where α_{ci} is the cost of increasing the capacity of arc i by 1 unit, α_{ri} is the cost of increasing reliability of arc i by .1, and β is the total amount of the budget available to be invested.

The above two-criterion objective problem (MCO) can be converted to a single objective mathematical programming model by using one objective as an additional model constraint and parametrically varying its value from its lower to upper bound. For this MCO model, the reliability criterion objective was chosen for the additional constraint. The lower bound for the reliability function is when there are no arc survivability improvements or in other words, the original network reliability of .8282. The upper bound is when the network reliability is equal to one. The term Λ will represent the parametric value of the reliability function CF_2. The nonlinear MCO model for the example network is then

$$\text{Max } \sum_{j=1}^{q} R_j f_j \qquad \text{where } R_j = \prod_i (P_i + .1 \cdot X_i)$$

s.t.

$$CF_2 \geq \Lambda$$

$$P_i + .1 \cdot X_i \leq 1 \qquad i = 1, 2, \ldots, n$$

$$\sum_{j=1}^{q} a_{ij} f_j \leq u_i + d_i$$

$$\sum_{i=1}^{n}(\alpha_{ci}d_i + \alpha_{ri}X_i) \leq \beta$$

$$X_i, P_i, f_j, \alpha_{ci}, \alpha_{ri}, \beta, \Lambda \geq 0 \text{ and } a_{ij} = 0, 1$$

5.1.1 MCO Analysis. The MCO model was applied to the example network, Figure 3, using the *GAMS* solver, on a personal computer. The example network consists of seven arcs, six nodes and three paths from source (node 1) to sink (node 6). Arc survivabilities were initially .8, with an associated capacity as illustrated in Figure 3. The model was run with a capacity improvement cost α_{c_t} of 1 and a survivability improvement cost α_{r_t} of 1. The total budget available β was determined to be 6. This is the minimum amount required to improve all arcs in a source to sink path to one, thus providing for a network reliability of one. Given this minimum budget, if a tradeoff exists, then the budget would not allow both the maximization of expected flow and a network reliability of one to occur simultaneously.

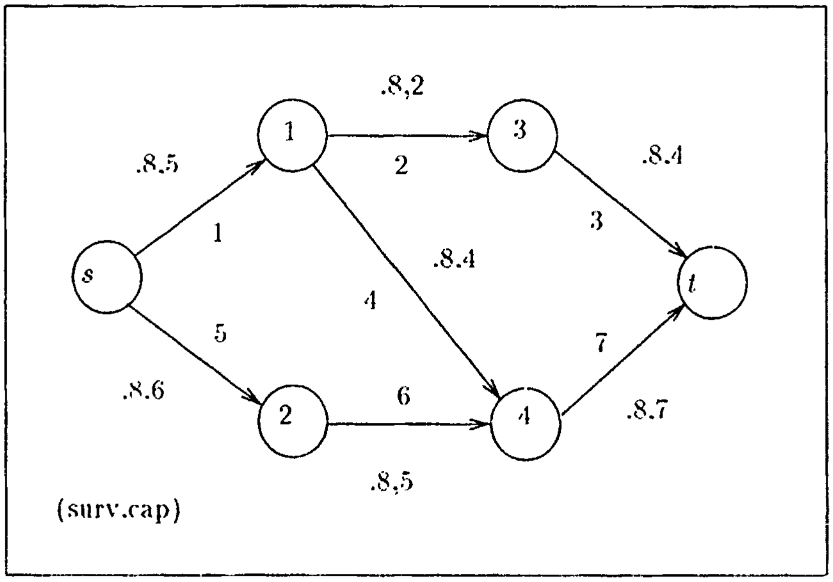


Figure 3. Example Network

The initial computer run of the model produced a reliability value of .975 and an expected flow of 7.16. This shows that the lower bound constraint of .8282 for network reliability was not a binding constraint. In other words, arc survivability improvements benefitted both reliability and expected flow without a loss to either. CF_2 was then parametrically bounded with reliability values of .985 and 1.0. Now the reliability constraint, CF_2, is binding with a resulting decrease in expected flow for an increase in network reliability. The resulting efficient frontier is illustrated in Figure 4. The model generated those points that were nondominated, creating the efficient frontier. For reliabilities less than .975, the resulting reliability and expected flow values were dominated by the point (.975, 7.16). The model can be forced to generate the dominated points by setting CF_2 equal to, rather than greater than, its parametric bounding level. This modified model was run for two dominated reliability levels, .87 and .92, with the resulting expected flow values plotted in Figure 4.

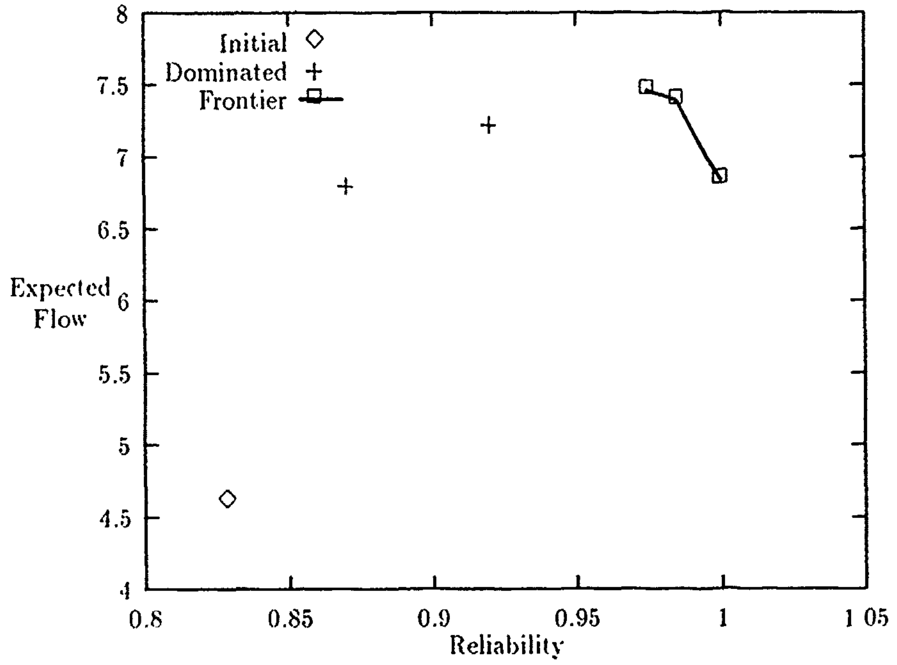


Figure 4. Example Network Efficient Frontier

5.1.2 Costing Analysis The MCO example above was run using both a capacity and survivability improvement cost of 1.0. The result was the generation of an efficient frontier bounded by a lower reliability value of .975 and an upper reliability value of 1.0. The effects of varying the cost improvement structure on the efficient frontier was examined. When the cost of improving a unit of capacity was increased to 2.0, the result was a shrinking of the efficient frontier to a single point at a reliability value of 1.0 and an expected flow of 7.30. In effect, the total budget was allocated to arc survivability improvements which maximixed both reliability and expected flow. On the other hand, when capacity improvement costs were decreased to 0.5, the efficient frontier was stretched between a lower reliability bound of .9156 with an expected flow of 8.196 and the original upper reliability bound of 1.0. Thus, decreasing the cost of capacity improvements relative to survivability improvements resulted in a stretching of the trade-off region whereas increasing the capacity improvement costs shrank the trade-off region to a point where there exists no trade-off. Further investigation into the effects of varying costing structure on the trade-off region is recommended in Chapter VIII.

5.1.3 MCO Summary. The MCO model formulated, successfully generated the efficient frontier for a small example network. The results illustrated that both reliability and expected flow can be improved with no loss to the other by increasing arc survivabilities up to a specific reliability level, after which further increases in reliability result in a tradeoff. By increasing reliability beyond that level, a decrease in expected flow results. It should be noted that the model is highly nonlinear in both the objective function ,CF_1, and the reliability constraint, CF_2. For large networks, if a reliability expression were obtainable, complications of optimizing highly nonlinear objectives with nonlinear constraints should be expected to be encountered.

Generating the efficient frontier generates all the possible nondominated solutions. One of these solutions will maximize a decision makers preference function. This optimal solution can be determined through two techniques. The first would be to compute a preference function based on surveys given to the decision maker and then maximize this preference function with respect to the efficient frontier solution set. The second technique would be to use an interactive approach where the efficient frontier points are generated interactively with the decision maker. In this technique, the decision maker's preference function is implicit in his interactive responses during the efficient frontier generation process with the process termination occurring at the solution point which maximizes the decision makers preferences.

5.2 Modifications

The above MCO model requires both model criteria be expressed in mathematical objective functions. The problem confronted is that for large stochastic networks, the reliability in this research is calculated using a recursive technique which does not result in a mathematical expression. Lacking this second criterion function, two methodologies, Relmax and Flomax, were developed to examine the tradeoff between expected flow and reliability.

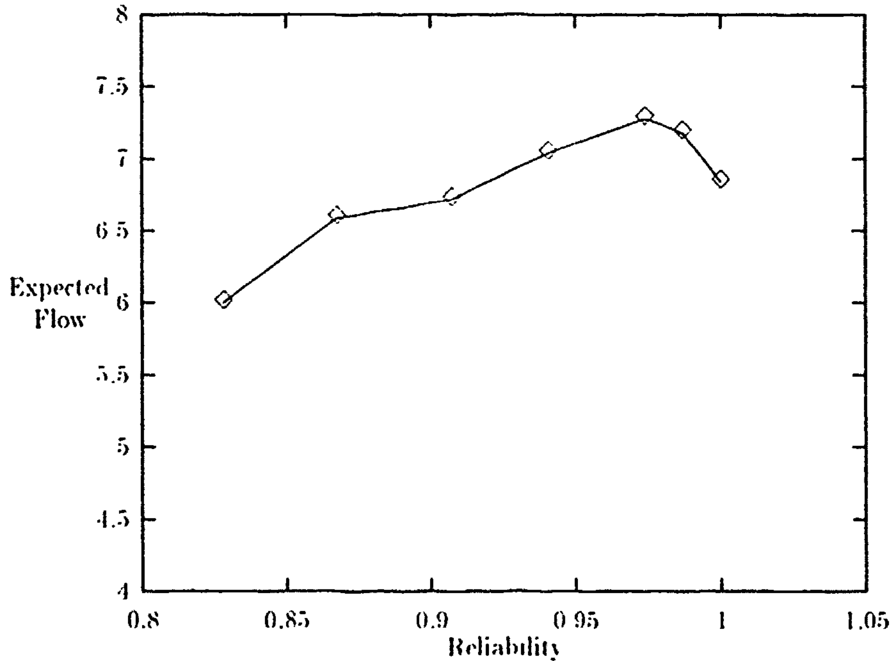
5.3 *Relmax*

The method Relmax used a two step solution process to examine reliability versus expected flow. The first step maximized the network reliability using the linear heuristic reliability improvement model described in Chapter IV, parametrically increasing the amount of arc survivability improvements allowed. This was accomplished by limiting the available arc survivability improvement budget to a specified portion of the total budget available. The remaining budget leftover and the resulting optimum arc survivabilities were then fed into Yim's expected flow improvement model. The network expected flow was then maximized constrained by the previously decided upon arc survivability improvements and the budget remaining after those improvements. For the example network, arcs one and seven have an important index of 74.07 while the remaining arcs have an index of 37.04. Arc four's index value was increased by one to ensure investment in arc four before the others to provide an improved path (1-4-7). The budget allocated to survivability improvement was increased by one unit each run with the remaining budget allocated to capacity improvement. The resulting reliability and expected flow are listed in Table 2 and Figure 5.

Table 2. Relmax Results

Survivability Budget	Capacity Budget	Network Reliability	Expected Flow
0	0	.8282	4.61
0	6	.8282	6.40
1	5	.8677	6.59
2	4	.9072	6.72
3	3	.9406	7.04
4	2	.9741	7.28
5	1	.9870	7.18
6	0	1.00	6.84

Figure 5. Relmax Plot



5.4 Flomax

The second method, Flomax, used Gaught's nonlinear model described in Chapter II to maximize the network expected flow using both arc capacity and survivability improvements. The model was modified with an additional arc capacity constraint. This constraint was incorporated into the model to ensure an arc's capacity was not improved by an unreasonable amount, as determined by the user. The arc capacity constraint is thus

$$u_i + di \le C_{max}$$

The resulting nonlinear model maximizing expected flow is

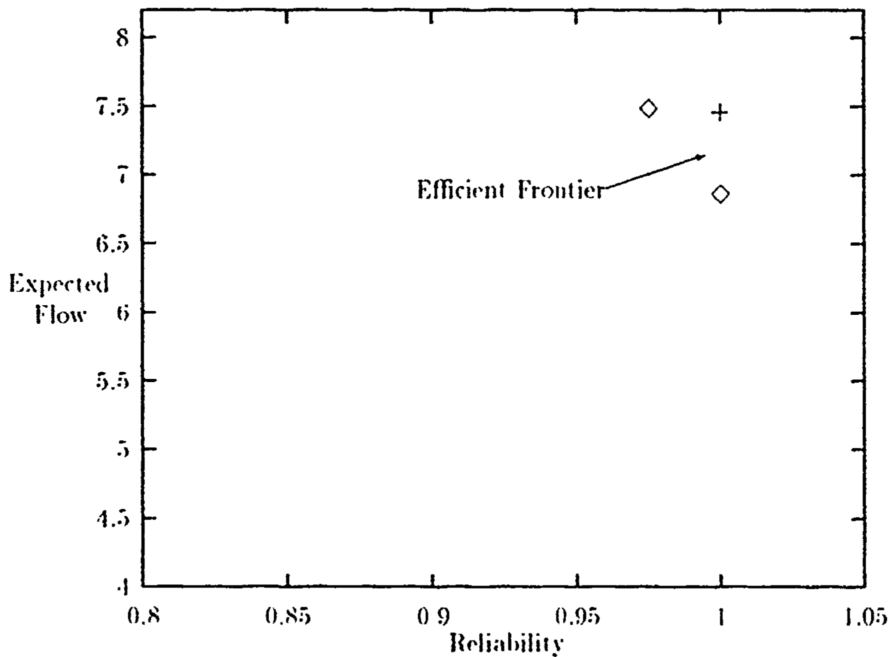
$$\text{Max } \sum_{j=1}^{q} R_j f_j \qquad \text{where } R_j = \prod_i (P_i + .1 \cdot X_i)$$

s.t.

$$
\begin{array}{ll}
P_i + .1 \cdot X_i \le 1 & i = 1,2,\ldots,n \\
u_i + d_i \le C_{max} & i = 1,2,\ldots,n \\
\sum_{j=1}^{q} a_{ij} f_j \le u_i + d_i & \\
\sum_{i=1}^{n} (\alpha_{ci} d_i + \alpha_{ri} X_i) \le \beta & \\
X_i, P_i, f_j, \alpha_{ci}, \alpha_{ri}, \beta, \lambda \ge 0 \text{ and } a_{ij} = 0,1 &
\end{array}
$$

By applying the above model to the example network, the maximum expected flow obtainable within the budget was obtained. The associated reliability is calculated using *Dirprog*. This point is where the tradeoff or efficient frontier begins between expected flow and reliability and thus bounds the one side of the efficient frontier. For the example network, this point was where reliability equaled .975 and expected flow equaled 7.16. Forcing the reliability beyond .975 caused a subsequent decrease in expected flow as shown with the MCO model. Lacking a reliability function, the reliability cannot be forced to increase with a subsequent maximization of expected flow. However, given the minimum budget to provide a perfect path or reliability equal to one, Yim's lower bound model can be run to calculate the expected flow for the condition when all the budget has been spent to provide a network reliability of one. For the example network, a budget of 6 will be used up in improving path (1-4-7) to a perfect reliability of one. The associated expected flow is 6.84. The efficient frontier is thus bounded for the example network as shown in Figure 6. In addition, an ideal was defined as the maximum obtainable values for the two critera. The trade-off region was thus bounded by the box formed by the lower and upper reliability bounds and the ideal. All other investment feasible investment options were dominated by this region.

Figure 6. Flomax Results



5.5 *Summary*

The ideal MCO approach requires mathematical expressions for both criterion functions, expected flow and reliability. Given that for large stochastic networks, a reliability expression is not computationally feasible to obtain, the tradeoff between the two objectives were analyzed using two different methodologies. The first method initially maximized the network reliability using the linear heuristic reliability improvement model, then used the resulting arc survivability improvements to maximize the expected flow through capacity improvements with the remaining budget. This was accomplished using Yim's linear expected flow improvement model. The second nonlinear method maximized the expected flow by both arc survivability and arc capacity improvements. The resulting network reliability was then calculated based on the arc survivability improvements made. This point was the lower reliability bound for the efficient frontier. The upper reliability bound for the efficient frontier was generateded by calculating the expected flow when the network reliability equaled one. Model two thus bounds the efficient frontier and thereby bounds the trade-off region for the network. Both techniques were applied to large, realistic networks and analyzed in Chapter VII.

VI. *Integration*

This chapter contains the description of tools and programs u[illegible] in integrating the expected flow and reliability improvement models with *Formula* and *Dirprog*. The following sections describe the tools, files, and programs used in the above integration.

6.1 *Tools*

The following software packages were required for this research: *GNA*, *Arity/Prolog* Version 5.0 Interpreter, *Formula* Version 3.0, *GAMS* Version 2.05/S, *Turbo Pascal* Version 5.0, and *Dirprog*. In addition, an IBM compatible 386-25 personal computer, a Sun 386i workstation, and a DEC 8550 mainframe were used. A description of each tool follows.

6.1.1 GNA. Graphical Network Analyzer, is an interactive, graphical computer program for the design, display, and analysis of network models (14:6). In addition to *GNA* providing a method to graphically input and represent a network, it also has the option to convert the network structure into an arc based input file for *Formula*. The convert option splits all stochastic nodes into two perfect nodes with a connecting arc representing the stochastic nodes survivability. Convert also adds an artificial source and sink to the network as required by *Formula*. *GNA* was used to graphically draw and convert networks analyzed for subsequent input into *Formula*. Due to the large memory requirements of *GNA*, it was run on the Sun 386i workstation.

6.1.2 Arity/Prolog Interpreter. The *Arity/Prolog* Version 5.0 Interpreter is an enhanced Prolog artificial intelligence language interpreter produced by the Arity Corporation of Concord, Massachusetts (4). Enhancements include an integrated editor, pull-down windows, and window creation capabilities. The interpreter runs under the DOS environment and was used on a personal computer to edit and run *Formula* Version 3.0.

6.1.3 Formula Version 3.0. Version 3.0 is a modified version of Gaught's *Formula* Version 2.0 (12:252-275) which was in turn based on Yim's original *Formula* program (23:99-117). *Formula* is written in the Prolog artificial intelligence programming language and requires the *Arity/Prolog* interpreter to run. Yim and Gaught used *Formula* to find all network source to sink paths and then to formulate their models creating appropriately formatted files for input into predetermined commercial mathematical solver packages (GINO or LP/MIP-83).

Formula has been modified creating version 3.0 for two primary purposes in relation to this work. The first is to create a file *path.f* which lists all network source to sink paths. The second purpose is to create three additional files — *prob.f, cap.f,* and *net.top* — which contain the arc survival probabilities, the arc capacities, and the network topology described in arc parent-child relationships. The above files will be created when option eight, Formulate Reliability Files, is selected when running *Formula* Version 3.0. A user's manual for Version 2.0 with appropriate additions for Version 3.0 enhancements is contained in appendix D while a complete source code listing of *Formula* Version 3.0 is contained in appendix E.

6.1.4 GAMS. General Algebraic Modeling System is a commercial software package developed by the World Bank, that allows mathematical programming models to be entered in concise algebraic statements, then solved using the linear (BDMLP), nonlinear (MINOS 5), or mixed integer (ZOOM) solvers included. Using a high-level language, *GAMS* allows for the compact representation of large and complex models, simple and safe changes to model specifications, unambiguous statements of algebraic relationships, and model descriptions that are independent of solution algorithms (7:3). Version 2.05/S is the personal computer student version running under the DOS environment.

6.1.5 Turbo Pascal. Version 5.0 of *Turbo Pascal* is a structured, high-level language used to write executable programs under the DOS environment. While there is a high degree of compatibility with Versions 3.0 and 4.0, there are differences which the reader can examine in (6:appendix A). It also should be noted that *Turbo Pascal* is not compatible with ANSI Pascal.

6.1.6 Dirprog. The reliability determination program, *Dirprog*, is a Turbo Pascal based program written by Lavon Page and Jo Perry to solve for the exact reliability of directed stochastic networks as described in (18). The input file name is input by the user from the screen and an output file is generated. A source code listing is in appendix F.

6.1.7 Modprog. This is a modified ANSI Pascal version of *Dirprog* compiled on the DEC 8550 mainframe under the VMS operating environment. *Modprog* reads from the input file *infile.dat* and the results are output to the file *outfile.dat.* The format of the input and poutput files are the same as with *Dirprog. Modprog* allows large networks that otherwise would exceed the 64k stack limit under MS DOS to be run. A source code listing is includedin appendix F.

6.1.8 Capinv. The Turbo Pascal program *Capinv* is used to generate the *GAMS* input file *Capinv.gms* for the capacity improvement model. It requests the cost of improving one unit of arc capacity and the total budget available. A source code listing is included in appendix G.

6.1.9 Relinv. This is another Turbo Pascal program used to generate the *GAMS* input file Relhuer.gms for the linear heuristic reliability improvement model. It request the user for the cost of improving an arc's survivability by .1, and the total budget available. A source code listing is included in appendix G.

6.2 Files

There are several different files used in this research. A brief description of each follows:

- *formin.ari* is a file generated by *GNA* which contains the network topology along with all arc survivabilities, capacities and costing information. It is formatted for input into *Formula.*
- *path.f* is a file generated by *Formula* Version 3.0 which contains all network source to sink paths in an arc based description.
- *prob.f* is a file generated by *Formula* Version 3.0 which contains the network arc survival probabilities.
- *cap.f* is a file generated by *Formula* Version 3.0 which contains the network arc capacities.

- *net.top* is a file generated by *Formula* Version 3.0 which contains the network topology as represented by arc parent-child relationships.
- *relhuer.gms* is a file generated by the program *Relinv* which contains the reliability improvement model for input to *GAMS*.
- *capinv.gms* is a file generated by *Capinv* which contains the capacity improvement model for input to *GAMS*.

The integration of the files and programs is illustrated in figure 7.

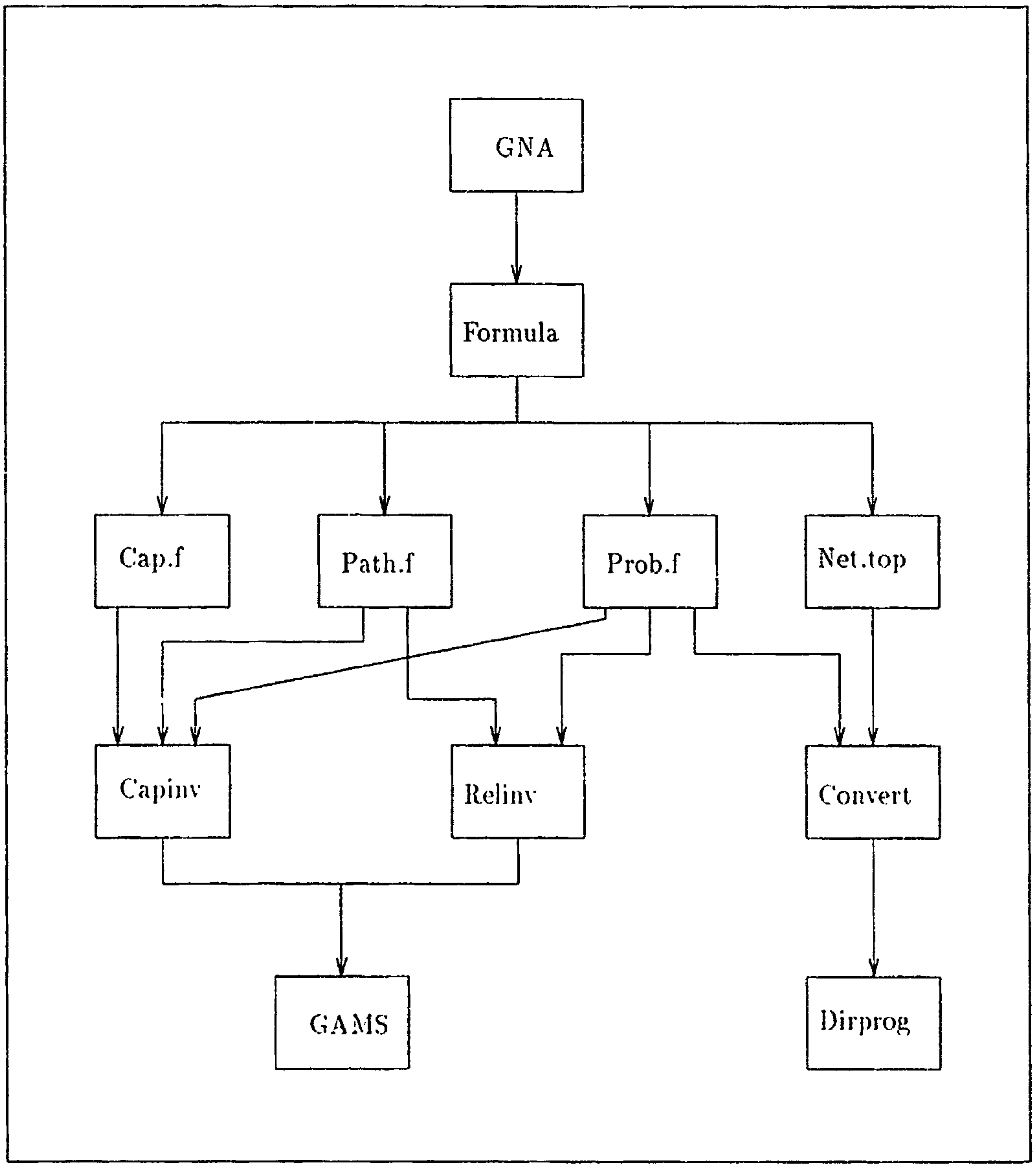


Figure 7. Flow Diagram

VII. Results and Analysis

Two networks, A and B, were analyzed with respect to the three objectives of this work. First, the original network performance parameters of expected flow and reliability were calculated. Second, a linear heuristic reliability model was applied to examine the priority ranking for arc selection for survivability improvement and the resulting network reliability achieved. Finally, two techniques to maximize expected flow and reliability given a specified budget were applied to examine tradeoffs between the two. In addition, Network C was included to demonstrate the limitations of the reliability measurement tool, *Dirprog*, under the MS DOS environment.

7.1 Network A

Network A is shown in Figure 8. It consists of 19 nodes and 27 arcs. Tables 3 and 4 show the capacity and survivability values of the arcs and nodes. In addition, Tables 5 and 6 show the dependent arc and node pairs. The dependent pairs represent a single medium where, if dependent, then both of the arcs/nodes in the pair will fail if one fails. In addition, Network A has 63 potential paths between the added artificial source and sink nodes. The procedure for adding an artificial source and sink is described in the *Formula* Users Manual contained in Appendix D. The converted Network A is illustrated in Appendix A.

7.1.1 Original Performance Parameters. The two performance parameters, lower bound for expected flow and network reliability, were calculated for Network A. Using the GAMS solver and Yim's expected flow model, an expected flow of 167 and a deterministic max flow of 9600 (all arc survivabilities equal to one) were obtained which agrees with previous values calculated by Yim and Gaught using the LP/MIP-83 solver. The reliability was calculated with Page and Perry's *Dirprog* program. A network reliability value of .1504 was obtained in .11 seconds using a 386 Dx personal computer. It required 44 reductions and 2 factorings. Network A was readily reducible thus allowing an efficient calculation if its reliability. The reliability value of .1504 agrees with that obtained by Bailey's simulation program *Maxflo* (5).

Table 3. Description of Arcs in Network A

Start Node	Terminate Node	Reliability	Capacity
1	12	1.0	1200
1	13	1.0	1200
1	14	1.0	1200
2	5	0.3	1200
2	14	0.6	1200
3	9	1.0	1200
3	11	1.0	1200
4	14	1.0	1200
5	10	0.6	1200
5	11	0.7	1200
6	14	0.6	4800
7	14	0.6	4800
8	14	0.3	4800
9	15	1.0	4800
10	15	0.6	4800
11	15	1.0	4800
12	15	0.7	4800
13	15	1.0	4800
14	17	0.3	4800
14	18	0.6	4800
14	19	0.6	4800
15	6	0.3	4800
15	7	0.6	4800
15	8	0.7	4800
17	16	0.7	4800
18	16	0.6	4800
19	16	0.3	4800

Table 4. Description of Nodes in Network A

Node	Reliability	Capacity
1	1.0	*
2	0.3	*
3	0.7	*
4	0.5	*
5	0.8	*
6	1.0	*
7	0.3	*
8	0.7	*
9	0.5	*
10	0.8	*
11	1.0	*
12	0.3	*
13	0.7	*
14	0.5	*
15	0.8	*
16	0.8	*
17	0.7	*
18	0.3	*
19	1.0	*

* implies capacity is infinite

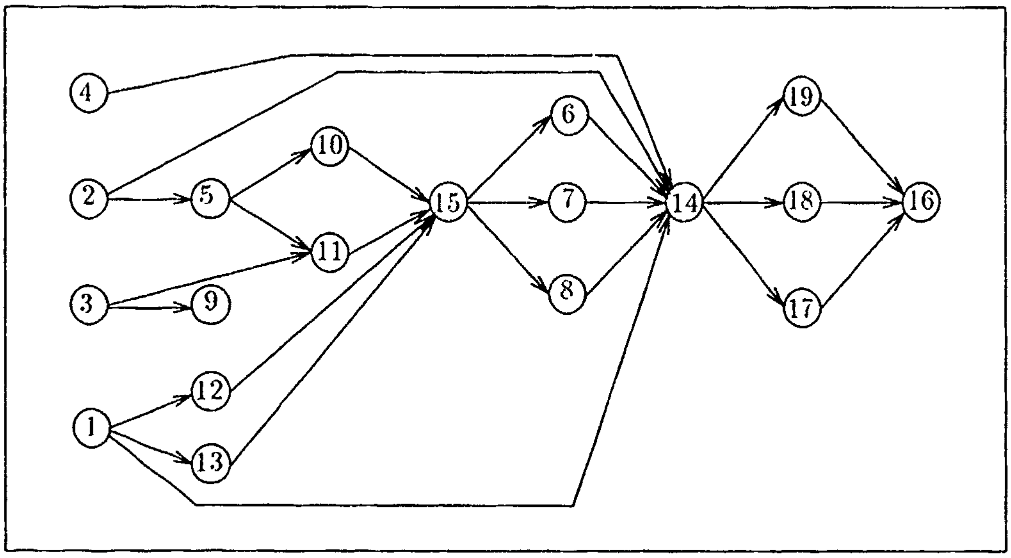


Figure 8. Network A

7.1.2 Reliability Improvement. Given the initial network reliability value, the linear heuristic reliability improvement model was applied. The importance index for each node and arc are shown in Figures 7 and 8. The heuristic model invested in arc survivability improvements for nodes 14 and 16 'nitially as their importance index of 20.25 were the greatest. The next candidate was node 15 with an index of 7.90, followed by node 2 with 7.57. Next were several arcs with an index of 6.75. By examining the location of the arcs, only one path between node 15 and 16 and again between 11 and 16 needed improved. Since the model does not have a tie breaker, it would improve them in numeric order. It is here that a user may interject a tie breaker by adding an amount to the preferred arc or arcs. In this case, arcs (14-19) and (19-16) were increased to 6.85 This ensured after their investments, a perfect path would exist with a resulting network reliability of 1.0. The arc improvements with the associated network reliabilities are listed in Table 9. It is noted that improvements in arcs or nodes located before node 14 had no affect on the network reliability as there existed a perfect path from the source to node 14 through node one. This is shown where investments in arcs 2 and 15 did not increase the network reliability.

Table 5. Dependent Arcs in Network A

Dependent Pairs			
Start Node	Terminate Node	Start Node	Terminate Node
6	14	14	19
7	14	14	18
8	14	14	17
15	6	19	16
15	7	18	16
15	8	17	16

7.1.3 Combined Capacity and Survivability Improvements. Finally both network performance parameters, expected flow and network reliability were improved through investments in arc capacity and survivabilities. Method one, Relmax, as described in Chapter V, was applied to maximize network reliability first, then maximize expected flow with the remaining budget.

Before applying the models, an improvement cost for both capacity and survivability needed to be decided upon. The costs were selected so as to equalize the effect on improving the effective arc capacities given an equal investment, thus eliminating any cost advantage of one over the other. First, a capacity improvement cost of 1 was selected. This resulted in a cost of 1200 to double the arc's effective capacity from 360 to 720 given a survivability of .3. To achieve the same increase in effective capacity through survivability improvements, an investment of .3 units of survivability was required Thus, an arc survivability cost was selected to be 400 which resulted in a total cost of 1200, the same as the capacity investment. In addition, arc

Table 6. Dependent Nodes in Network A

Dependent Pairs	
Node	Node
6	19
7	18
8	17
15	16

Table 7. Network A - Node Importance Index

Node	Index
1	6.34
2	7.57
3	2.22
4	4.11
5	3.46
6	2.63
7	2.63
8	2.63
9	1.11
10	1.73
11	2.84
12	1.11
13	1.11
14	20.25
15	7.90
16	20.25
17	6.75
18	6.75
19	6.75

capacity improvements were limited so that the final arc capacities were less than or equal to 4800 (the maximum arc capacity in the original network). Budgets of 5, 10, and 15 thousand were then used for the subsequent model runs. Each run varied the amount dedicated to each improvement (survivability or capacity) by a tenth where the left side of the curve represented all the budget going for capacity improvements and none for survivability, and the right side represented all the budget going to survivability improvements and none to capacity.

Relmax results are plotted in Figure 9. Examining this plot, it can be seen that a budget totally devoted to capacity increases does not increase network reliability and the subsequent expected flow is less than that which can be achieved with a combination of capacity and survivability improvements.

The second nonlinear model was applied with a budget of 8,000 which was determined to be the minimum amount required to have a network reliability of one. A program was written to examine all paths from source to sink and determine which path was the least costly to improve to reach a network reliability of one. For network A, this path consisted of nodes (1-14-19-16) with a cost of 8000. The

Table 8. Network A - Arc Importance Index

Start Node	Terminate Node	Index
1	12	1.11
1	13	1.11
1	14	4.11
2	14	4.11
2	5	3.46
3	11	1.11
3	9	1.11
4	14	4.11
5	10	1.73
5	11	1.73
6	14	2.63
7	14	2.63
8	14	2.63
9	15	1.11
10	15	1.73
11	15	2.84
12	15	1.11
13	15	1.11
14	19	6.75
14	18	6.75
14	17	6.75
15	6	2.63
15	7	2.63
15	8	2.63
17	16	6 75
17	16	6.75
18	16	6.75
19	16	6.75

Table 9. Network A Reliability Improvements

Node	Arc	Increase	Reliability
14		.5	.300
16		2	.376
15		2	.376
2		2	.376
	19-16	7	.6956
	14-19	1	1.00

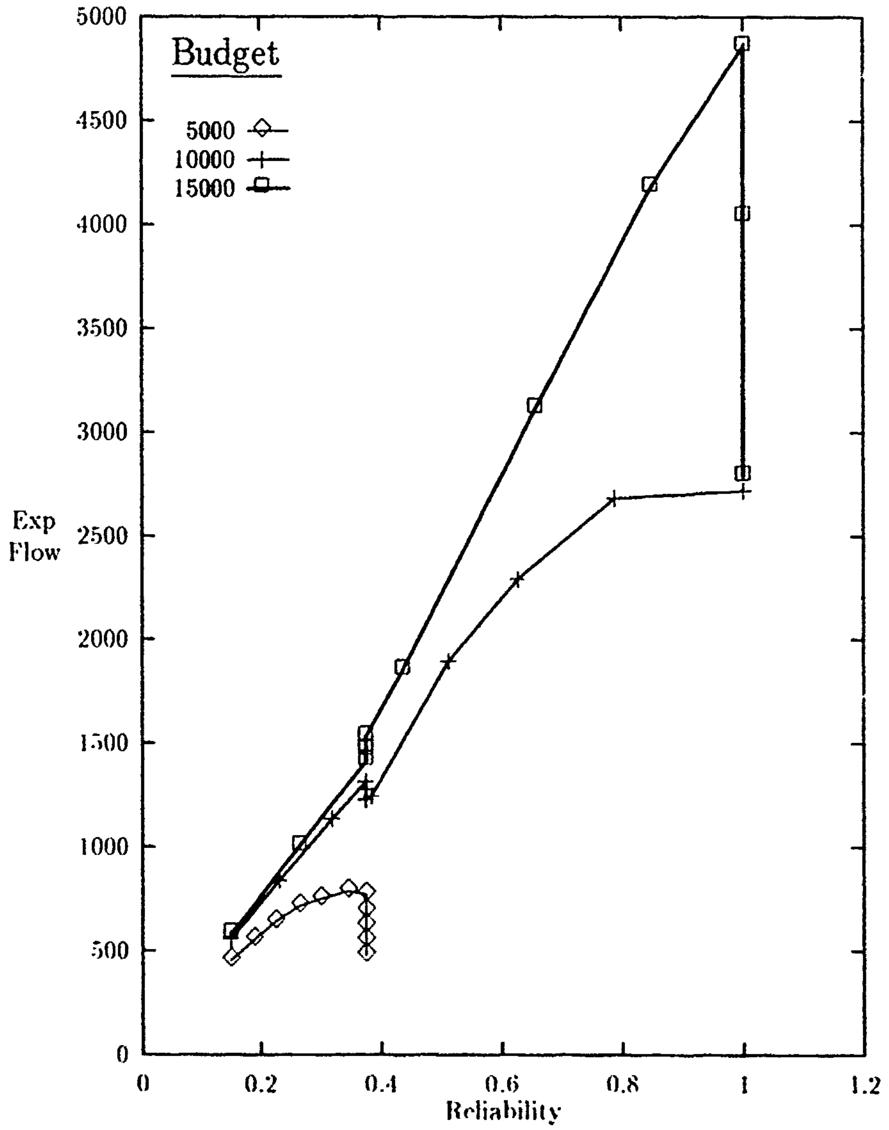


Figure 9. Network A Combined Improvements

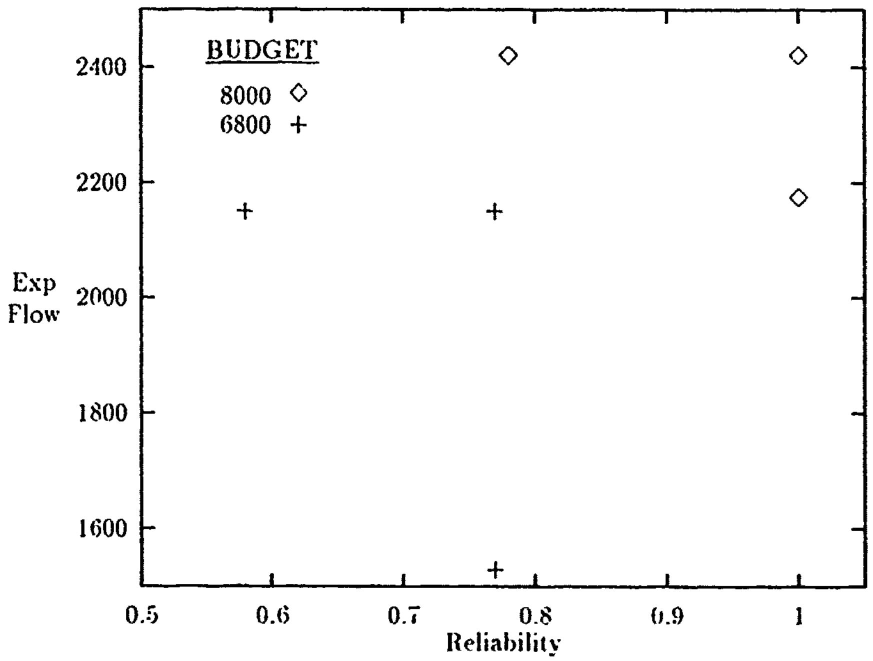


Figure 10. Network A Efficient Frontier

maximum expected flow was then calculated using the improved path above, with a result of 2170. Next, the nonlinear model was run to maximize the expected flow through both arc survivability and capacity improvements, limited by a budget of 8000. The resulting flow was 2415 with a reliability of .78. Thus, the efficient frontier is bounded as shown in Figure 10.

For budgets less than that required to reach a network reliability of one, the tradeoff point can still be calculated using the nonlinear model. The upper reliability bound for the efficient frontier however is in itself, now a lower bound as there are several different arc survivability configurations that could be used to maximize the reliability and expected flow. A lower bound for this value can be calculated by using the least cost path determined above and decreasing the arc survivabilities to a point that the budget will provide the remaining survivability increases. This reliability value is then used to maximize the associated expected flow. For Network A, a budget of 6800 produces a tradeoff point at (.58,3743) with a maximum reliability obtained at (.77,2151) as shown in Figure 10.

7.2 Network B

Network B is shown in figure 11. It consists of 26 nodes and 37 arcs. Tables 10 and 11 list the capacity and survivability values of the arc and nodes. The network contains 187 paths from the artificially added source to sink.

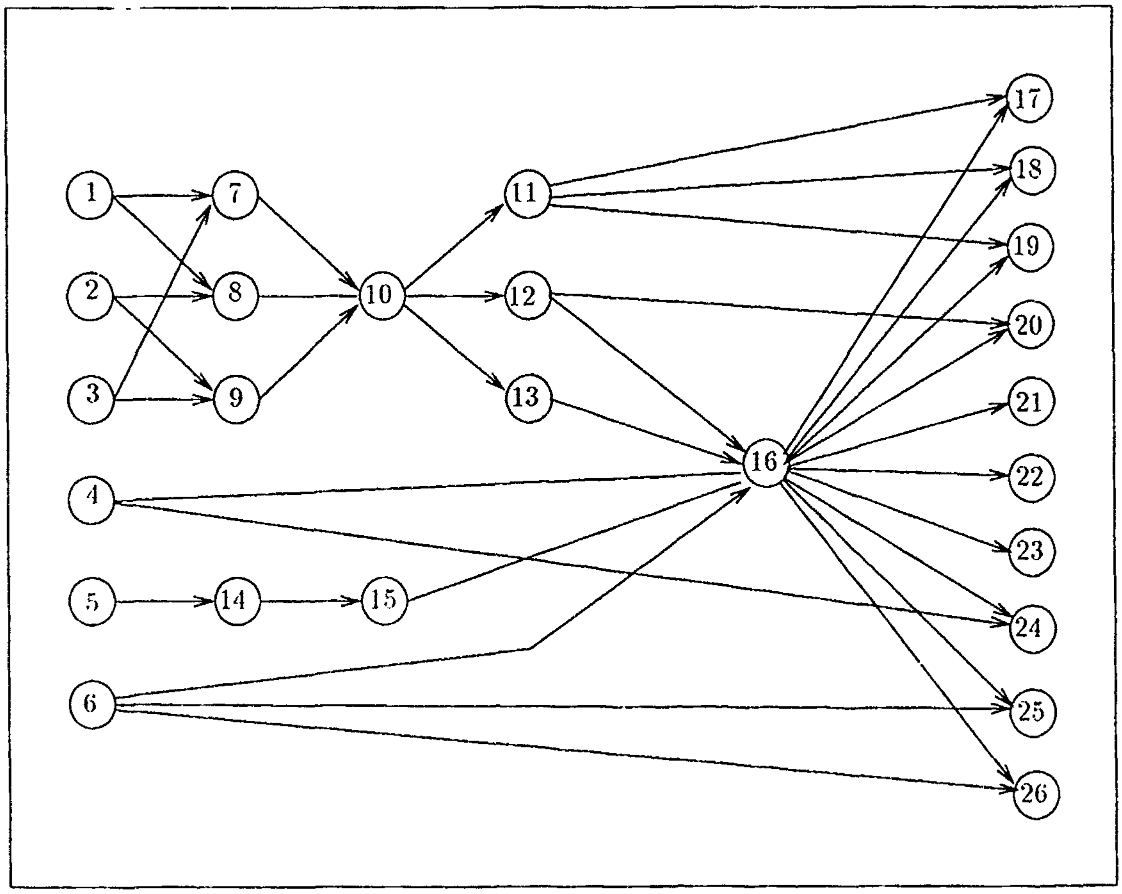


Figure 11. Network B

7.2.1 Original Performance Parameters. As with network A, the two performance parameters, lower bound for expected flow and network reliability, were calculated. An expected flow of 344 and a deterministic flow of 3900 were obtained which agreed with previous values calculated by Yim (23:82). The reliability was computed using *Dirprog*, producing a result of .6011. The computation took 58 sec-

Table 10. Description of Arcs in Network B

Start Node	Terminate Node	Reliability	Capacity
1	7	0.8	150
1	8	0.8	200
2	8	0.5	750
2	9	0.5	750
3	7	0.8	200
3	9	0.5	750
3	16	0.6	150
4	16	0.8	200
4	24	0.5	600
5	14	0.8	1200
6	16	0.5	1200
6	25	0.6	75
6	26	0.6	75
7	10	0.5	1200
8	10	0.7	1200
9	10	0.5	2400
10	11	0.5	1200
10	12	0.7	1200
10	13	0.7	1200
11	17	0.5	1200
11	18	0.5	75
11	19	0.5	1200
12	16	0.7	1200
12	20	0.5	1200
13	16	0.7	600
14	15	0.8	1200
15	16	0.8	1200
16	17	0.6	75
16	18	0.6	75
16	19	0.6	75
16	20	0.6	75
16	21	0.6	75
16	22	0.6	75
16	23	0.6	75
16	24	0.6	75
16	25	0.6	75
16	26	0.6	75

Table 11. Description of Nodes in Network B

Node	Reliability	Capacity
1	0.70	*
2	0.15	*
3	0.03	*
4	1.00	*
5	1.00	*
6	0.04	*
7	0.40	*
8	1.00	*
9	0.01	*
10	0.70	*
11	0.11	*
12	1.00	*
13	0.06	*
14	0.09	*
15	0.18	*
16	0.07	*
17	1.00	*
18	1.00	*
19	1.00	*
20	1.00	*
21	1.00	*
22	1.00	*
23	1.00	*
24	1.00	*
25	1.00	*
26	1.00	*

* implies capacity is infinite

onds on a 386 Dx personal computer, with 7,422 reductions and 3,128 factorings required.

7.2.2 Reliability Improvements. The linear heuristic model for reliability improvement was applied to Network B with the node and arc important indices listed in Tables 13 and 14. It should be noted that nodes 4,5,8, and 12 were not included as their survivability was already one, thus were not candidates for survivability improvement. Arc (4-24) had the highest important index of 111.11. An improvement of this arc from .5 to 1.0 provides for a perfectly reliable path from source to sink and thus a network reliability of 1.0. The linear heuristic model successfully selected the arc that improves the network reliability for the least cost. If network reliability improvement was desired through another path, then node 16 should be improved as its index is 95.41. From figure 14, it is seen that a majority of the paths go through node 16. A listing of reliability improvements of arc (4-24) by tenths with the resulting network reliability is listed in Table 12.

Table 12. Arc (4-24) Survivability Improvements

Arc Improvement	Arc Survivability	Network Reliability
0	.5	.6011
.1	.6	.6809
.2	.7	.7607
.3	.8	.8404
.4	.9	.9202
.5	1.0	1.0

7.2.3 Capacity and Survivability Improvements. As in Network A, the methods, Relmax and Flomax, were applied to Network B. Relmax maximized reliability with a portion of the total budget and the remaining budget was used to maximize expected flow through capacity improvements. The results are listed in Table 15 and shown in Figure 12.

Flomax was applied to Network B for a budget of 2000 which allows a network reliability of one to be achieved and a budget of 1200 which allows for a maximum

Table 13. Network B - Node Importance Index

Node	Index
1	18.84
2	18.84
3	23.23
6	76.39
7	9.34
9	9.34
10	47.01
11	7.45
13	9.70
14	6.17
15	6.17
16	95.41

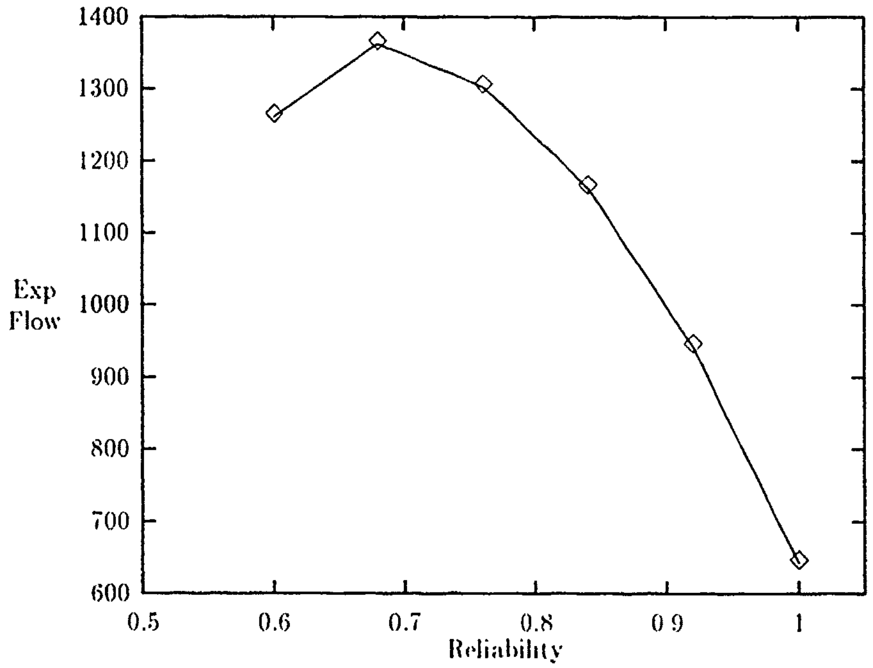


Figure 12. Combined Improvements

Table 14. Network B - Arc Importance Index

Start Node	Terminate Node	Index
1	7	4.67
1	8	14.17
2	8	14.17
2	9	4.67
3	7	4.67
3	9	4.67
3	16	13.89
4	16	40.00
4	24	111.11
5	14	6.17
6	16	13.89
6	25	31.25
6	26	31.25
7	10	9.34
8	10	28.33
9	10	9.34
10	11	7.45
10	12	29.85
10	13	9.70
11	17	2.48
11	18	2.48
11	19	2.48
12	16	11.76
12	20	18.09
13	16	9.70
14	15	6.17
15	16	6.17
16	17	9.54
16	18	9.54
16	19	9.54
16	20	9.54
16	21	9.54
16	22	9.54
16	23	9.54
16	24	9.54
16	25	9.54
16	26	9.54

Table 15. Network B Combined Improvements

Survivability Budget	Capacity Budget	Network Reliability	Expected Flow
0	2000	6011	1261
400	1600	.6809	1363
800	1200	.7607	1303
1200	800	.8404	1163
1600	400	.9202	943
2000	0	1.0	643

network reliability of 84. Again the efficient frontier was bounded for each budget between the maximum expected flow achievable and the maximum reliability achieved. The results are shown in Figure 13.

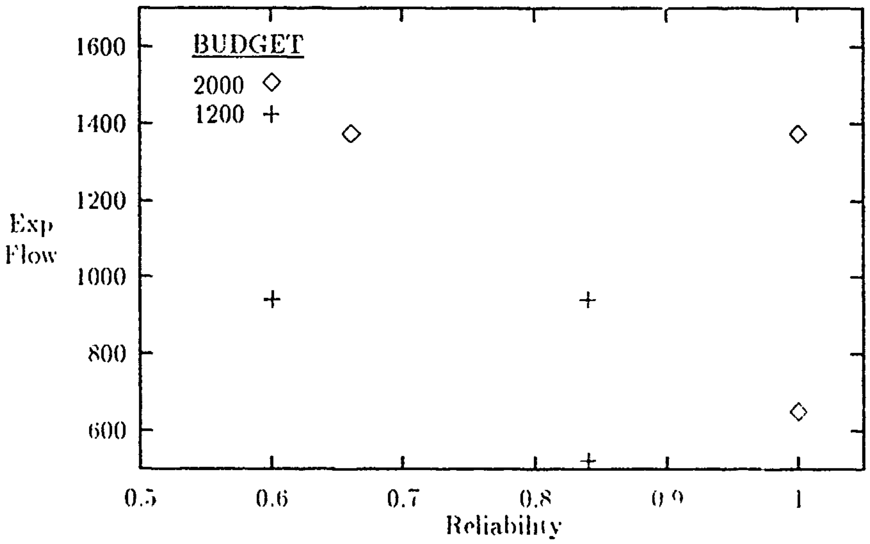


Figure 13. Network B Efficient Frontier

7.3 Network C

Network C was included to illustrate the limitations of the reliability program *Dirprog*. Network C consists of 39 nodes and 51 arcs with 198 paths. *Dirprog* is

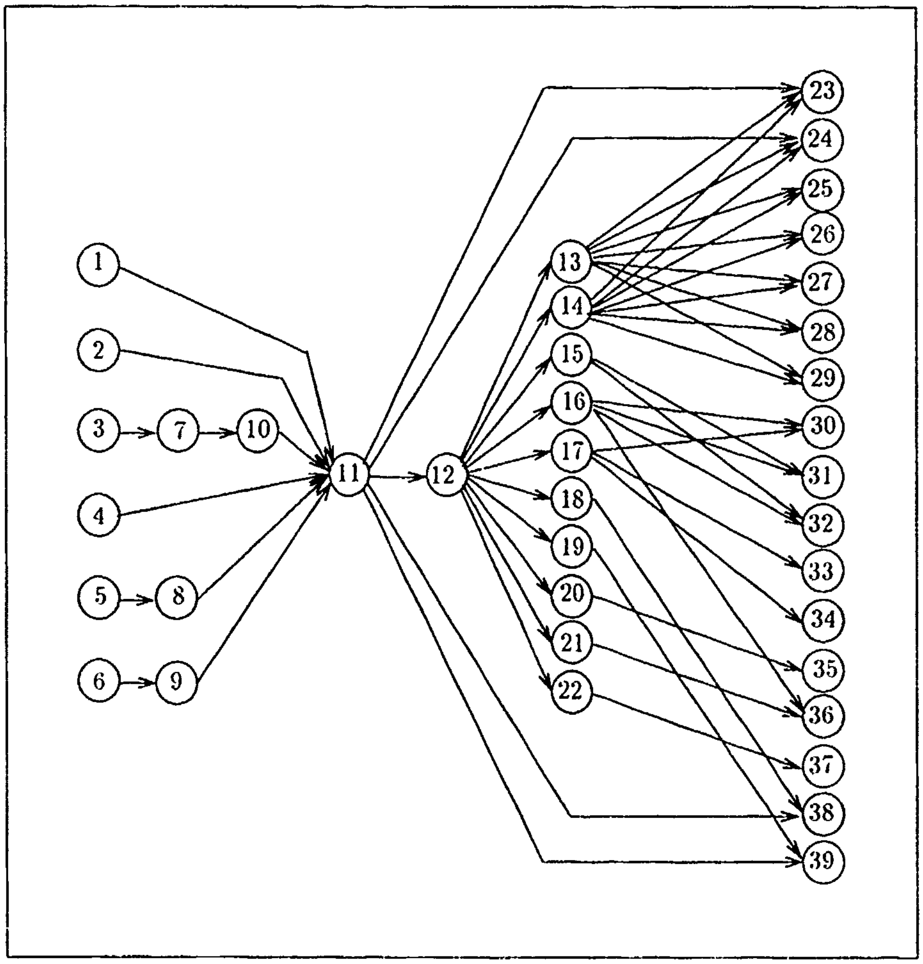


Figure 14. Network C

limited to the size of the network evaluated by a 64K stack limit under MS-DOS. Network C would not run under the Turbo Pascal Version of *Dirprog*. Modifications were made to convert the Turbo Pascal Version to an ANSI Pascal version and was run on a DEC 8550 mainframe operating under the VMS environment. The program was validated by running Networks A and B with the same results. A reliability value of .6157 was obtained in around 4 minutes. The calculation required 102.092 reductions and 32.239 factorings.

Table 16. Description of Arcs in Network C

Arc				Arc			
Init Node	Term Node	Survival Prob	Capacity	Init Node	Term Node	Survival Prob	Capacity
1	11	0.9	1200	13	25	0.7	2400
2	11	1.0	1200	13	26	0.9	1200
3	7	1.0	300	13	27	1.0	1200
4	11	0.6	1200	13	28	1.0	1200
5	8	0.3	1200	13	29	0.3	1200
6	9	0.6	1200	14	23	0.6	4800
7	10	1.0	300	14	24	0.3	4800
8	11	0.9	1200	14	25	0.6	2400
9	11	1.0	1200	14	26	0.7	1200
10	11	0.7	300	14	27	0.9	1200
11	12	0.9	9600	14	28	1.0	1200
11	23	0.6	75	14	29	1.0	1200
11	24	0.3	75	15	31	0.6	2400
11	38	0.6	1200	15	32	0.3	1200
11	39	0.7	1200	16	30	0.6	300
12	13	1.0	4800	16	31	0.7	2400
12	14	1.0	4800	16	32	0.9	1200
12	15	0.6	4800	16	33	1.0	300
12	16	0.3	4800	16	36	1.0	2400
12	17	0.6	4800	17	30	0.6	1200
12	18	0.7	2400	17	33	0.3	300
12	19	0.9	1800	17	34	0.6	300
12	20	1.0	4800	18	38	0.7	1200
12	21	1.0	4800	19	39	0.9	1200
12	22	0.6	2400	20	35	1.0	2400
13	23	0.3	1800	21	36	1.0	2400
13	24	0.6	4800	22	37	0.6	600

Table 17. Description of Nodes in Network C

Node	Survival Probability	Capacity	Node	Survival Probability	Capacity
1	0.3	*	21	0.3	*
2	0.7	*	22	0.7	*
3	0.5	*	23	0.5	*
4	0.8	*	24	0.8	*
5	1.0	*	25	1.0	*
6	0.3	*	26	0.3	*
7	0.7	*	27	0.7	*
8	0.5	*	28	0.5	*
9	0.8	*	29	0.8	*
10	1.0	*	30	1.0	*
11	0.7	*	31	0.3	*
12	0.7	*	32	0.7	*
13	0.5	*	33	0.5	*
14	0.8	*	34	0.8	*
15	1.0	*	35	1.0	*
16	0.3	*	36	0.3	*
17	0.7	*	37	0.7	*
18	0.5	*	38	0.5	*
19	0.5	*	39	0.5	*
20	1.0	*			*

* implies capacity is infinite

7.4 Summary

The exact reliability values for the three networks, A, B, and C, were successfully calculated with a computational time required on the order of seconds to minutes. On the other hand, previous work done by Bailey using a simulation approach was only successful in computing a relaibility value for Network A, and even then, required on the order of hours to compute. The factoring program has thus significantly reduced the computational times required to calculate the reliability of large stochastic networks and even more, provides a reliability measurement tool where previous simulation methodologies failed.

VIII. Conclusions and Recommendations

This research had three objectives to accomplish:

1. Develop a methodology to measure the network performance parameter, reliability, for large stochastic networks.
2. Develop a reliability improvement strategy.
3. Examine the trade-offs between the two criterion objectives, network expected flow and network reliability.

The following sections will discuss the conclusions arrived upon for each objective.

8.1 Reliability Measurement

Several techniques were examined in Chapter V for measuring the reliability of stochastic networks. It was determined that a recursive technique using reductions and factorings was the most appropriate. A program written by Page and Perry (18) was used to measure the exact value for three large networks. Networks A and B. demonstrated the efficiency of the program *Dirprog* to calculate the exact reliability value in a length of time short enough (seconds to a few minutes) to facilitate interactive uses. *Dirprog* used in conjunction with *GNA*, *Formula*, and *Convert* provided a system to graphically input a network, convert it into the appropriate input format, and measure the exact reliability value. This system readily lends itself to what-if type scenarios. Network C required solving on the DEC 8550 mainframe, thus demonstrating the network size limitations of *Dirprog* under the MS DOS operating system.

8.2 Reliability Improvement

Since the above reliability measurement procedure did not generate a mathematical expression for network reliability, a heuristic was used to determine the ranking of arcs for survivability improvements. The foundations of the heuristic are easily understood, thus lending credibility when presented to a decision maker.

The heuristic was incorporated into a linear model using the *GAMS* solver. The model was then applied to Networks A and B with promising results. It selected the optimum improvement path for Network B, though for Network A selected some arcs in where no improvement in network reliability was made. The heuristic seems most appropriate for reliability improvements where only a small number of arcs are improved, increasing the overall network reliability, but not providing a perfect source to sink path and a network reliability of 1.0. The method has the advantages of being computationally bounded by the number of arcs in the network, and not the number of states, thus can be applied to very large networks. The importance indices generated can also be examined and manipulated to incorporate a decision makers requirements or preferences in arc selection for survivability improvements.

8.3 Trade-off Analysis

While the preferred approach to trade-off analysis was a multicriteria optimization technique, lacking a criterion objective function for large networks limited the researcher to other methodologies. The first method of maximizing reliability with a portion of the budget, then applying the remainder to capacity improvements is useful in evaluating the proportion of the available budget to each given the desired expected flow and reliability. The second nonlinear method bounds the efficient frontier between the point representing the maximum expected flow obtainable and the maximum reliability point obtainable within the budget. This technique is useful in determining the tradeoff region. In addition, the two bounding points locate the maximum values of the two criteria attainable, and thus form the ideal point. If the desired performance parameters lie below this region then further tradeoff analysis is not required and the optimum arc investments can be obtained by using the nonlinear model to maximize the expected flow using both survivability and capacity. If the desired performance lies within the bounds of the efficient frontier, further analysis may be required to identify the efficient solutions bounded within the triangle formed by the two bounding points and the ideal. The feasible solution space below the bounded region though, is dominated by the efficient frontier and thus can be eliminated from further consideration, thus greatly decreasing the region of further analysis.

8.4 Recommendations

The following recommendations are made in furthering the research of improving the performance of stochastic networks:

1. Writing the program *Dirprog* in an artificial intelligence language such as PROLOG, which has an inherent, built-in recursive structure should be investigated. This may result in a reduction of the computational times required to calculate the reliability of large networks .
2. The reliability improvement heuristic should be applied to even larger networks and its performance analyzed. In addition, tie breaker rules should be implemented into the improvement model.
3. Further research in generating a usable mathematical expression for network reliability is needed to enable the use of multicriteria optimization in generating the efficient frontier. This then would allow the incorporation of the decision makers preference function in selecting an optimum investment strategy.
4. Developing a single mathematical model integrating both bounding end points in place of the seperate procedures used by Flomax will aide in resolving the gap between itself and MCO.
5. Further research is needed into the improvement cost functions to include non-linear cost functions and the affects of cost differences between the two types of improvements, capacity or survivability, on the trade-off or efficient frontier region.
6. While this research assumes independence between the arc survivabilities, investigations into models that take into account dependent arc survivabilities may better reflect real world conditions.

8.5 Summary

Measuring and improving the performance, both throughput and reliability, of large stochastic networks is desirable to ensure adequate capabilities are incorporated into communication networks during times of crisis. The everyday reliance on complex communication networks by DOD demands that the performance of these networks meet operational needs during adverse conditions. Using the methodologies formulated in this research, the performance parameters of both expected flow and reliability can be measured and subsequent improvements made providing insight into the operational capabalities of stochastic communication networks.

Appendix A. *Converted Networks*

Figure 15. Network A - Converted

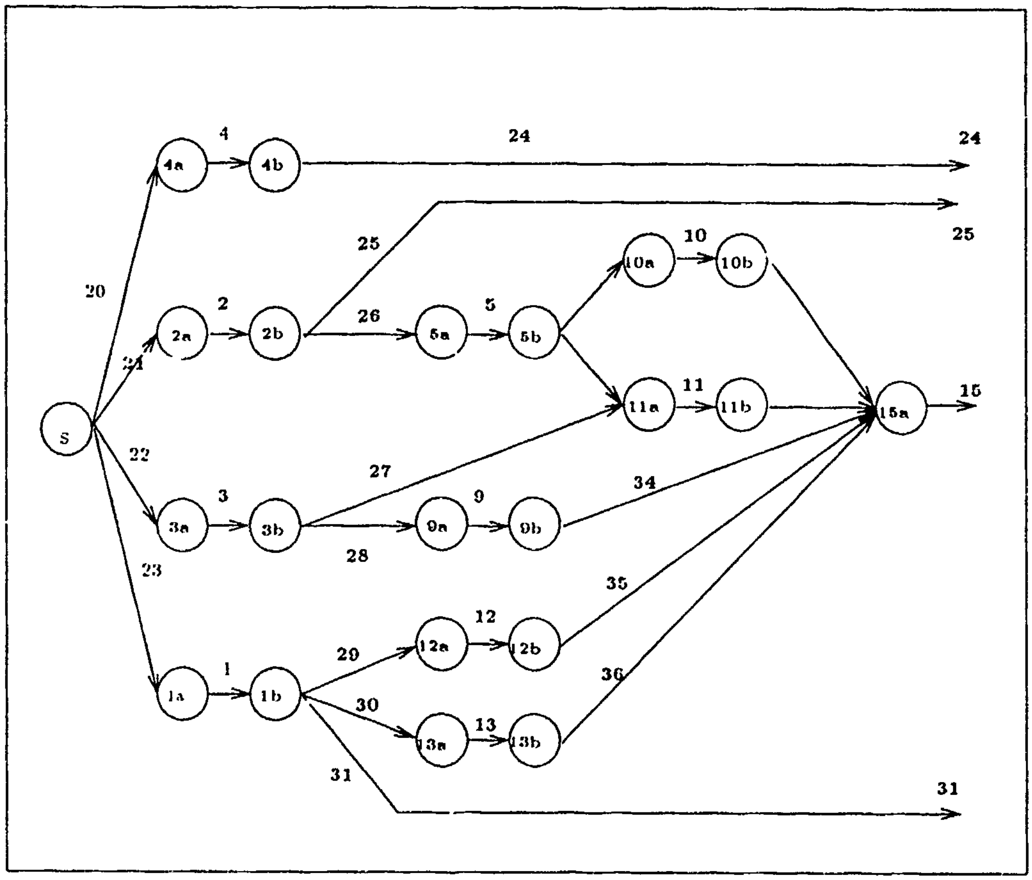


Figure 16. Network A - Converted

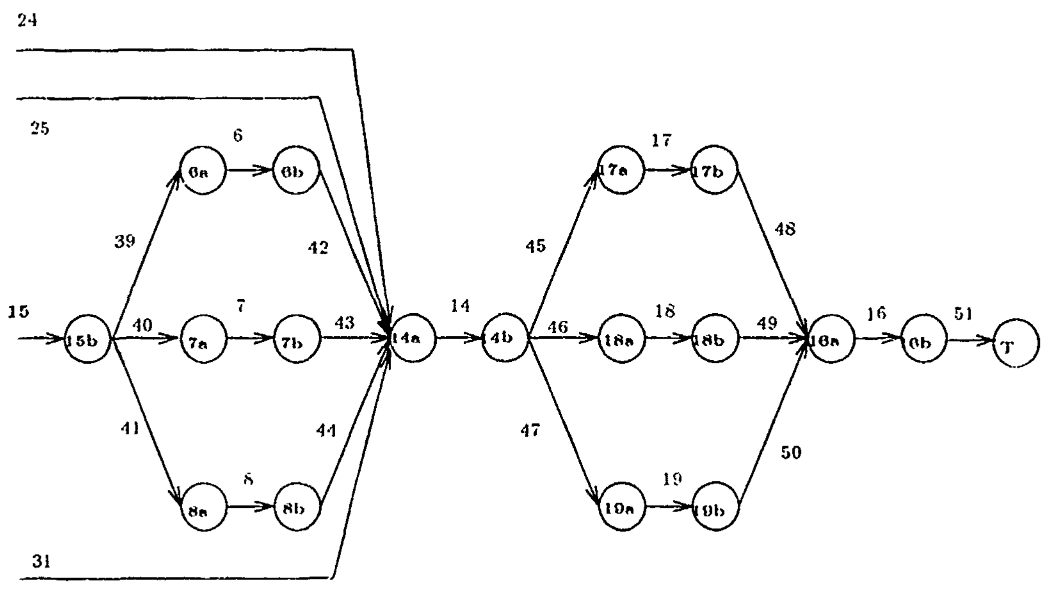


Figure 17. Network B - Converted

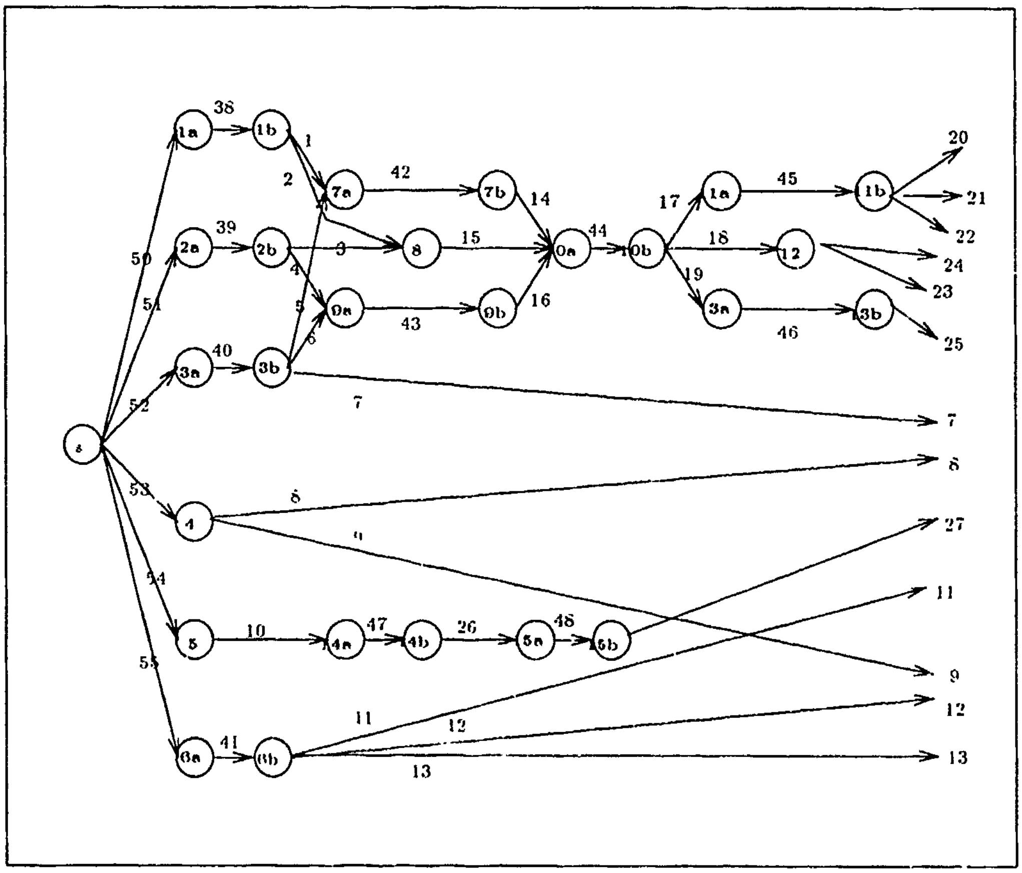
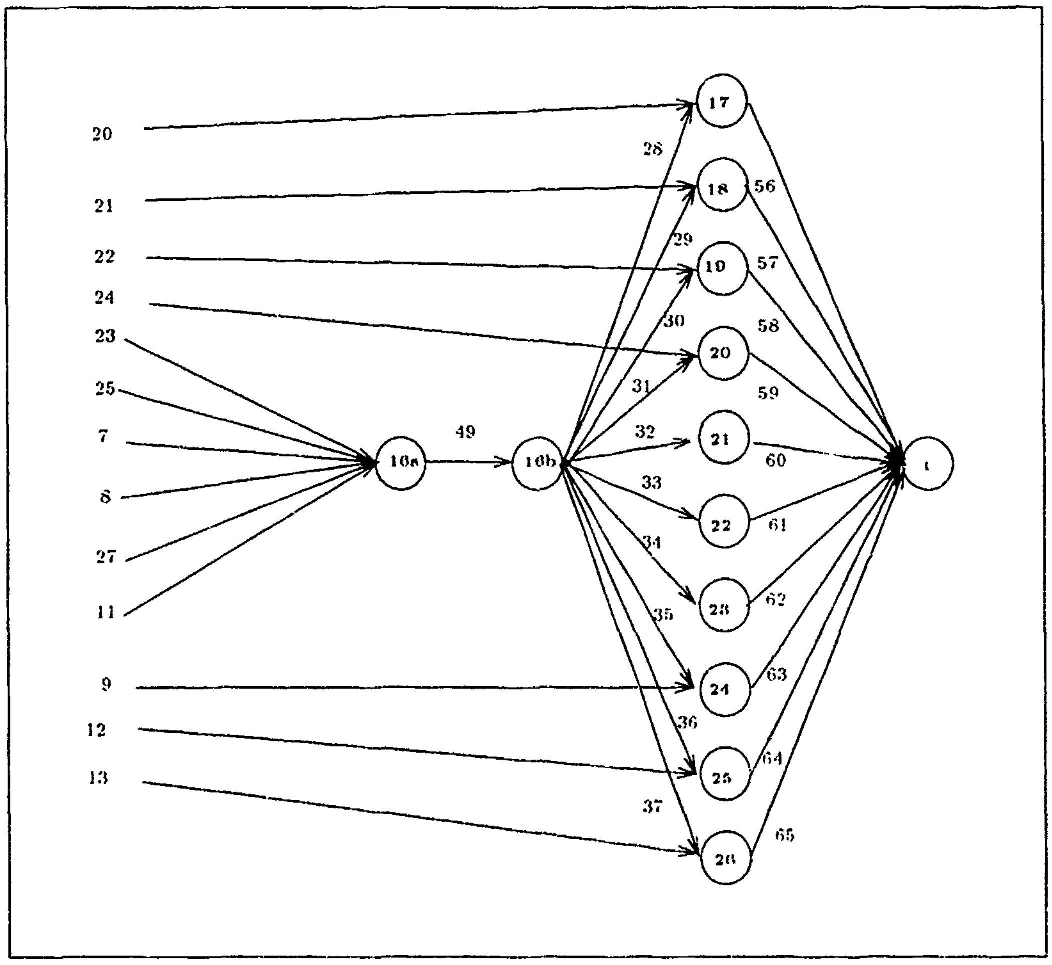


Figure 18. Network B - Converted



Appendix B. *Network A GAMS Files*

B.1 Linear Heuristic Reliability Improvement Model

```
$OFFSYMXREF OFFSYMLIST

SETS
   I   arcs  /1 * 51/;

PARAMETERS

    R(I)  arc reliability index
   /
 1  6.3374
 2  7.5754
 3  2.2222
 4  4.1152
 5  3.4602
 6  2.6349
 7  2.6349
 8  2.6349
 9  1.1111
10  1.7301
11  2.8412
12  1.1111
13  1.1111
14 20.2503
15  7.9047
16 20.2503
17  6.7501
18  6.7501
19  6.7501
20  4.1152
21  7.5754
22  2.2222
23  6.3374
24  4.1152
25  4.1152
26  3.4602
27  1.1111
28  1.1111
29  1.1111
30  1.1111
31  4.1152
32  1.7301
```

```
          .
          :
          .
          :
          :


33  1.7301
34  1.1111
35  1.1111
36  1.1111
37  1.7301
38  2.8412
39  2.6349
40  2.6349
41  2.6349
42  2.6349
43  2.6349
44  2.6349
45  6.7501
46  6.7501
47  6.7501
48  6.7501
49  6.7501
50  6.7501
51 20.2503
   /


   P(I)  arc probabilities
   /
  1 1.00
  2 0.30
  3 0.70
  4 0.50
  5 0.80
    1.00
  7 0.30
  8 0.70
  9 0.50
 10 0.80
 11 1.00
 12 0.30
 13 0.70
 14 0.50
 15 0.80
 16 0.80
 17 0.70
 18 0.30
 19 1.00
 20 1.00
 21 1.00
 22 1.00
 23 1.00
 24 1.00
 25 0.60
 26 0.30
```

```
27 1.00
28 1.00
29 1.00
30 1.00
31 1.00
32 0.60
33 0.70
34 1.00
35 0.70
36 1.00
37 0.60
38 1.00
39 0.30
40 0.60
41 0.70
42 0.60
43 0.60
44 0.30
45 0.60
46 0.60
47 0.30
48 0.30
49 0.60
50 0.70
51 1.00
   /;

SCALAR C cost of increasing arc rel by .1  /  400 /;
SCALAR B total budget available  / 8000 /;

VARIABLES
   X(I)  .1 arc rel increase
   Z     objective for rel index ;

POSITIVE VARIABLE X;

EQUATIONS

   MAXINDEX
   REL(I)
   BUDGET ;

MAXINDEX .. Z =E= SUM(I, R(I)*(P(I)+X(I))) ;
REL(I) .. P(I)+(.1*X(I)) =L= 1 ;
BUDGET .. SUM(I, C*X(I)) =L= B ;

MODEL RELHUER  /ALL/ ;

OPTION LIMROW = 0

SOLVE RELHUER USING LP MAXIMIZING Z ;
```

```
DISPLAY X.L ;
```

B.2 Linear Capacity Improvement Model

```
$OFFSYMXREF OFFSYMLIST

SETS
   I   arcs  /1 * 51/
   J   paths /1 * 63/;

PARAMETERS


  U(I)  arc capacities

    /
  1      0
  2      0
  3      0
  4      0
  5      0
  6      0
  7      0
  8      0
  9      0
 10      0
 11      0
 12      0
 13      0
 14      0
 15      0
 16      0
 17      0
 18      0
 19      0
 20      0
 21      0
 22      0
 23      0
 24   1200
 25   1200
 26   1200
 27   1200
 28   1200
 29   1200
 30   1200
 31   1200
 32   1200
 33   1200
```

```
34  4800
35  4800
36  4800
37  4800
38  4800
39  4800
40  4800
41  4800
42  4800
43  4800
44  4800
45  4800
46  4800
47  4800
48  4800
49  4800
50  4800
51     0
   /

   P(I)  arc probabilities
   /
 1 1.00
 2 0.30
 3 0.70
 4 0.50
 5 0.80
 6 1.00
 7 0.30
 8 0.70
 9 0.50
10 0.80
11 1.00
12 0.30
13 0.70
14 0.50
15 0.80
16 0.80
17 0.70
18 0.30
19 1.00
20 1.00
21 1.00
22 1.00
23 1.00
24 1.00
25 0.60
26 0.30
27 1.00
28 1.00
29 1.00
```

```
30 1.00
31 1.00
32 0.60
33 0.70
34 1.00
35 0.70
36 1.00
37 0.60
38 1.00
39 0.30
40 0.60
41 0.70
42 0.60
43 0.60
44 0.30
45 0.60
46 0.60
47 0.30
48 0.30
49 0.60
50 0.70
51 1.00
   /

   A(I,J)  arc-path matrix
   /
 1.43 1
 1.44 1
 1.45 1
 1.46 1
 1.47 1
 1.48 1
 1.49 1
 1.50 1
 1.51 1
 1.52 1
 1.53 1
 1.54 1
 1.55 1
 1.56 1
 1.57 1
 1.58 1
 1.59 1
 1.60 1
 1.61 1
 1.62 1
 1.63 1
 2.4 1
 2.5 1
 2.6 1
 2.7 1
```

```
2.8 1
2.9 1
2.10 1
2.11 1
2.12 1
2.13 1
2.14 1
2.15 1
2.16 1
2.17 1
2.18 1
2.19 1
2.20 1
2.21 1
2.22 1
2.23 1
2.24 1
3.25 1
3.26 1
3.27 1
3.28 1
3.29 1
3.30 1
3.31 1
3.32 1
3.33 1
3.34 1
3.35 1
3.36 1
3.37 1
3.38 1
3.39 1
3.40 1
3.41 1
3.42 1
4.1 1
4.2 1
4.3 1
5.7 1
5.8 1
5.9 1
5.10 1
5.11 1
5.12 1
5.13 1
5.14 1
5.15 1
5.16 1
5.17 1
5.18 1
5.19 1
```

```
5.20 1
5.21 1
5.22 1
5.23 1
5.24 1
6.7 1
6.8 1
6.9 1
6.16 1
6.17 1
6.18 1
6.25 1
6.26 1
6.27 1
6.34 1
6.35 1
6.36 1
6.43 1
6.44 1
6.45 1
6.52 1
6.53 1
6.54 1
7.10 1
7.11 1
7.12 1
7.19 1
7.20 1
7.21 1
7.28 1
7.29 1
7.30 1
7.37 1
7.38 1
7.39 1
7.46 1
7.47 1
7.48 1
7.55 1
7.56 1
7.57 1
8.13 1
8.14 1
8.15 1
8.22 1
8.23 1
8.24 1
8.31 1
8.32 1
8.33 1
8.40 1
```

```
 8.41 1
 8.42 1
 8.49 1
 8.50 1
 8.51 1
 8.58 1
 8.59 1
 8.60 1
 9.34 1
 9.35 1
 9.36 1
 9.37 1
 9.38 1
 9.39 1
 9.40 1
 9.41 1
 9.42 1
10.7 1
10.8 1
10.9 1
10.10 1
10.11 1
10.12 1
10.13 1
10.14 1
10.15 1
11.16 1
11.17 1
11.18 1
11.19 1
11.20 1
11.21 1
11.22 1
11.23 1
11.24 1
11.25 1
11.26 1
11.27 1
11.28 1
11.29 1
11.30 1
11.31 1
11.32 1
11.33 1
12.43 1
12.44 1
12.45 1
12.46 1
12.47 1
12.48 1
12.49 1
```

```
12.50 1
12.51 1
13.52 1
13.53 1
13.54 1
13.55 1
13.56 1
13.57 1
13.58 1
13.59 1
13.60 1
14.1 1
14.2 1
14.3 1
14.4 1
14.5 1
14.6 1
14.7 1
14.8 1
14.9 1
14.10 1
14.11 1
14.12 1
14.13 1
14.14 1
14.15 1
14.16 1
14.17 1
14.18 1
14.19 1
14.20 1
14.21 1
14.22 1
14.23 1
14.24 1
14.25 1
14.26 1
14.27 1
14.28 1
14.29 1
14.30 1
14.31 1
14.32 1
14.33 1
14.34 1
14.35 1
14.36 1
14.37 1
14.38 1
14.39 1
14.40 1
```

```
14.41 1
14.42 1
14.43 1
14.44 1
14.45 1
14.46 1
14.47 1
14.48 1
14.49 1
14.50 1
14.51 1
14.52 1
14.53 1
14.54 1
14.55 1
14.56 1
14.57 1
14.58 1
14.59 1
14.60 1
14.61 1
14.62 1
14.63 1
15.7 1
15.8 1
15.9 1
15.10 1
15.11 1
15.12 1
15.13 1
15.14 1
15.15 1
15.16 1
15.17 1
15.18 1
15.19 1
15.20 1
15.21 1
15.22 1
15.23 1
15.24 1
15.25 1
15.26 1
15.27 1
15.28 1
15.29 1
15.30 1
15.31 1
15.32 1
15.33 1
15.34 1
```

```
15.35 1
15.36 1
15.37 1
15.38 1
15.39 1
15.40 1
15.41 1
15.42 1
15.43 1
15.44 1
15.45 1
15.46 1
15.47 1
15.48 1
15.49 1
15.50 1
15.51 1
15.52 1
15.53 1
15.54 1
15.55 1
15.56 1
15.57 1
15.58 1
15.59 1
15.60 1
16.1 1
16.2 1
16.3 1
16.4 1
16.5 1
16.6 1
16.7 1
16.8 1
16.9 1
16.10 1
16.11 1
16.12 1
16.13 1
16.14 1
16.15 1
16.16 1
16.17 1
16.18 1
16.19 1
16.20 1
16.21 1
16.22 1
16.23 1
16.24 1
16.25 1
```

```
16.26 1
16.27 1
16.28 1
16.29 1
16.30 1
16.31 1
16.32 1
16.33 1
16.34 1
16.35 1
16.36 1
16.37 1
16.38 1
16.39 1
16.40 1
16.41 1
16.42 1
16.43 1
16.44 1
16.45 1
16.46 1
16.47 1
16.48 1
16.49 1
16.50 1
16.51 1
16.52 1
16.53 1
16.54 1
16.55 1
16.56 1
16.57 1
16.58 1
16.59 1
16.60 1
16.61 1
16.62 1
16.63 1
17.3 1
17.6 1
17.9 1
17.12 1
17.15 1
17.18 1
17.21 1
17.24 1
17.27 1
17.30 1
17.33 1
17.36 1
17.39 1
```

```
17.42 1
17.45 1
17.48 1
17.51 1
17.54 1
17.57 1
17.60 1
17.63 1
18.2 1
18.5 1
18.8 1
18.11 1
18.14 1
18.17 1
18.20 1
18.23 1
18.26 1
18.29 1
18.32 1
18.35 1
18.38 1
18.41 1
18.44 1
18.47 1
18.50 1
18.53 1
18.56 1
18.59 1
18.62 1
19.1 1
19.4 1
19.7 1
19.10 1
19.13 1
19.16 1
19.19 1
19.22 1
19.25 1
19.28 1
19.31 1
19.34 1
19.37 1
19.40 1
19.43 1
19.46 1
19.49 1
19.52 1
19.55 1
19.58 1
19.61 1
20.1 1
```

```
20.2 1
20.3 1
21.4 1
21.5 1
21.6 1
21.7 1
21.8 1
21.9 1
21.10 1
21.11 1
21.12 1
21.13 1
21.14 1
21.15 1
21.16 1
21.17 1
21.18 1
21.19 1
21.20 1
21.21 1
21.22 1
21.23 1
21.24 1
22.25 1
22.26 1
22.27 1
22.28 1
22.29 1
22.30 1
22.31 1
22.32 1
22.33 1
22.34 1
22.35 1
22.36 1
22.37 1
22.38 1
22.39 1
22.40 1
22.41 1
22.42 1
23.43 1
23.44 1
23.45 1
23.46 1
23.47 1
23.48 1
23.49 1
23.50 1
23.51 1
23.52 1
```

```
23.53 1
23.54 1
23.55 1
23.56 1
23.57 1
23.58 1
23.59 1
23.60 1
23.61 1
23.62 1
23.63 1
24.1 1
24.2 1
24.3 1
25.4 1
25.5 1
25.6 1
26.7 1
26.8 1
26.9 1
26.10 1
26.11 1
26.12 1
26.13 1
26.14 1
26.15 1
26.16 1
26.17 1
26.18 1
26.19 1
26.20 1
26.21 1
26.22 1
26.23 1
26.24 1
27.25 1
27.26 1
27.27 1
27.28 1
27.29 1
27.30 1
27.31 1
27.32 1
27.33 1
28.34 1
28.35 1
28.36 1
28.37 1
28.38 1
28.39 1
28.40 1
```

```
28.41 1
28.42 1
29.43 1
29.44 1
29.45 1
29.46 1
29.47 1
29.48 1
29.49 1
29.50 1
29.51 1
30.52 1
30.53 1
30.54 1
30.55 1
30.56 1
30.57 1
30.58 1
30.59 1
30.60 1
31.61 1
31.62 1
31.63 1
32.7 1
32.8 1
32.9 1
32.10 1
32.11 1
32.12 1
32.13 1
32.14 1
32.15 1
33.16 1
33.17 1
33.18 1
33.19 1
33.20 1
33.21 1
33.22 1
33.23 1
33.24 1
34.34 1
34.35 1
34.36 1
34.37 1
34.38 1
34.39 1
34.40 1
34.41 1
34.42 1
35.43 1
```

```
35.44 1
35.45 1
35.46 1
35.47 1
35.48 1
35.49 1
35.50 1
35.51 1
36.52 1
36.53 1
36.54 1
36.55 1
36.56 1
36.57 1
36.58 1
36.59 1
36.60 1
37.7 1
37.8 1
37.9 1
37.10 1
37.11 1
37.12 1
37.13 1
37.14 1
37.15 1
38.16 1
38.17 1
38.18 1
38.19 1
38.20 1
38.21 1
38.22 1
38.23 1
38.24 1
38.25 1
38.26 1
38.27 1
38.28 1
38.29 1
38.30 1
38.31 1
38.32 1
38.33 1
39.7 1
39.8 1
39.9 1
39.16 1
39.17 1
39.18 1
39.25 1
```

```
39.26 1
39.27 1
39.34 1
39.35 1
39.36 1
39.43 1
39.44 1
39.45 1
39.52 1
39.53 1
39.54 1
40.10 1
40.11 1
40.12 1
40.19 1
40.20 1
40.21 1
40.28 1
40.29 1
40.30 1
40.37 1
40.38 1
40.39 1
40.46 1
40.47 1
40.48 1
40.55 1
40.56 1
40.57 1
41.13 1
41.14 1
41.15 1
41.22 1
41.23 1
41.24 1
41.31 1
41.32 1
41.33 1
41.40 1
41.41 1
41.42 1
41.49 1
41.50 1
41.51 1
41.58 1
41.59 1
41.60 1
42.7 1
42.8 1
42.9 1
42.16 1
```

```
42.17 1
42.18 1
42.25 1
42.26 1
42.27 1
42.34 1
42.35 1
42.36 1
42.43 1
42.44 1
42.45 1
42.52 1
42.53 1
42.54 1
43.10 1
43.11 1
43.12 1
43.19 1
43.20 1
43.21 1
43.28 1
43.29 1
43.30 1
43.37 1
43.38 1
43.39 1
43.46 1
43.47 1
43.48 1
43.55 1
43.56 1
43.57 1
44.13 1
44.14 1
44.15 1
44.22 1
44.23 1
44.24 1
44.31 1
44.32 1
44.33 1
44.40 1
44.41 1
44.42 1
44.49 1
44.50 1
44.51 1
44.58 1
44.59 1
44.60 1
45.1 1
```

```
45.4 1
45.7 1
45.10 1
45.13 1
45.16 1
45.19 1
45.22 1
45.25 1
45.28 1
45.31 1
45.34 1
45.37 1
45.40 1
45.43 1
45.46 1
45.49 1
45.52 1
45.55 1
45.58 1
45.61 1
46.2 1
46.5 1
46.8 1
46.11 1
46.14 1
46.17 1
46.20 1
46.23 1
46.26 1
46.29 1
46.32 1
46.35 1
46.38 1
46.41 1
46.44 1
46.47 1
46.50 1
46.53 1
46.56 1
46.59 1
46.62 1
47.3 1
47.6 1
47.9 1
47.12 1
47.15 1
47.18 1
47.21 1
47.24 1
47.27 1
47.30 1
```

```
47.33 1
47.36 1
47.39 1
47.42 1
47.45 1
47.48 1
47.51 1
47.54 1
47.57 1
47.60 1
47.63 1
48.1 1
48.4 1
48.7 1
48.10 1
48.13 1
48.16 1
48.19 1
48.22 1
48.25 1
48.28 1
48.31 1
48.34 1
48.37 1
48.40 1
48.43 1
48.46 1
48.49 1
48.52 1
48.55 1
48.58 1
48.61 1
49.2 1
49.5 1
49.8 1
49.11 1
49.14 1
49.17 1
49.20 1
49.23 1
49.26 1
49.29 1
49.32 1
49.35 1
49.38 1
49.41 1
49.44 1
49.47 1
49.50 1
49.53 1
49.56 1
```

```
49.59 1
49.62 1
50.3 1
50.6 1
50.9 1
50.12 1
50.15 1
50.18 1
50.21 1
50.24 1
50.27 1
50.30 1
50.33 1
50.36 1
50.39 1
50.42 1
50.45 1
50.48 1
50.51 1
50.54 1
50.57 1
50.60 1
50.63 1
51.1 1
51.2 1
51.3 1
51.4 1
51.5 1
51.6 1
51.7 1
51.8 1
51.9 1
51.10 1
51.11 1
51.12 1
51.13 1
51.14 1
51.15 1
51.16 1
51.17 1
51.18 1
51.19 1
51.20 1
51.21 1
51.22 1
51.23 1
51.24 1
  1.25 1
51.26 1
51.27 1
51.28 1
```

```
51.29 1
51.30 1
51.31 1
51.32 1
51.33 1
51.34 1
51.35 1
51.36 1
51.37 1
51.38 1
51.39 1
51.40 1
51.41 1
51.42 1
51.43 1
51.44 1
51.45 1
51.46 1
51.47 1
51.48 1
51.49 1
51.50 1
51.51 1
51.52 1
51.53 1
51.54 1
51.55 1
51.56 1
51.57 1
51.58 1
51.59 1
51.60 1
51.61 1
51.62 1
51.63 1
   /;

SCALAR C cost of increasing arc cap by 1 / 1 /;

PARAMETER R(J) path rel ;
   R(J) = PROD(I $ A(I,J), P(I)) ;

VARIABLES
   X(I)  arc capacity increase
   F(J)  flow on path J
   Z     network lower bound maxflo ;

POSITIVE VARIABLES F,X ;

EQUATIONS
```

```
   MAXFLO
   PATHFLO(I)
   MAXCAP(I)
   BUDGET ;

MAXFLO .. Z =E= SUM(J, R(J)*F(J)) ;

PATHFLO(I)$(U(I) ne 0) ..SUM(J $ A(I,J), F(J)) =L= U(I)+X(I) ;

MAXCAP(I) .. X(I) =L= 2400 ;

BUDGET .. SUM(I, C*X(I)) =L= 8000 ;

MODEL CAPINV  /ALL/ ;

OPTION LIMROW = 0

SOLVE CAPINV USING LP MAXIMIZING Z ;

DISPLAY X.L, F.L ;
```

B.3 Nonlinear Combined Improvement Model

```
$OFFSYMXREF OFFSYMLIST

SETS
   I   arcs  /1 * 51/
   J   paths /1 * 63/;

PARAMETERS


  U(I)  arc capacities

   /
 1      0
 2      0
 3      0
 4      0
 5      0
 6      0
 7      0
 8      0
 9      0
10      0
11      0
12      0
13      0
14      0
15      0
16      0
17      0
18      0
19      0
20      0
21      0
22      0
23      0
24   1200
25   1200
26   1200
27   1200
28   1200
29   1200
30   1200
31   1200
32   1200
33   1200
34   4800
```

```
35  4800
36  4800
37  4800
38  4800
39  4800
40  4800
41  4800
42  4800
43  4800
44  4800
45  4800
46  4800
47  4800
48  4800
49  4800
50  4800
51     0
  /

  P(I)  arc probabilities
  /
 1 1.00
 2 0.30
 3 0.70
 4 0.50
 5 0.80
 6 1.00
 7 0.30
 8 0.70
 9 0.50
10 0.80
11 1.00
12 0.30
13 0.70
14 0.50
15 0.80
16 0.80
17 0.70
18 0.30
19 1.00
20 1.00
21 1.00
22 1.00
23 1.00
24 1.00
25 0.60
26 0.30
27 1.00
28 1.00
29 1.00
30 1.00
```

```
31 1.00
32 0.60
33 0.70
34 1.00
35 0.70
36 1.00
37 0.60
38 1.00
39 0.30
40 0.60
41 0.70
42 0.60
43   60
44 0.30
45 0.60
46 0.60
47 0.30
48 0.30
49 0.60
50 0.70
51 1.00
   /

   A(I,J)  arc-path matrix
   /
 1.43 1
 1.44 1
 1.45 1
 1.46 1
 1.47 1
 1.48 1
 1.49 1
 1.50 1
 1.51 1
 1.52 1
 1.53 1
 1.54 1
 1.55 1
 1.56 1
 1.57 1
 1.58 1
 1.59 1
 1.60 1
 1.61 1
 1.62 1
 1.63 1
 2.4 1

   .
   .
   .
```

```
(data same as linear model)

   .
   .
   .

51.60 1
51.61 1
51.62 1
51.63 1
   /;

SCALAR CC cost of increasing arc cap by 1  / 1 /;
SCALAR CR cost of increasing arc surv by .1  /400/;


VARIABLES
   C(I)  arc cap increase
   X(I)  arc surv increase
   F(J)  flow on path J
   Z     network lower bound maxflo ;

POSITIVE VARIABLES C,F,X ;

EQUATIONS

   MAXFLO
   PATHFLO(I)
   MAXCAP(I)
   REL(I)
   BUDGET ;

MAXFLO .. Z =E= SUM(J, PROD(I $ A(I,J), P(I)+(.1*X(I)))*F(J)) ;

PATHFLO(I)$(U(I) ne 0) ..SUM(J $ A(I,J), F(J)) =L= U(I)+C(I) ;

REL(I) .. P(I)+(.1*X(I)) =L= 1 ;

BUDGET .. SUM(I, CR*X(I) + CC*C(I)) =L= 8000 ;

MAXCAP(I) .. C(I) +U(I) =L= 4800 ;

MODEL AMAX  /ALL/ ;

OPTION LIMROW = 0

SOLVE AMAX USING NLP MAXIMIZING Z ;

DISPLAY X.L, C.L, F.L ;
```

Appendix C. *Formula Input Files*

C.1 Network A

```
arc(s,20).
arc(s,21).
arc(s,22).
arc(s,23).
arc(1,29).
arc(1,30).
arc(1,31).
arc(2,25).
arc(2,26).
arc(3,27).
arc(3,28).
arc(4,24).
arc(5,32).
arc(5,33).
arc(6,42).
arc(7,43).
arc(8,44).
arc(9,34).
arc(10,37).
arc(11,38).
arc(12,35).
arc(13,36).
arc(14,45).
arc(14,46).
arc(14,47).
arc(15,39).
arc(15,40).
arc(15,41).
arc(16,51).
arc(17,50).
arc(18,49).
arc(19,48).
arc(20,4).
arc(21,2).
arc(22,3).
arc(23,1).
arc(24,14).
arc(25,14).
arc(26,5).
arc(27,11).
arc(28,9).
arc(29,12).
arc(30,13).
arc(31,14).
arc(32,10).
arc(33,11).
arc(34,15).
arc(35,15).
arc(36,15).
arc(37,15).
arc(38,15).
arc(39,6).
arc(40,7).
arc(41,8).
arc(42,14).
arc(43,14).
arc(44,14).
arc(45,19).
arc(46,18).
arc(47,17).
arc(48,16).
arc(49,16).
arc(50,16).
arc(51,t).
prob(s,1).
prob(1,1).
prob(2,0.3).
prob(3,0.7).
prob(4,0.5).
prob(5,0.8).
prob(6,1).
prob(7,0 3).
prob(8,0.7).
prob(9,0.5).
prob(10,0.8).
prob(11,1).
prob(12,0.3).
prob(13,0.7).
prob(14,0.5).
prob(15,0.8).
prob(16,0.8).
prob(17,0.7).
prob(18,0.3).
prob(19,1).
prob(20,1).
prob(21,1).
prob(22,1).
prob(23,1).
prob(24,1).
prob(25,0.6).
prob(26,0.3).
prob(27,1).
prob(28,1).
prob(29,1).
prob(30,1).
prob(31,1).
prob(32,0.6).
prob(33,0.7).
prob(34,1).
prob(35,0.7).
prob(36,1).
prob(37,0.6).
prob(38,1).
prob(39,0.3).
prob(40,0.6).
prob(41,0.7).
prob(42,0.6).
prob(43,0.6).
prob(44,0.3).
prob(45,0.6).
prob(46,0.6).
prob(47,0.3).
prob(48,0.3).
prob(49,0.6).
prob(50,0.7).
prob(51,1).
prob(t,1).
cap(1,*).
cap(2,*).
cap(3,*).
cap(4,*).
cap(5,*).
cap(6,*).
cap(7,*).
cap(8,*).
cap(9,*).
cap(10,*).
cap(11,*).
cap(12,*).
cap(13,*).
cap(14,*).
cap(15,*).
cap(16,*).
cap(17,*).
cap(18,*).
cap(19,*).
cap(20,*).
cap(21,*).
cap(22,*).
cap(23,*).
cap(24,1200).
cap(25,1200).
cap(26,1200).
cap(27,1200).
cap(28,1200).
cap(29,1200).
cap(30,1200).
cap(31,1200).
cap(32,1200).
cap(33,1200).
cap(34,4800).
cap(35,4800).
cap(36,4800).
cap(37,4800).
cap(38,4800).
cap(39,4800).
cap(40,4800).
cap(41,4800).
cap(42,4800).
cap(43,4800).
cap(44,4800).
cap(45,4800).
cap(46,4800).
cap(47,4800).
cap(48.4800).
cap(49,4800).
cap(50,4800).
cap(51,*).
```

C.2 Network B

```
arc(s,50).
arc(s,51).
arc(s,52).
arc(s,53).
arc(s,54).
arc(s,55).
arc(56,t).
arc(57,t).
arc(58,t).
arc(59,t).
arc(60,t).
arc(61,t).
arc(62,t).
arc(63,t).
arc(64,t).
arc(65,t).
arc(50,38).
arc(51,39).
arc(52,40).
arc(53,8).
arc(53,9).
arc(54,10).
arc(55,41).
arc(1,42).
arc(5,42).
arc(2,15).
arc(3,15).
arc(4,43).
arc(6,43).
arc(14,44).
arc(15,44).
arc(16,44).
arc(17,45).
arc(18,23).
arc(18,24).
arc(19,46).
arc(10,47).
arc(26,48).
arc(7,49).
arc(8,49).
arc(11,49).
arc(23,49).
arc(25,49).
arc(27,49).
arc(20,56).
arc(28,56).
arc(21,57).
arc(29,57).
arc(22,58).
arc(30,58).
arc(24,59).
arc(31,59).
arc(32,60).
arc(33,61).
arc(34,62).
arc(9,63).
arc(35,63).
arc(12,64).
arc(36,64).
arc(13,65).
arc(37,65).
arc(38,1).
arc(38,2).
arc(39,3).
arc(39,4).
arc(40,5).
arc(40,6).
arc(40,7).
arc(41,11).
arc(41,12).
arc(41,13).
arc(42,14).
arc(43,16).
arc(44,17).
arc(44,18).
arc(44,19).
arc(45,20).
arc(45,21).
arc(45,22).
arc(46,25).
arc(47,26).
arc(48,27).
arc(49,28).
arc(49,29).
arc(49,30).
arc(49,31).
arc(49,32).
arc(49,33).
arc(49,34).
arc(49,35).
arc(49,36).
arc(49,37).
prob(s,1.0).
prob(1,0.8).
prob(2,0.8).
prob(3,0.5).
prob(4,0.5).
prob(5,0.8).
prob(6,0.5).
prob(7,0.6).
prob(8,0.8).
prob(9,0.5).
prob(10,0.8).
prob(11,0.5).
prob(12,0.6).
prob(13,0.6).
prob(14,0.5).
prob(15,0.7).
prob(16,0.5).
prob(17,0.5).
prob(18,0.7).
prob(19,0.7).
prob(20,0.5).
prob(21,0.5).
prob(22,0.5).
prob(23,0.7).
prob(24,0.5).
prob(25,0.7).
prob(26,0.8).
prob(27,0.8).
prob(28,0.6).
prob(29,0.6).
prob(30,0.6).
prob(31,0.6).
prob(32,0.6).
prob(33,0.6).
prob(34,0.6).
prob(35,0.6).
prob(36,0.6).
prob(37,0.6).
prob(38,0.7).
prob(39,0.15).
prob(40,0.03).
prob(41,0.04).
prob(42,0.4).
prob(43,0.01).
prob(44,0.7).
prob(45,0.11).
prob(46,0.06).
prob(47,0.09).
prob(48,0.18).
prob(49,0.07).
prob(50,1.0).
prob(51,1.0).
prob(52,1.0).
prob(53,1.0).
prob(54,1.0).
prob(55,1.0).
prob(56,1.0).
prob(57,1.0).
prob(58,1.0).
prob(59,1.0).
prob(60,1.0).
prob(61,1.0).
prob(62,1.0).
prob(63,1.0).
prob(64,1.0).
prob(65,1.0).
prob(t,1.0).
cap(1,150).
cap(2,200).
cap(3,750).
cap(4,750).
cap(5,200).
cap(6,750).
cap(7,150).
cap(8,200).
cap(9.600).
cap(10,1200).
cap(11,1200).
cap(12,75).
cap(13,75).
cap(14,1200).
cap(15,1200).
cap(16,2400).
cap(17,1200).
cap(18,1200).
cap(19,1200).
cap(20,1200).
cap(21,75).
cap(22,1200).
cap(23,1200).
cap(24,1200).
cap(25,600).
cap(26,1200).
cap(27,1200).
cap(28,75).
cap(29,75).
cap(30,75).
cap(31,75).
cap(32,75).
cap(33,75).
cap(34,75).
cap(35,75).
cap(36,75).
cap(37,75).
cap(38,*).
cap(39,*).
cap(40,*).
cap(41,*).
cap(42,*).
cap(43,*).
cap(44,*).
cap(45,*).
cap(46,*).
cap(47,*).
cap(48,*).
cap(49,*).
cap(50,*).
cap(51,*).
cap(52,*).
cap(53,*).
cap(54,*).
cap(55,*).
cap(56,*).
cap(57,*).
cap(58,*).
cap(59,*).
cap(60,*).
cap(61,*).
cap(62,*).
cap(63,*).
cap(64,*).
cap(65,*).
```

C.3 Network C

```
cost(1,1).
cost(2,1).
cost(3,1).
cost(4,1).
cost(5,1).
cost(6,1).
cost(7,1).
cost(8,1).
cost(9,1).
cost(10,1).
cost(11,1).
cost(12,1).
cost(13,1).
cost(14,1).
cost(15,1).
cost(16,1).
cost(17,1).
cost(18,1).
cost(19,1).
cost(20,1).
cost(21,1).
cost(22,1).
cost(23,1).
cost(24,1).
cost(25,1).
cost(26,1).
cost(27,1).
cost(28,1).
cost(29,1).
cost(30,1).
cost(31,1).
cost(32,1).
cost(33,1).
cost(34,1).
cost(35,1).
cost(36,1).
cost(37,1).
cost(38,1).
cost(39,1).
cost(40,1).
cost(41,1).
cost(42,1).
cost(43,1).
cost(44,1).
cost(45,1).
cost(46,1).
cost(47,1).
cost(48,1).
cost(49,1).
cost(50,1).
cost(51,1).
cost(52,1).
cost(53,1).
cost(54,1).
rcost(1,1).
rcost(4,1).
rcost(5,1).
rcost(6,1).
rcost(8,1).
rcost(10,1).
rcost(11,1).
rcost(12,1).
rcost(13,1).
rcost(14,1).
rcost(15,1).
rcost(18,1).
rcost(19,1).
rcost(20,1).
rcost(21,1).
rcost(22,1).
rcost(25,1).
rcost(26,1).
rcost(27,1).
rcost(28,1).
rcost(29,1).
rcost(32,1).
rcost(33,1).
rcost(34,1).
rcost(35,1).
rcost(36,1).
rcost(37,1).
rcost(40,1).
rcost(41,1).
rcost(42,1).
rcost(43,1).
rcost(44,1).
rcost(47,1).
rcost(48,1).
rcost(49,1).
rcost(50,1).
rcost(51,1).
rcost(54,1).
arc(s,87).
arc(s,88).
arc(s,89).
arc(s,90).
arc(s,91).
arc(s,92).
arc(93,t).
arc(94,t).
arc(95,t).
arc(96,t).
arc(97,t).
arc(98,t).
arc(99,t).
arc(100,t).
arc(101,t).
arc(102,t).
arc(103,t).
arc(104,t).
arc(105,t).
arc(106,t).
arc(107,t).
arc(108,t).
arc(109,t).
arc(87,55).
arc(88,56).
arc(89,57).
arc(90,58).
arc(91,5).
arc(92,59).
arc(3,60).
arc(5,61).
arc(6,62).
arc(7,10).
arc(1,63).
arc(2,63).
arc(4,63).
arc(8,63).
arc(9,63).
arc(10,63).
arc(11,64).
arc(16,65).
arc(17,66).
arc(18,40).
arc(18,41).
arc(19,67).
arc(20,68).
arc(21,69).
arc(22,70).
arc(23,52).
arc(24,71).
arc(25,72).
arc(12,73).
arc(26,73).
arc(33,73).
arc(13,74).
arc(27,74).
arc(34,74).
arc(28,93).
arc(35,93).
arc(29,75).
arc(36,75).
arc(30,76).
arc(37,76).
arc(31,77).
arc(38,77).
arc(32,78).
arc(39,78).
arc(42,94).
arc(47,94).
arc(40,79).
arc(43,79).
arc(41,80).
arc(44,80).
arc(45,81).
arc(48,81).
arc(49,82).
arc(52,95).
arc(46,83).
arc(53,83).
arc(54,84).
arc(14,85).
arc(50,85).
arc(15,86).
arc(51,86).
arc(55,1).
arc(56,2).
arc(57,3).
arc(58,4).
arc(59,6).
arc(60,7).
arc(61,8).
arc(62,9).
arc(63,11).
arc(63,12).
arc(63,13).
arc(63,14).
arc(63,15).
arc(64,16).
arc(64,17).
arc(64,18).
arc(64,19).
arc(64,20).
arc(64,21).
arc(64,22).
arc(64,23).
arc(64,24).
arc(64,25).
arc(65,26).
arc(65,27).
arc(65,28).
arc(65,29).
arc(65,30).
arc(65,31).
arc(65,32).
arc(66,33).
arc(66,34).
arc(66,35).
arc(66,36).
arc(66,37).
arc(66,38).
arc(66,39).
arc(67,42).
arc(67,43).
arc(67,44).
arc(67,45).
arc(67,46).
arc(68,47).
arc(68,48).
arc(68,49).
arc(69,50).
arc(70,51).
arc(71,53).
arc(72,54).
arc(73,96).
arc(74,97).
arc(75,98).
arc(76,99).
arc(77,100).
arc(78,101).
arc(79,102).
arc(80,103).
arc(81,104).
arc(82,105).
arc(83,106).
arc(84,107).
arc(85,108).
arc(86,109).
prob(s,1).
prob(1,0.900).
prob(2,1.000).
prob(3,1.000).
prob(4,0.600).
prob(5,0.300).
prob(6,0.600).
prob(7,1.000).
prob(8,0.900).
prob(9,1.000).
prob(10,0.700).
prob(11,0.900).
prob(12,0.600).
prob(13,0.300).
prob(14,0.600).
prob(15,0.700).
prob(16,1.000).
prob(17,1.000).
prob(18,0.600).
prob(19,0.300).
prob(20,0.600).
prob(21,0.700).
prob(22,0.900).
prob(23,1.000).
prob(24,1.000).
prob(25,0.600).
prob(26,0.300).
prob(27,0.600).
prob(28,0.700).
prob(29,0.900).
prob(30,1.000).
prob(31,1.000).
prob(32,0.300).
prob(33,0.600).
prob(34,0.300).
prob(35,0.600).
prob(36,0.700).
prob(37,0.900).
prob(38,1.000).
prob(39,1.000).
prob(40,0.600).
prob(41,0.300).
prob(42,0.600).
prob(43,0.700).
prob(44,0.900).
prob(45,1.000).
prob(46,1.000).
prob(47,0.600).
prob(48,0.300).
prob(49,0.600).
prob(50,0.700).
prob(51,0.900).
prob(52,1.000).
prob(53,1.000).
prob(54,0.600).
prob(55,0.300).
prob(56,0.700).
prob(57,0.500).
prob(58,0.800).
prob(59,0.300).
prob(60,0.700).
prob(61,0.500).
prob(62,0.800).
prob(63,0.700).
prob(64,0.700).
prob(65,0.500).
prob(66,0.800).
prob(67,0.300).
prob(68,0.700).
prob(69,0.500).
prob(70,0.800).
prob(71,0.300).
prob(72,0.700).
prob(73,0.500).
prob(74,0.800).
prob(75,0.300).
prob(76,0.700).
prob(77,0.500).
prob(78,0.800).
prob(79,0.300).
prob(80,0.700).
prob(81,0.500).
prob(82,0.800).
prob(83,0.300).
prob(84,0.700).
prob(85,0.500).
prob(86,0.800).
prob(87,1.000).
prob(88,1.000).
prob(89,1.000).
prob(90,1.000).
prob(91,1.000).
prob(92,1.000).
prob(93,1.000).
prob(94,1.000).
prob(95,1.000).
prob(96,1.000).
prob(97,1.000).
prob(98,1.000).
prob(99,1.000).
prob(100,1.000).
prob(101,1.000).
prob(102,1.000).
prob(103,1.000).
prob(104,1.000).
prob(105,1.000).
prob(106,1.000).
prob(107,1.000).
prob(108,1.000).
prob(109,1.000).
prob(t,1).
cap(1,1200.000).
cap(2,1200.000).
cap(3,300.000).
cap(4,1200.000).
cap(5,1200.000).
cap(6,1200.000).
cap(7,300.000).
cap(8,1200.000).
cap(9,1200.000).
cap(10,300.000).
cap(11,9600.000).
cap(12,75.000).
cap(13,75.000).
cap(14,1200.000).
cap(15,1200.000).
cap(16,4800.000).
cap(17,4800.000).
cap(18,4800.000).
cap(19,4800.000).
cap(20,4800.000).
cap(21,2400.000).
cap(22,4800.000).
cap(23,4800.000).
cap(24,4800.000).
cap(25,2400.000).
cap(26,4800.000).
cap(27,4800.000).
cap(28,2400.000).
cap(29,1200.000).
cap(30,1200.000).
cap(31,1200.000).
cap(32,1200.000).
cap(33,4800.000).
cap(34,4800.000).
cap(35,2400.000).
cap(36,1200.000).
cap(37,1200.000).
cap(38,1200.000).
cap(39,1200.000).
cap(40,2400.000).
cap(41,1200.000).
cap(42,300.000).
cap(43,2400.000).
cap(44,1200.000).
cap(45,300.000).
cap(46,2400.000).
cap(47,1200.000).
cap(48,300.000).
cap(49,300.000).
cap(50,1200.000).
cap(51,1200.000).
cap(52,2400.000).
cap(53,2400.000).
cap(54,600.000).
cap(55,*).
cap(56,*).
cap(57,*).
cap(58,*).
cap(59,*).
cap(60,*).
cap(61,*).
cap(62,*).
cap(63,*).
cap(64,*).
cap(65,*).
cap(66,*).
cap(67,   .
cap(68,*).
cap(69,*).
cap(70,*).
cap(71,*).
cap(72,*).
cap(73,*).
cap(74,*).
cap(75,*).
cap(76,*).
cap(77,*).
cap(78,*).
cap(79,*).
cap(80,*).
cap(81,*).
cap(82,*).
cap(83,*).
cap(84,*).
cap(85,*).
cap(86,*).
cap(87,*).
cap(88,*).
cap(89,*).
cap(90,*).
cap(91,*).
cap(92,*).
cap(93,*).
cap(94,*).
cap(95,*).
cap(96,*).
cap(97,*).
cap(98,*).
cap(99,*).
cap(100,*).
cap(101,*).
cap(102,*).
cap(103,*).
cap(104,*).
cap(105,*).
cap(106,*).
cap(107,*).
cap(108,*).
cap(109,*).
invest(1,0).
invest(2,0).
invest(3,0).
invest(4,0).
invest(5,0).
invest(6,0).
invest(7,0).
invest(8,0).
invest(9,0).
invest(10,0).
invest(11,0).
invest(12,0).
invest(13,0).
invest(14,0).
invest(15,0).
invest(16,0).
invest(17,0).
invest(18,0).
invest(19,0).
invest(20,0).
invest(21,0).
invest(22,0).
invest(23,0).
invest(24,0).
invest(25,0).
invest(26,0).
invest(27,0).
invest(28,0).
invest(29,0).
invest(30,0).
invest(31,0).
invest(32,0).
invest(33,0).
invest(34,0).
invest(35,0).
invest(36,0).
invest(37,0).
invest(38,0).
invest(39,0).
invest(40,0).
invest(41,0).
invest(42,0).
invest(43,0).
invest(44,0).
invest(45,0).
invest(46,0).
invest(47,0).
invest(48,0).
invest(49,0).
invest(50,0).
invest(51,0).
invest(52,0).
invest(53,0).
invest(54,0).
budget(10000).
rbudget(10000).
```

Appendix D. *Formula Version 3.0 User's Manual*

This manual explains how to use Prolog program, FORMULA Version 3.0. It consists of the Version 2.0 user's manual written by Gaught (12:243-251), updated with version 3.0 enhancements. This program consists of two files:

* FORMULA3.ARI - Contains the main computer program.

* WINDOWS3.ARI - Contains window dialog boxes.

D.1 Required Equipment

Currently, FORMULA requires ARITY/PROLOG interpreter program to run. The interpreter and FORMULA can be installed on an IBM-XT/AT compatible microcomputer. The computer with at least 512 kilobytes of random access memory and 20 megabyte hard disk is desired.

D.2 Running the FORMULA

Assuming both ARITY/PROLOG interpreter and FORMULA are installed on your computer, start the interpreter by typing 'API'. When '?- prompt appears, consult your program by typing 'consult('formula3.ari'). ' After the program has been consulted correctly, type in 'go.' to start the program FORMULA. The program proceeds through the following steps:

1) First, it displays an introductory screen displaying what this programcan do. (Just hit any key to go on.)

2) Next, it asks for the name of input data file which contains the description of network to be analyzed (Type in the exact name of input file and hit return.)

3) After the input file name has been typed in, the program displays a menu window from which you can choose to generate the specific output (Select number 1, 2, 3, 4, 5, 6, 7,8, or 9):

1. Find all paths and calculate path reliabilities.
2. Generate the Maximum Flow Formulation.
3. Generate the Lower Bound Formulation
4. Generate the Upper Bound Formulation
5. Generate the Investment Strategy Model 1

6. Generate the Investment Strategy Model 2.

7. Generate the Investment Strategy Model 3.

8. Generate Reliability Files.

9. Exit.

4) It then asks where to send the output. If you just want to screen the output, choose 1; otherwise, choose 2 to save the output in a file. The output filename is automatically generated by the program. When the user requests the output to be sent to a file, the output of paths and reliabilities is sent to 'output1.lp', maximum flow to 'output2.lp', lower bound to 'output3.lp', upper bound to 'output4.lp', investment strategy model 1 to 'output5.lp', investment strategy model 2 to 'output6.lp', and investment strategy model 3 to 'output7.nlp'. The reliability files generated are 1) path.f, 2) prob.f, 3) cap.f, and 4) net.top. (Select 1 or 2.)

5) After the output has been generated, the program asks if you want to run the program again. If you are interested in getting other output, type in 'y'; otherwise type in 'n' which exits the program. (Type in y or n.)

D.3 Input

In preparing the input data, you have to follow the following procedures:

1) If the network contains stochastic and/or capacitated nodes, convert the nodes to a dummy arc joined by two nodes. The dummy arc represents the stochastic and/or capacitated node.

2) Introduce an artificial single source and single sink. The source and sink must be named s and t, respectively. Connect all source nodes in the network to s, and all sink nodes to t.

3) Draw a revised network, and assign arc numbers. It has been found very helpful if you number the dummy arcs representing the node with the same node number. Then number the remaining arcs starting with one number higher than the number of nodes in the network. Thus, for example, if you have 20 nodes in the original network, number the remaining arcs starting with 21.

4) Using a text editor (Arity/Prolog comes with its own editor), prepare the input data by typing in the description of revised network that is to be analyzed. The input consists of eight data sets: description of an arc relationship with respect to one another, the survival probability of each arc, the capacity of each arc, the cost of improving each arc by one unit of capacity, the predetermined amount of capacity increase in each component, total budget available for capacity

investment, the cost of improving the reliability of each arc by 0.1, and the total budget available for reliability investment. These input are described as *facts* in Prolog term as follows:

Input	Description
`arc(Arc1,Arc2).`	Arc1 is the parent of Arc2 (Arc1 precedes Arc2).
`prob(A,Pb).`	The survival probability of arc A is Pb.
`cap(A,Cp).`	The capacity of arc A is Cp. The unlimited (infinite) capacity is denoted by '*'.
`cost(A,Cs).`	The cost of increasing one unit of capacity in arc A is Cs.
`invest(A,Am).`	The predetermined amount of capacity increase for arc A is Am.
`budget(Bc).`	Total budget available for investment in capacity is Bc.
`rcost(A,Rs).`	The cost of increasing one unit of reliability in arc A is Rs.
`rbudget(Br).`	Total budget available for investment in reliability is Br.

When defining arc relationships, you must define the relationship of *s* and *t* with respect to other arcs So '`arc(s,2).`' defines that the arc 2 is incidence to node `s`, that is, the arc 2 leaves the node `s`. In a likely manner, '`arc(22,t).`' defines that the arc 22 arrives at node `t`. In addition to specifying the probability of each arc, the probability of `s` and `t` must be defined as 1 Thus, in the probability description, make sure you include '`prob(s,1)`' and '`prob(t,1)`'.

5) Depending on your need, some of the input data may be omitted. Refer to the table below to see exactly which data set are required to get the desired output. For example, if you are only interested in finding paths and their reliabilities, all you need is arc relationships and survival probabilities.

For this output:	You must have the following data:
Paths and Reliabilities	arc(X,Y).
	prob(X,Y).
Maximum Flow,	arc(X,Y).
Lower Bound, or	prob(X,Y).
Upper Bound	cap(X,Y).
Improvement Strategy 1	arc(X,Y).
	prob(X,Y).
	cap(X,Y).
	cost(X,Y).
	budget(X,Y).
Improvement Strategy 2	arc(X,Y).
	prob(X,Y).
	cap(X,Y).
	cost(X,Y).
	budget(X,Y).
	invest(X,Y).
Improvement Strategy 3	arc(X,Y).
	prob(X,Y).
	cap(X,Y).
	rcost(X,Y).
	rbudget(X,Y).

6) When using Arity/Prolog, it is customary to name the file with '.ari' extension. So name your input file with '.ari' extension.

D.4 Example

To illustrate how to prepare the input data, consider a network shown in Figure 18. This network contains 4 nodes and 3 arcs. It has multiple sources, 1 and 2, and a single sink, 4.

The cost of increasing one unit of capacity is as follows: node 3 = 10, $arc_{1,3} = 20$, $arc_{2,3} = 30$, and $arc_{3,4} = 40$. The predetermined amount of capacity increase is 5 units for all components, and the total budget available for investment is \$1000. The cost of increasing reliability 0.1 is the same as increasing capacity by 1, and the reliability budget is the same as the capacity budget. The revised network is shown in Figure 19.

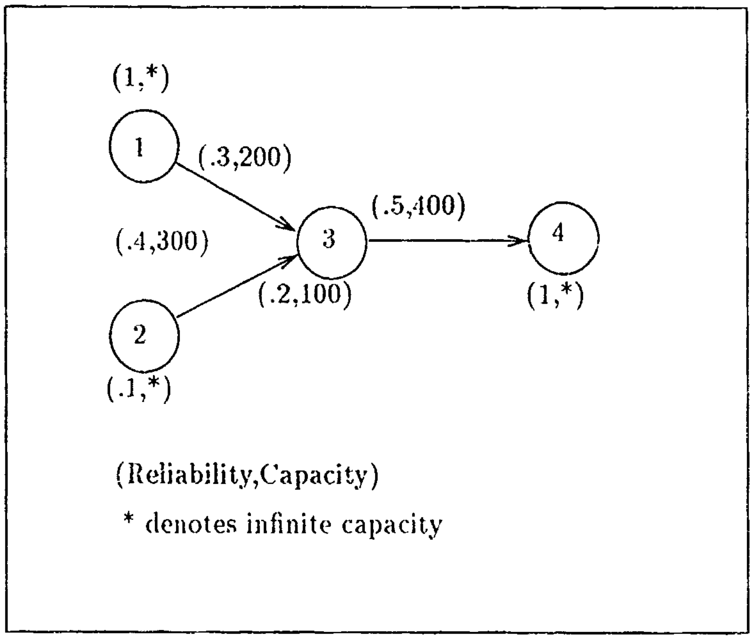


Figure 19. Sample Network

Since node 1 and 4 are not stochastic and not capacitated, they do not need to be represented as arcs. A dummy arc representing a stochastic or capacitated node is assigned the same number as its node number In numbering arcs, the number 1 and 4 are not used, since 1 and 4 are not represented as arcs. The remaining arcs are numbered starting with 5. Now, referring to the revised network shown in Figure 19, the input data can be prepared as follows:

```
% Arc Relationship
   arc(s,5).
   arc(s,6).
   arc(2,8).
   arc(3,9).
   arc(5,7).
   arc(6,2).
   arc(7,3).
   arc(8,3).
   arc(9,10).
   arc(10,t).

% Survival Probabilities
   prob(s,1)
   prob(2,0.1).
```

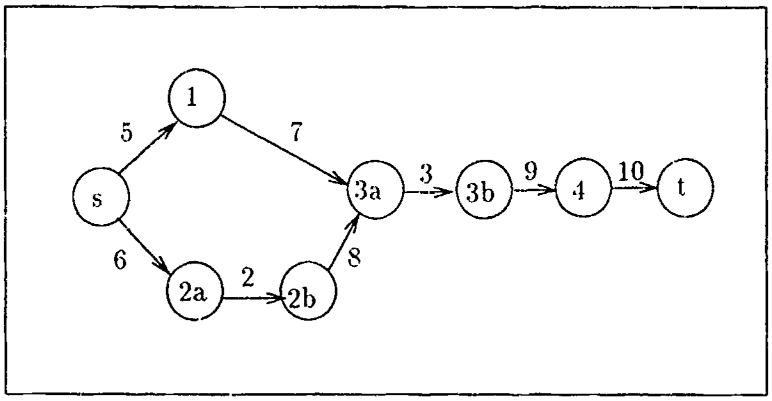


Figure 20. Revised Network

```
    prob(3,0.2).
    prob(5,1).
    prob(6,1).
    prob(7,0.3).
    prob(8,0.4).
    prob(9,0.5).
    prob(10,1).
    prob(t,1).

% Capacity
    cap(3,100).
    cap(5,*).
    cap(6,*).
    cap(7,200).
    cap(8,300).
    cap(9,400).
    cap(10,*).

% Cost of Increasing One unit of Capacity
    cost(3,10).
    cost(7,20).
    cost(8,30).
    cost(9,40).

% Predetermined Amount of Capacity Increase
    invest(_,5).

% Capacity Budget Available
    budget(1000).

% Cost of Increasing Reliability by 0.1
    rcost(3,10).
```

```
   rcost(7,20).
   rcost(8,30).
   rcost(9,40).

% Reliability Budget Available
   rbudget(1000).

% --- end--- %
```

Any line that starts with a % sign is a comment line, and it is ignored by the interpreter. The underscore (_) in '`invest(_,5).`' denotes *all components*. Thus, '`invest(_,5).`' denotes the predetermined amount of capacity increase for all components is 5 units.

D.5 Output

An example of output1.lp, containing paths and path reliabilities, are shown below:

```
****************************************************
* Following is a list of all patns from "s" to "t" *
* of the network described in the input data file. *
****************************************************

  Path1: s 5 7 3 9 10 t
  Reliability:  0.003

  Path2: s 6 2 8 3 9 10 t
  Reliability:  0.004

* ----- end ----- *
```

The rest of the outputs, containing mathematical programming models, are in the same format as the input format of LP/MIP-83. Each output consists of . Title. Objective Maximize. and Constraints section. An example of output2.lp containing the maximum flow formulation, is shown below:

```
..Title

Maximum Flow Formulation 2

. Objective Maximize

f1 + f2
```

```
..Constraints

Arc 3: f1 + f2 <= 100

Arc 7: f1 <= 200

Arc 8: f2 <= 300

Arc 9: f1 + f2 <= 400

* ----- end ----- *
```

D.6 LP/MIP-83 Commands

All models, except Investment Strategy Models 2 and 3, can be solved using either LP83 or MIP83 The Investment Strategy Model 2 can only be solved by MIP83, because it requires integer solutions. The following are some commands to run LP/MIP-83:

a) c :> lp83 a:output2

Find all solutions to the linear programming model stored in 'output2.lp' in 'a' drive, and display the solutions on the screen.

b) c :> lp83 a:output2 output a:list

Same as a) above, except send the solutions to the output file named 'list.prn', in 'a' drive. The extension, '.prn' is automatically added.

c) c :> lp83 a:output2 output a:list alternate 1

Same as b) above, except find only one solution.

d) c :> mip83 a:output6

Find all solutions to the mixed integer programming model stored in 'output6.lp' in 'a' drive, and display the solutions on the screen.

D.7 GINO Commands

Investment Strategy Model 3 must be solved using *GINO*. *GINO* can be run on many different types of computer systems. The following example is for a computer running the MS-DOS operating system:

c :> GINO	(this command starts the *GINO* program)
:retr output7.nlp	(this command loads the file created by *Formula*)
:go	(this command tells *GINO* to solve the model)
:quit	(this command exits *GINO*)

The user should consult a *GINO* user's manual for further details.

D.8 Helpful Comments

1) The program, FORMULA has been tested and successfully generated formulations for the network containing 70 nodes, 112 arcs, and 2198 paths. Since the capacity of LP/MIP-83 is approximately 1200 variables (paths), any problem bigger than the capacity of LP/MIP-83, must be solved using other mathematical programming packages, such as MINOS or SAS. Of course preparing an input file for these packages will be different from that of LP/MIP-83.

2) If you want to stop the execution (or if the computer is hung up), press 'control' and 'break' simultaneously. When '?-' appears, type 'clear_windows.' and/or 'exit_popup.'. It will get you to the main window of Arity/Prolog. Before starting the program again, you must erase the datafile by typing 'Restore.' Otherwise, you will get erroneous output

3) Any questions about the Arity/Prolog interpreter or general questions about the Arity/Prolog, refer to (4).

Appendix E. *Formula Version 3.0 Source Code*

```
/*============================================================*/
/*                                                            */
/*                  FORMULA   Version 3.0                     */
/*                                                            */
/*============================================================*/
/*                                                            */
/* This program does the following eight tasks:               */
/*                                                            */
/*  1) Finds all paths in the network from source (s) to      */
/*     sink (t) and calculates all path reliabilities.        */
/*  2) Generates the formulation of the Maximum Flow through  */
/*     the network.                                           */
/*  3) Generates the formulation of the Lower Bound of the    */
/*     Expected Maximum Flow.                                 */
/*  4) Generates the formulation of the Upper Bound of the    */
/*     Expected Maximum Flow.                                 */
/*  5) Generates the formulation of the investment strategy   */
/*     model 1.                                               */
/*  6) Generates the formulation of the investment strategy   */
/*     model 2.                                               */
/*  7) Generates the formulation of the investment strategy   */
/*     model 3.                                               */
/*  8) Generates files used for network reliabity formulation */
/*                                                            */
/* The six mathematical programming models (2 to 7) are       */
/* developed based on arc-path incidence matrix built from    */
/* the description of the network in the input file.          */
/* The network described in the input file must contain a     */
/* single source node, named 's', and sink node, named 't'.   */
/* All capacitated or stochastic nodes must be represented    */
/* as a dummy arc with two nodes. Refer to FORMULA user's     */
/* manual for details on how to prepare the input file.       */

/*                                                            */
/* The formulations, 2 thru 6, generated from this program    */
/* are in the same format as the input data file of LP/MIP 83 */
/* mathematical programming package.  The formulation 7 is in */
/* the same format as the input data file of GINO mathematical*/
/* programming package.  Thus the ouputs 2 thru 6 can be used */
/* as an input to LP/MIP 83 and the output 7 can be used as   */
/* an input to GINO for further analysis.                     */
/*                                                            */
/*------------------------------------------------------------*/
/*                                                            */
/*    DATE: 1 October, 1991                                   */
/*    FILENAME: FORMULA3.ARI                                 */
/*                                                           */
/* This program was written in Prolog language using the     */
/* Arity/Prolog Version 5.0.                                 */

/*                                                           */
/*===========================================================*/

/* Start the program by typing 'go.' */

 go :-
     fileerrors(_,off),      % Turn off system file error message.
     windows,                % Call windows to display the
                             % introduction screen.
     open_input_datafile,    % Get the name of input file and
                             % consult it.
     start_program.
 go.

 start_program :-
     get_selection_number(Selection),
                               % Ask user what need to be done.
   ( Selection = 9,            % If '9'is selected, exit.
     clear_windows,!
   ;                           % Otherwise,
     nl,                       % Where should output be sent ?
     get_where_to_send_output(Where),
     execute_request(Selection,Where)   % Execute the request.
   ).


 execute_request(Selection,Where) :-
    ( Selection = 1,
      search_paths(Where),!
    ;
      Selection >= 2,
      Selection =< 4,
      performance_formulations(Selection,Where)
    ;
      Selection >= 5,
      Selection =< 6,
      lp_investment_formulations(Selection,Where)
    ;
      Selection = 7,
      nlp_investment_formulation(Selection,Where)
    ;
      rel_file_formulation(Selection,Where)
    ),
    get_run_again_reply(Reply),      % Run the program again ?
    cls,                             % Clear screen.
    ( Where = 1,                     % If the output was sent to
     true, !                          % the screen, do nothing.
   ;                                  % Otherwise,
     exit_popup                       % delete popup window 'done'.
   ),
     want_more(Reply).

 want_more(Reply) :-
   ( Reply = 121,               % Reply is 'y' (ASCII 121),
     removeallh(matrix),        % delete hash table 'matrix', and
     start_program,!            % run program again.
   ;
    clear_windows,              % Otherwise, exit.
     nl
   ),!.


/*************************************************************/
/* 'search_path' initiates search to find all paths in the   */
/*  network and calculates path reliabilities.               */
/*************************************************************/

 search_paths(Where) :-
   ctr_set(1,1),         % Initialize counter one to 1
                         % to keep track of path number.
  ( Where = 1,           % When Where = 1, output is displayed
                         % on the monitor screen.
   continue_find_paths

  ;                      % 'Executing' message is displayed on
                         % the screen while the output is being
   executing_message,    % sent to the 'output1.lp' file.

   stdout('output1.lp',continue_find_paths),

   exit_popup,           % Delete popup window 'executing'.
   done_message(1)       % Display 'done' message.
  ).


 continue_find_paths :-
   nl,

write('*******************************************************'),
   nl,
   write('*  Following is a list of all paths from "s" to "t"
*'),
   nl,
   write('*  of the network described in the input data file.
*'),
   nl,
write('***********************************************************'),
   nl,nl,nl,
   find_paths(s,t),
   nl,nl,
   write('* ------ end ------ *'),
   nl,nl,nl,nl.

/*-----------------------------------------------------------*/

 find_paths(Start,Goal) :-          % Find all paths using
                                    % depth-first
                                    % search method.
    depth_first([Start],Goal,Path),
    find_reliability(Path,Rel),        % Calculates the reliability
                                       % of  a path.
    ctr_inc(1,Pnbr),                % Get current path number and
                                    % increment counter one by 1.
    display_outputs(Path,Pnbr,Rel),
    fail.                           % Go on to find next path until
                                    % there isn't any to be searched.

 find_paths(_,_).


 depth_first(Path,Goal,Path) :-        % Path is found if it
                                       % satisfies
    satisfies(Path,Goal).              % Goal.

 depth_first([X|Rest],Goal,Path) :-
    arc(X,Y),                       % Get next arc.
    not member(Y,[X|Rest]),         % Prevents cycles.
    depth_first([Y,X|Rest],Goal,Path).  % Recursive call.

 satisfies([Goal|_],Goal).          % A path is found if the head of
                                    % a list describing Path matches
                                    % with Goal (t).

 member(X,[X|Tail]).
 member(X,[Head|Tail]) :-
    member(X,Tail).


 find_reliability([],1) :- !.       % Reliability of an empty list is
                                    %  1.
 find_reliability([Arc|Rest],Rel) :-
    find_reliability(Rest,RelRest),    % Calculate Rel recursively.
    prob(Arc,Pb),                      % Get survival probability of
                                       % Arc.
    Rel is Pb * RelRest.
```

```
/*------------------------------------------------------------*/

 display_outputs(Path,Pnbr,Rel) :-
    print_path(Path,Pnbr),            % Print path and
    print_reliability(Rel).           % reliability.


 print_path(Path,Pnbr) :-
    write(' Path '),
    write(Pnbr),
    write(': '),
    write_reverse(Path).


 print_reliability(Rel) :-
    nl,
    write(' Reliability: '),
    write(Rel), nl, nl.


 write_reverse([]) :- !.           % Prints path from 's' to 't'.
 write_reverse([Arc|Rest]) :-
    write_reverse(Rest),
    write(Arc), write(' ').


/********* ***************************************************/
/* 'performance_formulations' generates the formulations of */
/*  maximum flow, lower bound of expected maximum flow,     */
/* and upper bound of expected  maximum flow.               */
/************************************************************/


 performance_formulations(Selection,Where) :-

   ( Where = 1,                    % Display output on the screen.
     find_formulations(Selection)
   ;
     executing_message,              % Display output sending
                                     % message.
     ( Selection = 2,
       stdout('output2.lp',find_formulations(Selection))
     ;
       Selection = 3,
       stdout('output3.lp',find_formulations(Selection))
     ;
       stdout('output4.lp',find_formulations(Selection))
     ),
     exit_popup,                     % Delete popup window
                                     % 'executing'.
     done_message(Selection)         % Display 'done' message.
```

```
  ).

 find_formulations(Selection) :-
     title(Selection),
     objective(Selection),
     constraints(Selection),
     nl,nl,
     write('* ------ end ------ *'),
     nl,nl,nl,nl.


/*----------------------------------------------------------*/

 title(Selection) :-                % Print title of the
                                    % formulation.
    write(' ..Title'),
    nl,nl,
    ( Selection = 2,
      write(' Maximum Flow Formulation')
    ;
      Selection = 3,
      write(' Lower Bound Formulation')
    ;
      write(' Upper Bound Formulation')
    ).
/*----------------------------------------------------------*/

 objective(Selection) :-            % Get objective function
    nl,nl,
    write(' ..Objective Maximize'),
    nl,nl,
    ctr_set(1,1),    % Counter one generates path number.
    ctr_set(3,1),    % Initialize counter three to 1
                         % to keep track of how many terms
                         % are printed in a line in the objective
                         % function.
    find_objective(s,t,Selection).


 find_objective(Start,Goal,Selection) :-
    depth_first([Start],Goal,Path),
    find_reliability(Path,Rel),
    ctr_inc(1,Pnbr),                % Get current path number and
                                    % increment the counter one by
                                    % one.
    print_objective(Pnbr,Rel,Selection),
    make_arc_path_matrix(Path,Pnbr),  % Make arc-path incidence
                                      % matrix.
    fail.
 find_objective(_,_,_).
```

```
print_objective(Pnbr,Rel,Selection) :-
   tab(1),
   ( Pnbr = 1
   ;
     write('+ ')
   ),
   ( ctr_inc(3,Value),    % Get current variable number and
                          % increment the counter three by 1.
     Value > 4,
     Mod_Value is Value mod 4,   % If the remainder of Value
     Mod_Value = 1,              % divided by 4 is 1, then skip
     nl,                         % to next line.
     write(' ')
   ;
     true
   ),
   ( Selection = 3,             % If finding lower bound (Sel =
                                % 3),
     write(Rel)                 % print reliability.
   ;
     true                       % Otherwise, do nothing.
   ),
   write(' f'),                 % Print path flow variable.
   write(Pnbr), !.

/*-----------------------------------------------------------*/

make_arc_path_matrix([],_).
make_arc_path_matrix(Arc_List,Pnbr) :-   % Seperate the elements,
                                         % arcs, in the path and
                                         % store each arc with
                                         % associated path number
                                         % to form arc-path
                                         % incidence
   get_arc(Arc_List,Arc,Rem_List),       % matrix.
   cap(Arc,Capacity),
   ( number(Capacity),
     recordh(matrix,Arc,arc_path_matrix(Arc,Pnbr))
   ;
     true
   ),
   make_arc_path_matrix(Rem_List,Pnbr), !.


get_arc([Head|[H|Rest]],Head,Rem_List) :-
   Head \= 's',                    % Ignore 's' and 't'
   Head \= 't',
   ( Rest = [],
     Rem_List = []
   ;
```

```
    Rem_List = [H|Rest]
   ), !.

get_arc([Head|Rest],Arc,Rem_List) :-
   get_arc(Rest,Arc,Rem_List).       % Get one arc at a time
                                     % that is in the path.


/*-------------------------------------------------------*/

constraints(Selection) :-            % Get constraints
                                     % function.


    nl, nl,
    write(' ..Constraints'),
    asserta(init_string($$)),        % Initialize to empty
                                     % string.
    find_all_arc_lists(Arc_List),
    sort(Arc_List,Sor),              % Sort Arc_List in ascending
                                     % order.
    make_arc_array(Sor),             % Make sorted arc_list into
                                     % arc array.
    generate_constraints(Selection).

find_all_arc_lists(_) :-             % Find all arcs that are in
                                     % the
                                     % arc-path incidence matrix.
    retrieveh(matrix,_,arc_path_matrix(AN,_)),
    int_text(AN, Arc_String),
    [!   concat([$0000$, Arc_String, $,$], New_Arc_String),
       retract(init_string(Init)),
       ( string_search(New_Arc_String,Init,_), % Do not include
                                               % the
         asserta(init_string(Init))            % duplicate arc
                                               % string.
       ;
         concat(New_Arc_String, Init, New_String), % Append the
                                                   % new
         asserta(init_string(New_String))          % string.
       )
    !],
    fail.


find_all_arc_lists(Final) :-
    retract(init_string(Main_String)),
    string_length(Main_String, Length),
    dec(Length,Pos),
    substring(Main_String,0,Pos,New_String),
    concat([$[$, New_String, $]$], Output_String),
    string_term(Output_String, Final), !.     % Change string into
```

```
                                                % a list.


/*--------------------------------------------------------------*/

 make_arc_array([]).                 % Seperate the arc_list and
 make_arc_array([Anbr|Rest]) :-      % put it into an array format.
     assertz(arc_array(Anbr)),
     make_arc_array(Rest).


/*--------------------------------------------------------------*/

 generate_constraints(Selection) :-
     retract(arc_array(Anbr)),
     [! cap(Anbr,Capacity),
        generate_constraint_inequality(Anbr),
        write(' '),
        write('<= '),
        write(' '),
        ( Selection = 4,
          prob(Anbr,Pb),
          Expected_Cap is Capacity * Pb,
          write(Expected_Cap)
        ;
          write(Capacity)
        )
     !],
     fail.

 generate_constraints(_).

 generate_constraint_inequality(Anbr) :-
     nl, nl,
     write(' Arc '),
     write(Anbr),
     write(': '),
     ctr_set(10,0),        % First time flag; this is used to
                               % control when to print '+' in
                               % the constraint equation.
     ctr_set(4,1),         % Initialize counter four to 1;
                               % this counter keeps track of
                               % how many terms are printed in
                               % the constraint equation.
     output_paths_containing_Anbr(Anbr).


 output_paths_containing_Anbr(Anbr) :-
     removeh(matrix,Anbr,arc_path_matrix(Anbr,Pnbr)),
     [!  ( ctr_inc(10,Flag),
           Flag = 0              % If first term, don't print '+'.
        ;
          write(' + ')
        ),
        ( ctr_inc(4,Value),     % Get current term number and
                                % increment the counter four by 1
          Value > 7,
          Mod_Value is Value mod 7,
          Mod_Value = 1,
          nl, write('         ')
        ;
          true
        ),
        write('f'),
        write(Pnbr)
    !],
    fail.

output_paths_containing_Anbr(_).

/***************************************************************/
/* 'lp_investment_formulations' generates formulations of     */
/*  investment strategy model 1 and 2 to improve the          */
/*  lower bound.                                               */
/***************************************************************/


lp_investment_formulations(Selection,Where) :-
    ( Where = 1,
      get_investment_model(Selection)
    ;
      executing_message,
      ( Selection = 5,
        stdout('output5.lp',get_investment_model(Selection))
      ;
        stdout('output6.lp',get_investment_model(Selection))
      ),
      exit_popup,
      done_message(Selection)
    ).

get_investment_model(Selection) :-
    invest_model_title(Selection),
    invest_model_objective(Selection),
    invest_model_constraints(Selection),
    nl, nl,
    write('* ------ end ------ *'),
    nl,nl,nl,nl.

invest_model_title(Selection) :-
   write(' ..Title'),
   nl,nl,
   ( Selection = 5,
     write(' Investment Strategy Model 1')
   ;
     write(' Investment Strategy Model 2')
   ).

invest_model_objective(Selection) :-
    nl, nl,
    write(' ..Objective Maximize'),
    nl, nl,
    ctr_set(1,1),                        % path number
    ctr_set(3,1),                        % no. of terms in a line
    find_invest_model_objective(s,t,Selection).

find_invest_model_objective(Start,Goal,_) :-
    depth_first([Start],Goal,Path),
    find_reliability(Path,Rel),
    ctr_inc(1,Pnbr),
    output_invest_model_objective_variables(Pnbr,Rel),
    make_arc_path_matrix(Path,Pnbr),
    fail.

find_invest_model_objective(_,_,Selection) :-
    asserta(init_invest_string($$)),
    get_investment_variables(Invest_Vars),
    sort(Invest_Vars,Sorted_Vars),
    make_cap_array(Sorted_Vars),
    nl,
    ctr_set(2,1),
    ctr_set(10,0),                           % First time flag.
    ( Selection = 5
    ;
      write(' [ ')
    ),
    output_investment_variables(Selection),
    ( Selection = 5
    ;
      write(' ] ')
    ).

find_invest_model_objective(_,_,_).

make_cap_array([]).
make_cap_array([Anbr|Rest]) :-
    assertz(cap_array(Anbr)),
    make_cap_array(Rest).


output_invest_model_objective_variables(Pnbr,Rel) :-
    write(' '),
    write(Rel),
    write(' f'),
    write(Pnbr),
    write(' +'),
    ( ctr_inc(3,Value),
      Mod_Value is Value mod 4,
      Mod_Value = 0,
      nl
    ;
      true
    ), !.


get_investment_variables(_) :-
    cap(AN,Capacity),
    [! ( not number(Capacity)
       ;
         int_text(AN, Arc_String),
         concat([$0000$, Arc_String, $,$], New_Arc_String),
         retract(init_invest_string(Init)),
         concat(New_Arc_String, Init, New_String),
         asserta(init_invest_string(New_String))
    )
    !],
    fail.


get_investment_variables(Final) :-
    retract(init_invest_string(Main_String)),
    string_length(Main_String, Length),
    dec(Length,Pos),
    substring(Main_String,0,Pos,New_String),
    concat([$[$, New_String, $]$], Output_String),
    string_term(Output_String, Final), !.


output_investment_variables(Selection) :-

   cap_array(Anbr),
   [!  ( ctr_inc(10,Flag),
         Flag = 0
       ;
         write(' +')
       ),
       ( ctr_inc(2,Value),
         Value > 7,
         Mod_Value is Value mod 7,
         Mod_Value = 1,
         nl
       ;
         true
       ),
        ( Selection = 5,
          write(' 0 d')
        ;
          write(' 0 g')
        )
    !],
    write(Anbr),
    fail.

 output_investment_variables(_).

/*-------------------------------------------------------*/


 invest_model_constraints(Selection) :-

     nl, nl,
     write(' ..Constraints'),
     asserta(init_string($$)),
     find_all_arc_lists(Arc_List),
     sort(Arc_List,Sor),
     make_arc_array(Sor),
     generate_arc_constraints(Selection),
     ctr_set(10,0),              % counter for budget term
     nl, nl,
     write(' Budget: '),
     generate_budget_constraint(Selection).


 generate_arc_constraints(Selection) :-
     retract(arc_array(Anbr)),
     [! cap(Anbr,Capacity),
        generate_constraint_inequality(Anbr),
        ( Selection = 5,
          write(' - d')
        ;
          write(' - '),
          invest(Anbr,Amount),
          write(Amount),
          write(' g')
        ),
        write(Anbr),
        write(' '),
        write('<= '),
        write(' '),
        write(Capacity)
     !],
     fail.

 generate_arc_constraints(_).
```

```
generate_budget_constraint(Selection) :-
   retract(cap_array(Anbr)),
   [! cost(Anbr,Unit_cost),
      ( Selection = 5,
        Cost is Unit_cost
      ;
        invest(Anbr,Amount),
        Cost is Unit_cost * Amount
      ),
      print_budget_terms(Anbr,Cost,Selection)
   !],
    fail.

generate_budget_constraint(_) :-
    ctr_set(5,1),
    budget(Budget),
    write(' '),
    write(' <= '),
    write(Budget).


print_budget_terms(Anbr,Cost,Selection) :-
   ( ctr_inc(10,Flag),
     Flag = 0
   ;
     write(' + ')
   ),
   ( ctr_inc(5,Value),
     Value > 5,
     Mod_Value is Value mod 5,
     Mod_Value = 1,
     nl, write('         ')
   ;
     true
   ),
   write(Cost),
   ( Selection = 5,
     write(' d')
   ;
     write(' g')
   ),
   write(Anbr).

/**************************************************************/
/* 'nlp_investment_formulation' generates formulation of     */
/* investment strategy model 3.                              */
/**************************************************************/

 nlp_investment_formulation(Selection,Where) :-
```

```
    ( Where = 1,
      begin_model_building
      ;
      executing_message,
      stdout('output7.nlp',begin_model_building),
      exit_popup,
      done_message(Selection)
    ).

 begin_model_building :-
    nl, nl, nl,
    write('MODEL: '), nl,
    invest3_objective,  nl,
    invest3_constraints,
    write('LEAVE'),
    nl,nl,nl.

/***************************************************************/
/*-- Objective function  --*/

 invest3_objective :-
    write('MAX= '),
    ctr_set(1,1),   % Counter one generates path number.
    ctr_set(3,1),   % Initialize counter three to 1
                       % to keep track of how many terms
                       % are printed in a line in the objective
                       % function.
    find_invest3_objective(s,t),
    write(' ;').


 find_invest3_objective(Start,Goal) :-
   depth_first([Start],Goal,Path),
   ctr_inc(1,PathNmbr),                  % Get current path number
                                         % and
                                  % increment the counter one by
                                  % one.
   print_invest3_objective(PathNmbr),
   make_arc_path_matrix(Path,PathNmbr),  % Make arc-path incidence
                                         % matrix.
   fail.

 find_invest3_objective(_,_).

/*---------------------------------------------------------*/

 print_invest3_objective(PathNmbr) :-
   ( PathNmbr = 1
   ;
     write(' + ')
   ),
    ( ctr_inc(3,Value),     % Get current variable number and
                            % increment the counter three by 1.
      Value > 5,
      Mod_Value is Value mod 5,   % If the remainder of Value
      Mod_Value = 1,              % divided by 4 is 1, then skip
      nl,                         % to next line.
      write('     ')
    ;
      true
    ),
    write('R'),
    write(PathNmbr),
    write(' * F'),                % Print path flow variable.
    write(PathNmbr), !.

/**************************************************************/
/* constraint functions */


 invest3_constraints :-              % Get constraints
                                     % function.
     asserta(init_string($$)),       % Initialize to empty
                                     % string.
    find_all_arc_lists(Arc_List),
    sort(Arc_List,Sor),              % Sort Arc_List in ascending
                                     % order.
    make_arc_array(Sor),             % Make sorted arc_list into
                                     % arc array.
    ctr_set(1,1),           % Counter 1 contains pathnumber.
    generate_rj_descriptions,
    generate_arc_prob_constraints,
    ctr_set(9,0),                  % Flag counter to control line
                                   % feed.
    generate_path_flow_constraints,
    ctr_set(10,0),               % Flag counter to control + sign.
    generate_rel_budget_constraints,
    write('END'), nl,
    generate_x_nonnegativity,
    ctr_set(1,1),                % Path number counter
    generate_f_nonnegativity.

/*------------------------------------------------------------*/

 generate_rj_descriptions :-
    depth_first([s],t,Path),
    [! ctr_inc(1,PathNmbr),
      write('R'),
      write(PathNmbr),
      write(' = '),
      sort(Path,SortedPath),
      ctr_set(7,1),     % Counter to control the number of terms.
      write_rhs_of_equality(SortedPath),
      write(';'),
      nl
   !],
   fail.

generate_rj_descriptions.

write_rhs_of_equality([]).

write_rhs_of_equality([Arc|Rest]) :-
    ( not number(Arc)
    ;
      ( ctr_inc(7,Value),
        Value > 3,
        Mod_Value is Value mod 3,
        Mod_Value = 1,
        nl, write('     ')
      ;
        true
      ),
      write('( '),
      prob(Arc,Probability),
      write(Probability),
      write(' + .1 * X'),
      write(Arc),
      write(' ) '),
      ( Rest == [s,t]
        ;
        write('* ')
      )
    ),
    write_rhs_of_equality(Rest).

/*-----------------------------------------------------------*/

generate_arc_prob_constraints :-
   prob(ArcNmbr,Probability),
   [' ( not number(ArcNmbr)
      ;
        write(Probability),
        write(' + .1 * X'),
        write(ArcNmbr),
        write(' < 1 ;'),
        nl
      )
  '],
  fail.

generate_arc_prob_constraints.
```

```
/*------------------------------------------------------------*/

 generate_path_flow_constraints :-
     retract(arc_array(ArcNmbr)),
     [! cap(ArcNmbr,Capacity),
        generate_path_flow_constraint_inequality(ArcNmbr),
        write(' '),
        write('< '),
        write(Capacity),
        write(' ;')
     !],
     fail.

 generate_path_flow_constraints :- nl.

 generate_path_flow_constraint_inequality(ArcNmbr) :-
     ( ctr_inc(9,Flag),
       Flag = 0
     ;
       nl
     ),
     ctr_set(10,0),        % First time flag; this is used to
                                % control when to print '+' in
                                % the constraint equation.
     ctr_set(4,1),         % Initialize counter four to 1;
                                % this counter keeps track of
                                % how many terms are printed in
                                % the constraint equation.
     output_paths_containing_ArcNmbr(ArcNmbr).


 output_paths_containing_ArcNmbr(ArcNmbr) :-
     removeh(matrix,ArcNmbr,arc_path_matrix(ArcNmbr,PathNmbr)),
     [!  ( ctr_inc(10,Flag),
           Flag = 0              % If first term, don't print '+'.
         ;
           write(' + ')
         ),
         ( ctr_inc(4,Value),    % Get current term number and
                                 % increment the counter four by 1
           Value > 9,
           Mod_Value is Value mod 9,
           Mod_Value = 1,
           nl
         ;
           true
         ),
         write('F'),
         write(PathNmbr)
     !],
     fail.
```

```
output_paths_containing_ArcNmbr(_).

/*-------------------------------------------------------------*/

generate_rel_budget_constraints :-       % probably can revise
                                         % this: I dont think
                                         % there
                                         % is any need for using
                                         % prob-array from
                                         % get-invest3-variables
   prob(ArcNmbr,_),
   [! ( not number(ArcNmbr)
      ;
       rcost(ArcNmbr,Unit_cost),
       print_rel_budget_terms(ArcNmbr,Unit_cost)
      )
   !],
   fail.

generate_rel_budget_constraints :-
    ctr_set(5,1),
    rbudget(Budget),
    write(' '),
    write(' < '),
    write(Budget),
    write(' ;'),
    nl.


print_rel_budget_terms(ArcNmbr,Cost) :-
   ( ctr_inc(10,Flag),
     Flag = 0
   ;
     write(' + ')
   ),
   ( ctr_inc(5,Value),
     Value > 5,
     Mod_Value is Value mod 5,
     Mod_Value = 1,
     nl
   ;
     true
   ),
   write(Cost),
   write(' * X'),
   write(ArcNmbr).

/*-------------------------------------------------------------*/

generate_x_nonnegativity :-
```

```
   prob(ArcNmbr,Probability),
   [! ( not number(ArcNmbr)
      ;
         write('SLB X'),
        write(ArcNmbr),
        write(' 0 '),
        nl
      )
   !],
   fail.

generate_x_nonnegativity.

/*----------------------------------------------------------*/

 generate_f_nonnegativity :-
    depth_first([s],t,_),
    [! ctr_inc(1,PathNmbr),
       write('SLB F'),
       write(PathNmbr),
       write(' 0 '),
       nl
    !],
    fail.

 generate_f_nonnegativity.

/***********************************************************/
/* 'rel_file_formulation' generates 4 files of inputs used */
/* in formulating Reliability models.                      */
/***********************************************************/

rel_file_formulation(Selection,Where) :-
    ( Where = 1,
      file_building
      ;
      executing_message,
      stdout('path.f', file_building),
      stdout('prob.f', get_arc_prob),
      stdout('cap.f', get_arc_cap),
      stdout('net.top',net_top),
      exit_popup,
      done_message(Selection)
    ).

file_building:-
    get_arc_factors.

/*----------------------------------------------------------*/

get_arc_factors :-
```

```
   asserta(init_string($$)),
   find_all_arc_lists(Arc_List),
   sort(Arc_List,Sor),
   make_arc_array(Sor),
   ctr_set(1,1),
   get_ord_paths.



/*--------------------------------------------------------*/

 get_ord_paths :-
    depth_first([s],t,Path),
    [! ctr_inc(1,PathNmbr),
       write(PathNmbr),
       write(' '),
       sort(Path,SortedPath),
       write_rhs_of_path(SortedPath),
       nl
    !],
    fail.

 get_ord_paths.

 write_rhs_of_path([]).

 write_rhs_of_path([Arc|Rest]) :-
     ( not number(Arc)
     ;
       prob(Arc,Probability),
       write(' '),
       write(Arc),
       write(''),
       ( Rest == [s,t]
         ;

         write('')
       )
      ),

 write_rhs_of_path(Rest).

/*--------------------------------------------------------*/

get_arc_prob :-
 prob(ArcNmbr,Probability),
    [! (not number(ArcNmbr)
        ;
          write(ArcNmbr),
          write(' '),
          write(Probability),
          nl
        )
      !],
      fail.

get_arc_prob.

/*-----------------------------------------------------------*/

get_arc_cap :-
   cap(ArcNmbr,Capacity),
   [! ( not number(Capacity)
   ;
   write(ArcNmbr),
   write(' '),
   write(Capacity),
   nl
   )
  !],
  fail.

get_arc_cap.

/*-----------------------------------------------------------*/

get_arc_inv_cost :-
   cost(ArcNmbr,Cost),
   write(Cost),
   nl,
   fail.

get_arc_inv_cost.

/*-----------------------------------------------------------*/

net_top :-
arc(From,To),
[!( not number(From);not number(To)
   ;
write(From),
write(' '),
write(To),
nl
)
!],
fail.

net_top.

/*-----------------------------------------------------------*/
```

```
 :- reconsult('windows3.ari').

/* ----------------------- end ----------------------------- */
```

```
/***************************************************************/
/*                                                             */
/*                     WINDOWS3.ARI                            */
/*                                                             */
/*  This file contains windows to communicate with the user.   */
/*                                                             */
/***************************************************************/

/*-------------------------------------------------------------*/
/*  Introduction Screen                                        */
/*-------------------------------------------------------------*/

windows :-
  cls,
  define_window(program_title,'',(23,0),(23,79),(91,0)),
  define_window(intro,'',(0,0),(22,79),(26,0)),
  current_window(_,program_title),
  tmove(0,12),
  write(' FORMULA   Version 3.0      AFIT   October, 1991'),
  current_window(_,intro),
  define_intro_window.

 define_intro_window :-
  nl,nl,
  tab(17),
  write('*************************************'),
  nl,tab(17),
  write('*     F O R M U L A    V e r  3.0     *'),
  nl,tab(17),
  write('*************************************'),
  nl,nl,nl,
  tab(11),
  write('This program finds all paths in the network from '),
  nl,tab(11),
  write('source to sink and calculates all paths reliabilities.'),
  nl,tab(11),
  write('It also generates six mathematical programming '),
  nl,tab(11),
  write('models that will assist in analyzing the performance '),
  nl,tab(11),
  write('of the network and in determining the investment '),
  nl,tab(11),
  write('strategy to improve the performance of the network.'),
  nl,tab(11),
  write('These models are developed based on the arc-path '),
  nl,tab(11),
  write('incidence matrix built from the description of the '),
  nl,tab(11),
  write('network in the input file. '),
  nl,tab(11),
  write(' Finally, it will generate the required files used '),
  nl,tab(11),
  write('in formulating network reliability analysis. '),
  nl,nl,tab(11),
  write('PLEASE make sure the input file contains correct'),
  nl,tab(11),
  write('description of the network to be analyzed.'),
  nl,nl,tab(23),
  write('Press any key to continue. '),
  get0(_),
  cls.


/*-------------------------------------------------------------*/
/* Asks for the input file name. If the file name is not found, */
/* the program prints the error message; otherwise, consults    */
/* the input file.                                              */
/*-------------------------------------------------------------*/

open_input_datafile :-
  create_popup(query1,(7,20),(14,60),(62,-62)),
  write(' Please type in your input file name.   '),
  tmove(3,2),
  write(' > '),
  read_line(0,File),
  (
    consult_file(File),
    exit_popup
  ;
    display_filename_error,
    exit_popup,
    open_input_datafile
  ).

consult_file(File) :-
   stdin(File,_),
   consult(File).

display_filename_error :-
  create_popup(error1,(16,20),(21,60),(79,-79)),
  write('     Error: File not found. '),
  put(7),
  nl, nl,
  write(' Type in any key to continue or '),
  nl,
  write(' press RETURN to exit. '),
  get0(Reply),
  ( Reply = 13,
    exit_popup,
    exit_popup,
    clear_windows
  ;
```

```
    exit_popup
  ).


/*------------------------------------------------------------*/
/* Ask user what to do.                                       */
/*------------------------------------------------------------*/

get_selection_number(Selection) :-
  create_popup(query2,(3,12),(19,68),(62,-62)),
  tmove(1,16),
  write('How may I help you?'),
  tmove(4,2),
  write('1. Find all paths and calculate path reliabilities.'),
  tmove(5,2),
  write('2. Generate the Maximum Flow Formulation.'),
  tmove(6,2),
  write('3. Generate the Lower Bound Formulation. '),
  tmove(7,2),
  write('4. Generate the Upper Bound Formulation. '),
  tmove(8,2),
  write('5. Generate the Investment Strategy Model 1. '),
  tmove(9,2),
  write('6. Generate the Investment Strategy Model 2. '),
  tmove(10,2),
  write('7. Generate the Investment Strategy Model 3. '),
  tmove(11,2),
  write('8. Generate Reliability Files. '),
  tmove(12,2),
  write('9. Exit'),
  tmove(13,18),
  write('Type in number >  '),
  get0(Choice),            % The selection chosen is in ASCII code,
  exit_popup,              % that is one is represented as 49, two
  Sel_Nbr is Choice - 48,  % represented as 50, etc. Thus, 48 is
  ( Sel_Nbr >= 1,          % substracted to make it back to regular
    Sel_Nbr =< 9,          % arabic number.
    Selection = Sel_Nbr
  ;
    put(7),
    get_selection_number(Selection)
  ).


/*------------------------------------------------------------*/
/* Asks user where to display the output.                     */
/*------------------------------------------------------------*/



get_where_to_send_output(Where) :-
```

```
  create_popup(query3,(5,20),(16,60),(62,-62)),
  tmove(1,8),
  write('Where do you want the'),
  tmove(2,8),
  write('output displayed ?'),
  tmove(4,11),
  write('1. Screen'),
  tmove(6,11),
  write('2. File'),
  tmove(8,8),
  write('Type in Number >  '),
  get0(Choice),
  exit_popup,
  Sel_Nbr is Choice - 48,
  ( Sel_Nbr >= 1,
    Sel_Nbr =< 2,
    Where = Sel_Nbr
  ;
    put(7),
    get_where_to_send_output(Where)
  ).


/*--------------------------------------------------------------*/
/* Asks user to run the program again with same input file.     */
/*--------------------------------------------------------------*/

get_run_again_reply(Reply):-
  create_popup(query4,(20,0),(22,79),(62,-62)),
  write(' Do you want to run the program again (y or n) ?  '),
  get0(User_Reply),
  exit_popup,
  (
    ( User_Reply = 121;      % y
      User_Reply = 110        % n
    ),
    Reply = User_Reply
  ;
    put(7),
    get_run_again_reply(Reply)
  ).


/*--------------------------------------------------------------*/
/* Prints 'executing' message while output is sent to an output */
/* file.                                                        */
/*--------------------------------------------------------------*/

executing_message :-
  create_popup('',(11,25),(14,50),(207,79)),
  write('     Executing ... '),
  nl,
  write('      Please Wait.').


/*-------------------------------------------------------------*/
/* Prints 'Done' message after output is sent to an output file. */
/*-------------------------------------------------------------*/

done_message(Output_File) :-
  create_popup('',(11,20),(14,57),(58,58)),
  write('              Done. '),
  nl,
  write('  Output was sent to "output'),
  write(Output_File),
  ( Output_File = 7,
    write('.nlp".')
  ;
    write('.lp".')
  ).


/*-------------------------------------------------------------*/
/* Clears all windows before exiting(quiting) the program.      */

/*-------------------------------------------------------------*/

clear_windows :-
  delete_window(program_title),
  delete_window(intro),
  abolish(arc/2),
  abolish(prob/2),
  abolish(cap/2),
  abolish(cost/2),
  abolish(budget/1),
  removeallh(matrix),
  current_window(_,main).


/*----------------------------- end---------------------------*/
```

Appendix F. *Reliability Programs and Files*

F.1 Dirprog - Turbo Pascal Version

```
{This is an MS-DOS Turbo Pascal version of the program described in
 the paper "Reliability of Directed Networks Using the Factoring
 Theorem" in the Dec. 1989 issue of the IEEE Transactions on
 Reliability.  The program was developed on a Macintosh II and
 ported to MS-DOS.  Because stack space is limited to 64k on MS-DOS
 computers, you may run into problems with stack overflow on very large
 networks. The program will, however, treat all the networks described
 in the aforementioned paper. There are a number of sample network files
 on this disk.  Check them out first.  The first line gives the source
 and sink, and each additional line describes an edge (by giving the
 vertices joined and the reliability).  The program creates an output
 file with the same name except that ".OUT" is appended.  Since the
 program uses this convention, use names for your input files that do
 not have an extension as part of their name. This program has been
 compiled and tested with Turbo Pascal 5.5.  A compiled version of the
 program is also included on this disk.}

{$M 65520,0,200000}
program DirNetworkerApp;
   uses Crt, DOS;

{ Programmers:  Lavon Page and Jo Perry


 Input:  A text file describing a directed graph with a source and
sink pair of vertices.  All vertices are represented by integers in
the range of 1..maxv.  All edges are represented by their endpoints
followed by their reliabilities. The first line of the file consists
of the source then sink. Each subsequent line in the file describes
an edge.  A maximum of "maxe" edges are allowed.

 Output:  A text file whose name is the catenation of the name of
the input file with the string 'Output'.  The first line identifies
the input file and contains an execution time and date stamp.  The
next lines echo the input.  The results of execution of the algorithm
come last.  Included are the reliability of the graph and execution
statistics: algorithm execution time, number of single edge to source
or sink reductions, number of reductions of a vertex with in-degree
and out-degree 1, number of times factoring is performed.}
```

```
const
   maxv = 50;          {Maximum number of vertices in the graph}
   maxe = 100;          {Maximum number of edges}

type
   degreeType = array [1..maxv] of integer; {List of vertex degrees}
   graphSet = set of 1..maxv;       {Set of vertices}
   edge = record                    {Edge in a graph}
      start,                        {Start vertex}
      stop : 1..maxv;               {Stop endvertex}
      pr : real                 {Probability of the edge}
      end; {edges}
   graph = record                   {Describes a graph}
      vert : graphSet;              {Set of graph vertices}
      source,                       {Source vertex}
      sink : integer;               {Sink vertex}
      inDegree,                     {In degree of each vertex}
      outDegree : degreeType;       {Out degree of each vertex}
      nb : array [1..maxv] of graphSet; {edge (i,j) puts j in nb[i]}
      numEdges : integer;           {Number of edges in the graph}
      maxVertex : integer;          {Largest numbered vertex in the graph}
      e : array [1..maxe] of edge  {Describes all edges in the graph}
      end; {Graph}

var
   filename : string;       {for name of input file}
   outfile : text;         {Output text file}
   g : graph;              {Network graph}
   inOut1Ct: longint;      {# times an in- and out-degree 1 vertex is removed}
   sourceSinkCt : longint; {# times the source,sink incident to 1 edge}
   factorCt: longint;      {# times factoring is ' ed}
   timer : real;       {Measures execution time}


function seconds: real;
  var hour,minute,second,sec100:word;
begin
GetTime(hour,minute,second,sec100);
seconds := 3600*hour + 60*minute + second + sec100/100
end;

procedure GetGraph (var g:graph);
   {Initialize the graph g from a text file.  The first line of the file lists
    the source and sink.  Each subsequent line contains information about an
    edge--its initial endpoint, its terminal one, and its probability.  This
    procedure does the initial parallel edge reduction on the graph.}
    var
       infile : text;       {Input file containing directed graph}
       k : integer;         {Edge or vertex counter}
       n : integer;         {Number of edge currently being read}
       vCount : integer;    {Number of graph vertices}
    hour, minute, second, sec100: word;
    year, month, day, dayofweek : word;

begin  {GetGraph}
gotoxy(5,8);
write('Enter name of file representing network -->');
readln(filename);
writeln;
assign(infile,filename);
reset(infile);
GetTime(hour, minute, second, sec100);
GetDate(year, month, day, dayofweek);
assign(outfile,concat(filename,'.out'));
rewrite(outfile);
write(outfile,'*** ',filename,' ***   ');
write(outfile,month,'/',day,'/',year mod 100,'    ');
write(outfile,hour,':');
if minute < 10 then write(outfile,'0');
writeln(outfile,minute);
vCount := 0;
n := 0;
g.maxVertex := 0;
g.vert := [];
for k := 1 to maxv do
    g.nb[k] := [];
readln(infile,g.source,g.sink);
writeln(outfile,'Source vertex = ',g.source,'   Sink vertex = ',g.sink);
writeln(outfile);
while not eof(infile) do
    begin  {Read the endpoints and probability of each edge}
    n := n + 1;
    readln(infile, g.e[n].start, g.e[n].stop, g.e[n].pr),
    writeln(outfile,g.e[n].start:3,' ----',g.e[n].stop:3,'rel = ':10,
                                                          g.e[n].pr:9:4);

    for k := 1 to n - 1 do
        if (g.e[k].start=g.e[n].start) and (g.e[k].stop=g.e[n].stop) then
            begin {Edges n and k are parallel. Combine them into edge k.}
            g.e[k].pr := g.e[k].pr*(1 - g.e[n].pr) + g.e[n].pr;
            n := n - 1
            end; {Edges n and k are parallel.}
    if g.e[n].start > g.maxVertex then
        g.maxVertex := g.e[n].start;
    if g.e[n].stop > g.maxVertex then
        g.maxVertex := g.e[n].stop;
    g.vert := g.vert + [g.e[n].start, g.e[n].stop];
    g.nb[g.e[n].start] := g.nb[g.e[n].start] + [g.e[n].stop];
    end;  {Read endpoints and probability of each edge}
g.numEdges := n;
close(infile);
for k:=1 to g.maxVertex do
```

```
        if k in g.vert then
            vCount := vCount + 1;
    write(outfile,'Number of edges = ',g.numEdges);
    writeln(outfile,'   Number of vertices = ',vCount);
    writeln('   The program is now determining the reliability of "',
                filename,'".');
    writeln;
    end;  {GetGraph}


procedure FindDegree (var g:graph);
    {Determine the degree of every vertex in the graph g.}
    var
        i : integer;  {Edge number}

    begin  {FindDegree}
    for i:= 1 to g.maxVertex do
        begin
        g.inDegree[i] := 0;
        g.outDegree[i] := 0;
        end;
    for i := 1 to g.numEdges do
        begin
        g.outDegree[g.e[i].start] := g.outDegree[g.e[i].start] + 1;
        g.inDegree[g.e[i].stop] := g.inDegree[g.e[i].stop] + 1
        end
    end; {FindDegree}


procedure Delete (var g:graph; n:integer);
    {Deletes edge n from the graph g. Degrees and neighbors are changed.}
    var
        u,v : integer;  {Endpoints of the deleted edge}
        j: integer;     {Edge number}

    begin  {Delete}
    u := g.e[n].start;
    v := g.e[n].stop;
    g.nb[u] := g.nb[u] - [v];
    g.inDegree[v] := g.inDegree[v] - 1;
    g.outDegree[u] := g.outDegree[u] -1;
    for j := n to g.numEdges-1 do
        g.e[j] := g.e[j + 1];
    g.numEdges := g.numEdges - 1;
    end;  {Delete}


procedure CleanSink (var g : graph);
    {Remove all edges in g that have the sink as starting vertex.}
    var
        j : integer;    {Edge number}
    begin   {CleanSink}
    for j := g.numEdges downto 1 do
        if g.e[j].start = g.sink then
            Delete (g,j);
    end;   {CleanSink}


procedure CleanSource (var g : graph);
    {Remove all edges in g that have the source as terminating vertex.}
    var
        j : integer;    {Edge number}

    begin  {CleanSource}
    for j := g.numEdges downto 1 do
        if g.e[j].stop = g.source then
            Delete (g,j);
    end;  {CleanSource}

 procedure CleanUp (var g:graph);
    {Eliminates all dead end and false start vertices in g.}
    var
        reduced : boolean;{Set false if a dead end or false start vertex found}
        u : integer;      {Graph vertex}
        j : integer;      {Graph edge}

    begin   {CleanUp}
    CleanSource(g);
    CleanSink(g);
    repeat
       reduced := true;
       for u:=1 to g.maxVertex do
           if (u<>g.source) and (u<>g.sink) then
              if (g.inDegree[u] = 0) or (g.outDegree[u] = 0) then
                 if (u in g.vert) then
                    begin {eliminate vertex u}
                    reduced := false;
                    for j:=g.numEdges downto 1 do
                        if (g.e[j].start = u) or (g.e[j].stop = u) then
                            Delete(g,j);
                    g.vert := g.vert - [u]
                    end;  {eliminate vertex u}
    until reduced
    end;  {CleanUp}


procedure ForwardSimplify (var g:graph; var simplified:boolean);
   {If one exists, eliminates a nonnecessary edge coming into a vertex and sets
   simplified to true.}
   var
       v : integer;  {Initial vertex for an edge}
```

```
      w : integer;  {Terminal vertex of edge out of v}
      j : integer;  {Edge number}

   begin {ForwardSimplify}
   for v:=1 to g.maxVertex do
      if (g.outDegree[v] = 1) then
          begin  {Look for edge antiparallel to the edge out of v.}
          for j:=1 to g.numedges do
              if (g.e[j].start = v) then
                  w := g.e[j].stop;
          for j:= g.numEdges downto 1 do
              if (g.e[j].stop = v) and (g.e[j].start = w) then
                  begin  {Delete the antiparallel edge.}
                  Delete(g,j);
                  simplified := true
                  end  {Delete the antiparallel edge.}
          end  {Look for edge antiparallel to the edge out of v.}
     end;  {ForwardSimplify}


procedure BackSimplify (var g:graph; var simplified:boolean);
   {If one exists, eliminates a nonnecessary edge coming out of a vertex and
   sets simplified to true.}
   var
      v : integer;  {Terminal vertex for an edge}
      w : integer;  {Initial vertex of edge out of v}
      j : integer;  {Edge number}

    begin  {BackSimplify};
    for v:=1 to g.maxVertex do
       if g.inDegree[v] = 1 then
           begin  {Look for edge antiparallel to the edge into v.}
           for j:=1 to g.numedges do
               if (g.e[j].stop = v) then
                   w := g.e[j].start;
           for j:=g.numEdges downto 1 do
               if (g.e[j].start = v) and (g.e[j].stop = w) then
                   begin  {Delete the antiparallel edge.}
                   Delete(g,j);
                   simplified := true
                   end  {Delete the antiparallel edge.}
           end  {Look for edge antiparallel to the edge into v.}
      end;  {BackSimplify}


procedure SourceSinkRed (var g:graph; var found:boolean; var factor:real);
   {If the sink of graph g has in-degree 1, then it is merged into its
   neighbor and the resulting sink is cleaned of out-edges.  If the souce has
   out-degree 1, then the parallel result occurs.  Factor is returned as the
   appropriate multiplying factor for the graph.}
   var
```

```
    j: integer;             {Possible edge incident to source or sink}
    intoSink : integer;     {Edge into the sink}
    outOfSource : integer;  {Edge out of the source}
    oldSink : integer;      {Original sink vertex}
    oldSource : integer;    {Original source vertex}

begin  {SourceSinkRed}
found :- false;
if g.inDegree[g.sink] = 1 then
    begin  {Merge the sink into its adjacent vertex.}
    found := true;
    sourceSinkCt := sourceSinkCt + 1;
    for j := 1 to g.numEdges do
        if g.e[j].stop = g.sink then
            intoSink := j;
    factor := factor * g.e[intoSink].pr;
    oldSink := g.sink;
    g.sink := g.e[intoSink].start;
    Delete(g,intoSink);
    g.vert := g.vert - [oldSink];
    CleanSink(g);
    end;  {Merge the sink into its adjacent vertex.}
if (g.outDegree[g.source] = 1) and (g.source <> g.sink) then
    begin  {Merge the source into its adjacent vertex.}
    found := true;
    sourceSinkCt := sourceSinkCt + 1;
    for j := 1 to g.numEdges do
        if g.e[j].start = g.source then
            outOfSource := j;
    factor := g.e[outOfSource] pr * factor;
    oldSource := g.source;
    g.source := g.e[outOfSource].stop;
    Delete(g,outOfSource);
    g.vert := g.vert - [oldSource];
    CleanSource(g);
    end;  {Merge the source into its adjacent vertex.}
end; {SourceSinkRed}


procedure InOutDeg1Red (var g : graph; var found:boolean);
  {G is scanned to find a vertex with in-degree and out-degree 1. If such a
  vertex is found, it it removed and the resulting graph is simplified.}
   var
     j : integer;          {Graph edge}
     u : integer;          {Graph vertex (with possible in/out degree 1)}
     inRel : real;     {Reliability of edge into u}
     outRel : real;   {Reliability of edge out of u}
     doubleRel : real;{Reliability of both edges in sequence}
     initV : integer;      {Initial vertex of edge into u}
     termV : integer;      {Terminal vertex of edge out of u}
    begin {InOutDeg1Red}
    for u:=1 to g.maxVertex do
        if (g.inDegree[u] = 1) and (g.outDegree[u] = 1) then
            begin  {Vertex u has in and out-degree 1.  Eliminate it.}
            inOut1Ct := inOut1Ct + 1;
            found := true;
            for j := g.numEdges downto 1 do
                if g.e[j].stop = u then
                    begin  {This is the edge into u.}
                    initV := g.e[j].start;
                    inRel := g.e[j].pr;
                    Delete(g,j)
                    end;  {This is the edge into u.}
            for j:=g.numEdges downto 1 do
                if g.e[j].start = u then
                    begin  {This is the edge out of u.}
                    termV := g.e[j] stop;
                    outRel := g.e[j].pr;
                    Delete(g,j);
                    end;  {This is the edge out of u.}
            doubleRel := inRel * outRel;
            g.vert := g.vert - [u];
            if termV <> initV then
                if termV in g.nb[initV] then
                    begin {Redo reliability of edge from initV to termV.}
                    for j:=1 to g.numEdges do
                        if (g.e[j].start = initV) and (g.e[j].stop = termV) then
                           g.e[j].pr := g.e[j].pr * (1 - doubleRel) + doubleRel;
                    end  {Redo reliability of edge from initV to termV.}
                else
                    begin  {Construct a new edge from initV to termV}
                    g.numEdges := g.numEdges + 1;
                    g.e[g.numEdges].start := initV;
                    g.e[g.numEdges].stop := termV;
                    g.e[g.numEdges].pr := doubleRel;
                    g.nb[initV] := g.nb[initV] + [termV];
                    g.inDegree[termV] := g.inDegree[termV] + 1;
                    g.outDegree[initV] := g.outDegree[initV] + 1;
                    end;  {Construct a new edge from initV to termV}
             exit;
             end; {Vertex u has in and out-degree 1.  Eliminate it.}
    end; {DegTwoRed}


procedure Contract (var g:graph; newSink:integer);
   {Contracts the sink of graph g into the vertex newSink.}
   var
      v : integer;         {Graph vertex}
      j : integer;         {Graph edge}
      inSink : integer;   {Edge from v into the old sink}
      inNewSink : integer;{Edge from v into the new sink}
```

```
     parallel : boolean; {True if an edge go from v to the newSink}

  begin {Contract}
  for v := 1 to g.maxVertex do
      if (g.sink in g.nb[v]) then
         begin  {There is an edge from v to the sink.  Change it.}
         parallel := false;
         for j:=1 to g.numEdges do
             if (g.e[j].start = v) and (g.e[j].stop = g.sink) then
                 inSink := j
             else if (g.e[j].start = v) and (g.e[j].stop = newSink) then
                 begin  {There is also an edge from v to the newSink.}
                 parallel := true;
                 inNewSink := j;
                 end;  {There is also an edge from v to the newSink.}
         if parallel then
             begin {Eliminate edge (v,sink). Change reliability of (v,newSink)}
             g.e[inNewSink].pr := g.e[inNewSink].pr * (1 - g.e[inSink].pr)
                                  + g.e[inSink].pr;
             Delete(g,inSink)
             end  {Eliminate edge inSink. Change reliability of inNewSink.}
          else
             begin {Change the edge inSink to have terminal vertex newSink.}
             g.e[inSink].stop := newSink;
             g.inDegree[newSink] := g.inDegree[newSink] + 1;
             g.inDegree[g.sink] := g.inDegree[g.sink] - 1;
             end;  {Change the edge inSInk to have terminal vertex newSink.}
          g.nb[v] := (g.nb[v] + [newSink]) - [g.sink];
          end;
  g.vert := g.vert - [g.sink];
  g.sink := newSink;
  CleanSink(g);
  end;  {Contract}


function Connected (var g:graph) : boolean;
  {Determine if the sink can be reached from the source (they're connected).}
  var
      comp : GraphSet;   {Vertices so far reachable from the source}
      u : integer;       {Possible vertex in comp}
      oldSet : GraphSet; {Comp on the last pass through the graph}
      changed : boolean; {True when a new vertex is added to comp}

  begin {BFS}
  comp := [g source];
  repeat
     oldSet := comp;
     changed := false;
     for u:=1 to g.maxVertex do
         if u in comp then
             comp := comp + g.nb[u];
      if comp <> oldSet then
      changed := true
   until not changed;
   Connected := g.sink in comp
   end;  {BFS}


procedure SinkEdge (var g:graph;var k:integer;var initVert:integer);
   {Find an edge k into the sink. Initial vertex is initVert.}
   var
       j : integer;     {Edge number}

   begin   {SinkEdge}
   for j:=1 to g.numEdges do
      if (g.e[j].stop = g.sink) then
         begin
         k := j;
         initVert := g.e[j].start
         end
   end;  {SinkEdge}


function Prob (g:graph) : real;
   {Returns the reliability of the graph g.}
   var
       reducible : boolean;   {True if the graph was just reduced}
       p : real;           {Factor for the probability of the reduced graph}
       markedEdge : integer;  {Edge used for factoring}
       probEdge : real;    {Probability of edge used for factoring}
       initVert : integer;    {Endpoint of factored edge}
       p1 : real;          {Probability of g with edge removed}

   begin  {Prob}
   p := 1.0;
   repeat
     reducible := false;
     CleanUp(g);
     if (g.source <> g.sink) and (g.indegree[g.sink] > 0) and
                              (g.outdegree[g.source] > 0)  then
        begin
        SourceSinkRed(g,reducible,p);
        if not reducible then
            begin  {No source or sink reduction was possible}
            BackSimplify(g,reducible);
            ForwardSimplify(g,reducible);
            InOutDeg1Red(g,reducible);
            end
        end
   until not reducible;
   if (g.source = g.sink) then
     Prob := p
```

```
  else if (g.indegree[g.sink] = 0) or (g.outDegree[g.source] = 0) then
     Prob := 0
  else
     begin {Factor the graph -- no more reductions are possible}
     SinkEdge(g,markedEdge,initVert);
     probEdge := g.e[markedEdge].pr;
     Delete(g, markedEdge);
     If not Connected(g) then
         p1 := 0
     else
         p1 := Prob(g);
     Contract(g,initVert);
     factorCt:= factorCt + 1;
     Prob :=p*( (1 - probEdge) * p1 + probEdge * Prob(g))
     end  {Factor}
   end;        {Prob}


begin {Main program}
ClrScr;
sourceSinkCt := 0;
inOut1Ct := 0;
factorCt := 0;
getGraph(g);
timer := seconds;
FindDegree(g);
writeln(outfile, Prob(g) : 0 : 18, ' = probability ');
timer := seconds - timer;
writeln(outfile,'Time = ', timer : 0 : 2,' seconds');
writeln(outfile,'Number of source/sink reductions     ', sourceSinkCt);
writeln(outfile,'Number of chain vertex reductions     ', inOut1Ct);
writeln(outfile,'Number of times factoring theorem is used  ', factorCt);
writeln(outfile);
writeln('    The program has finished executing.  Output for the program is');
writeln('  now in a file on disk.  It is a text file which you may view with');
writeln('  the Turbo Pascal editor or any other editor.');
writeln;
writeln('             Press a key to exit this program.');
repeat until keypressed;
close(outfile)
end   {Main program}.
```

F.2 Dirprog - ANSI Pascal Version

```
{This is an ANSI Pascal version of the Turbo Pascal program Dirprog written
                                                         by Page and Perry }

program modprog (infile,outfile);


{ Programmers:  Lavon Page and Jo Perry - Turbo Pascal version
                Leonard Jansen  - ANSI Pascal modifications


 Infile.dat:  A text file describing a directed graph with a source
 and sink pair of vertices.  All vertices are represented by integers
 in the range of 1..maxv.  All edges are represented by their
 endpoints followed by their reliabilities. The first line of the
 file consists of the source then sink. Each subsequent line in the
 file describes an edge.  A maximum of "maxe" edges are allowed.

 Outfile.dat:  The first lines echo the input.  The results of
 execution of the algorithm come last.  Included are the reliability
 of the graph and execution statistics: number of single edge to
 source or sink reductions, number of reductions of a vertex with
 in-degree and out-degree 1, number of times factoring is performed.}


const
    maxv = 150;         {Maximum number of vertices in the graph}
    maxe = 200;         {Maximum number of edges}

type
    degreeType = array [1..maxv] of integer; {List of vertex degrees}
    graphSet = set of 1..maxv;        {Set of vertices}
    edge = record                     {Edge in a graph}
        start,                        {Start vertex}
        stop : 1..maxv;               {Stop endvertex}
        pr : real                 {Probability of the edge}
        end; {edges}
    graph = record                    {Describes a graph}
        vert : graphSet;              {Set of graph vertices}
        source,                       {Source vertex}
        sink : integer;               {Sink vertex}
        inDegree,                     {In degree of each vertex}
        outDegree : degreeType;       {Out degree of each vertex}
        nb : array [1..maxv] of graphSet; {edge (i,j) puts j in nb[i]}
        numEdges : integer;           {Number of edges in the graph}
        maxVertex : integer;          {Largest numbered vertex in the graph}
        e : array [1..maxe] of edge   {Describes all edges in the graph}
        end; {Graph}
```

```
var

    infile,outfile : text;          {Input,Output text file}
    g : graph;                 {Network graph}
    inOut1Ct: integer;         {# times an in- and out-degree 1 vertex is removed}
    sourceSinkCt : integer; {# times the source,sink incident to 1 edge}
    factorCt: integer;         {# times factoring is used}
    timer : real;          {Measures execution time}


procedure GetGraph (var g:graph);
   {Initialize the graph g from a text file.  The first line of the file lists
    the source and sink.  Each subsequent line contains information about an
    edge--its initial endpoint, its terminal one, and its probability.  This
    procedure does the initial parallel edge reduction on the graph.}
    var
        k : integer;           {Edge or vertex counter}
        n : integer;           {Number of edge currently being read}
        vCount : integer;      {Number of graph vertices}

    begin  {GetGraph}
    reset(infile);
    rewrite(outfile);
    vCount := 0;
    n := 0;
    g.maxVertex := 0;
    g.vert := [];
    for k := 1 to maxv do
        g.nb[k] := [];
    readln(infile,g.source,g.sink);
    writeln(outfile,'Source vertex = ',g.source,'   Sink vertex = ',g.sink);
    writeln(outfile);
    while not eof(infile) do
        begin  {Read the endpoints and probability of each edge}
        n := n + 1;
        readln(infile, g.e[n].start, g.e[n].stop, g.e[n].pr);
        writeln(outfile,g.e[n].start:3,' ----',g.e[n].stop:4,'rel = ':10,
                                                                g.e[n].pr:9:4);

        for k := 1 to n - 1 do
            if (g.e[k].start=g.e[n].start) and (g.e[k].stop=g.e[n].stop) then
                begin {Edges n and k are parallel. Combine them into edge k.}
                g e[k].pr := g.e[k].pr*(1 - g.e[n].pr) + g.e[n].pr;
                n := n - 1
                end; {Edges n and k are parallel.}
        if g.e[n].start > g.maxVertex then
            g.maxVertex := g.e[n].start;
        if g.e[n].stop > g.maxVertex then
            g.maxVertex := g.e[n].stop;
        g.vert := g.vert + [g.e[n].start, g.e[n].stop];
        g.nb[g.e[n].start] := g.nb[g.e[n].start] + [g.e[n].stop];
        end;  {Read endpoints and probability of each edge}
    g.numEdges := n;
    close(infile);
    for k:=1 to g.maxVertex do
        if k in g.vert then
            vCount := vCount + 1;
    write(outfile,'Number of edges = ',g.numEdges);
    writeln(outfile,'   Number of vertices = ',vCount);
    end;  {GetGraph}


procedure FindDegree (var g:graph);
    {Determine the degree of every vertex in the graph g.}
    var
        i : integer;  {Edge number}

    begin  {FindDegree}
    for i:= 1 to g.maxVertex do
        begin
        g.inDegree[i] := 0;
        g.outDegree[i] := 0;
        end;
    for i := 1 to g.numEdges do
        begin
        g.outDegree[g.e[i].start] := g.outDegree[g.e[i].start] + 1;
        g.inDegree[g.e[i].stop] := g.inDegree[g.e[i].stop] + 1
        end
    end; {FindDegree}


procedure Delete (var g:graph; n:integer);
    {Deletes edge n from the graph g. Degrees and neighbors are changed.}
    var
        u,v : integer;  {Endpoints of the deleted edge}
        j: integer;     {Edge number}

    begin  {Delete}
    u := g.e[n].start;
    v := g.e[n].stop;
    g.nb[u] := g.nb[u] - [v];
    g.inDegree[v] := g.inDegree[v] - 1;
    g.outDegree[u] := g.outDegree[u] -1;
    for j := n to g.numEdges-1 do
        g.e[j] := g.e[j + 1];
    g.numEdges := g.numEdges - 1;
    end;  {Delete}


procedure CleanSink (var g : graph);
    {Remove all edges in g that have the sink as starting vertex.}
    var
        j : integer;    {Edge number}

    begin   {CleanSink}
    for j := g.numEdges downto 1 do
        if g.e[j].start = g.sink then
            Delete (g,j);
    end;   {CleanSink}


procedure CleanSource (var g : graph);
    {Remove all edges in g that have the source as terminating vertex.}
    var
        j : integer;    {Edge number}

    begin  {CleanSource}
    for j := g.numEdges downto 1 do
        if g.e[j].stop = g.source then
            Delete (g,j);
    end;  {CleanSource}

 procedure CleanUp (var g:graph);
    {Eliminates all dead end and false start vertices in g.}
    var
        reduced : boolean;{Set false if a dead end or false start vertex found}
        u : integer;      {Graph vertex}
        j : integer;      {Graph edge}

    begin   {CleanUp}
    CleanSource(g);
    CleanSink(g);
    repeat
       reduced := true;
       for u:=1 to g.maxVertex do
           if (u<>g.source) and (u<>g.sink) then
              if (g.inDegree[u] = 0) or (g.outDegree[u] = 0) then
                 if (u in g.vert) then
                    begin {eliminate vertex u}
                    reduced := false;
                    for j:=g.numEdges downto 1 do
                        if (g.e[j].start = u) or (g.e[j].stop = u) then
                            Delete(g,j);
                    g.vert := g.vert - [u]
                    end;  {eliminate vertex u}
    until reduced
    end;  {CleanUp}


 procedure ForwardSimplify (var g:graph; var simplified:boolean);
   {If one exists, eliminates a nonnecessary edge coming into a vertex and sets
   simplified to true.}
   var
       v : integer;  {Initial vertex for an edge}
       w : integer;  {Terminal vertex of edge out of v}
       j : integer;  {Edge number}

   begin {ForwardSimplify}
   for v:=1 to g.maxVertex do
      if (g.outDegree[v] = 1) then
          begin  {Look for edge antiparallel to the edge out of v.}
          for j:=1 to g.numedges do
              if (g.e[j].start = v) then
                  w := g.e[j].stop;
          for j:= g.numEdges downto 1 do
              if (g.e[j].stop = v) and (g.e[j].start = w) then
                  begin  {Delete the antiparallel edge.}
                  Delete(g,j);
                  simplified := true
                  end  {Delete the antiparallel edge.}
          end  {Look for edge antiparallel to the edge out of v.}
     end;  {ForwardSimplify}


procedure BackSimplify (var g:graph; var simplified:boolean);
   {If one exists, eliminates a nonnecessary edge coming out of a vertex and
   sets simplified to true.}
   var
      v : integer;  {Terminal vertex for an edge}
      w : integer;  {Initial vertex of edge out of v}
      j : integer;  {Edge number}

    begin  {BackSimplify};
    for v:=1 to g.maxVertex do
       if g.inDegree[v] = 1 then
           begin  {Look for edge antiparallel to the edge into v.}
           for j:=1 to g.numedges do
               if (g.e[j].stop = v) then
                   w := g.e[j].start;
           for j:=g.numEdges downto 1 do
               if (g.e[j].start = v) and (g.e[j].stop = w) then
                   begin  {Delete the antiparallel edge.}
                   Delete(g,j);
                   simplified := true
                   end  {Delete the antiparallel edge.}
           end  {Look for edge antiparallel to the edge into v.}
      end;  {BackSimplify}


procedure SourceSinkRed (var g:graph; var found:boolean; var factor:real);
   {If the sink of graph g has in-degree 1, then it is merged into its
   neighbor and the resulting sink is cleaned of out-edges.  If the souce has
   out-degree 1, then the parallel result occurs.  Factor is returned as the
```

```
  appropriate multiplying factor for the graph.}
  var
      j: integer;                {Possible edge incident to source or sink}
      intoSink : integer;        {Edge into the sink}
      outOfSource : integer;     {Edge out of the source}
      oldSink : integer;         {Original sink vertex}
      oldSource : integer;       {Original source vertex}

  begin  {SourceSinkRed}
  found := false;
  if g.inDegree[g.sink] = 1 then
      begin  {Merge the sink into its adjacent vertex.}
      found := true;
      sourceSinkCt := sourceSinkCt + 1;
      for j := 1 to g.numEdges do
          if g.e[j].stop = g.sink then
              intoSink := j;
      factor := factor * g.e[intoSink].pr;
      oldSink := g.sink;
      g.sink := g.e[intoSink].start;
      Delete(g,intoSink);
      g.vert := g.vert - [oldSink];
      CleanSink(g);
      end;  {Merge the sink into its adjacent vertex.}
  if (g.outDegree[g.source] = 1) and (g.source <> g.sink) then
      begin  {Merge the source into its adjacent vertex.}
      found := true;
      sourceSinkCt := sourceSinkCt + 1;
      for j := 1 to g.numEdges do
          if g.e[j].start = g.source then
              outOfSource := j;
      factor := g.e[outOfSource].pr * factor;
      oldSource := g.source;
      g.source := g.e[outOfSource].stop;
      Delete(g,outOfSource);
      g.vert := g.vert - [oldSource];
      CleanSource(g);
      end;  {Merge the source into its adjacent vertex.}
  end; {SourceSinkRed}


procedure InOutDeg1Red (var g : graph; var found:boolean);
  {G is scanned to find a vertex with in-degree and out-degree 1. If such a
  vertex is found, it it removed and the resulting graph is simplified.}
   var
      e,j : integer;          {Graph edge}
      u : integer;         {Graph vertex (with possible in/out degree 1)}
      inRel : real;    {Reliability of edge into u}
      outRel : real;   {Reliability of edge out of u}
      doubleRel : real;{Reliability of both edges in sequence}
      initV : integer;     {Initial vertex of edge into u}
```

```
    termV : integer;     {Terminal vertex of edge out of u}

  begin {InOutDeg1Red}
  e:=0;
  if e < 100 then
  for u:=1 to g.maxVertex do
      if (g.inDegree[u] = 1) and (g.outDegree[u] = 1) then
         begin  {Vertex u has in and out-degree 1.  Eliminate it.}
         inOut1Ct := inOut1Ct + 1;
         found := true;
         for j := g.numEdges downto 1 do
             if g.e[j].stop = u then
                begin  {This is the edge into u.}
                initV := g.e[j].start;
                inRel := g.e[j].pr;
                Delete(g,j)
                end;  {This is the edge into u.}
         for j:=g.numEdges downto 1 do
             if g.e[j].start = u then
                begin  {This is the edge out of u.}
                termV := g.e[j].stop;
                outRel := g.e[j].pr;
                Delete(g,j);
                end;  {This is the edge out of u.}
         doubleRel := inRel * outRel;
         g.vert := g.vert - [u];
         if termV <> initV then
             if termV in g.nb[initV] then
                begin {Redo reliability of edge from initV to termV.}
                for j:=1 to g.numEdges do
                    if (g.e[j].start = initV) and (g.e[j].stop = termV) then
                       g.e[j].pr := g.e[j].pr * (1 - doubleRel) + doubleRel;
                end  {Redo reliability of edge from initV to termV.}
             else
                begin  {Construct a new edge from initV to termV}
                g.numEdges := g.numEdges + 1;
                g.e[g.numEdges].start := initV;
                g.e[g.numEdges].stop := termV;
                g.e[g.numEdges].pr := doubleRel;
                g.nb[initV] := g.nb[initV] + [termV];
                g.inDegree[termV] := g.inDegree[termV] + 1;
                g.outDegree[initV] := g.outDegree[initV] + 1;
                end;  {Construct a new edge from initV to termV}
            e:=500;
          end; {Vertex u has in and out-degree 1.  Eliminate it.}
  end; {DegTwoRed}


procedure Contract (var g:graph; newSink:integer);
   {Contracts the sink of graph g into the vertex newSink.}
   var
```

```
     v : integer;          {Graph vertex}
     j : integer;          {Graph edge}
     inSink : integer;     {Edge from v into the old sink}
     inNewSink : integer;{Edge from v into the new sink}
     parallel : boolean; {True if an edge go from v to the newSink}

  begin {Contract}
  for v := 1 to g.maxVertex do
      if (g.sink in g.nb[v]) then
         begin  {There is an edge from v to the sink.  Change it.}
         parallel := false;
         for j:=1 to g.numEdges do
             if (g.e[j].start = v) and (g.e[j].stop = g.sink) then
                 inSink := j
             else if (g.e[j].start = v) and (g.e[j].stop = newSink) then
                 begin  {There is also an edge from v to the newSink.}
                 parallel := true;
                 inNewSink := j;
                 end;  {There is also an edge from v to the newSink.}
         if parallel then
             begin {Eliminate edge (v,sink). Change reliability of (v,newSink)}
             g.e[inNewSink].pr := g.e[inNewSink].pr * (1 - g.e[inSink].pr)
                                  + g.e[inSink].pr;
             Delete(g,inSink)
             end  {Eliminate edge inSink. Change reliability of inNewSink.}
          else
             begin {Change the edge inSink to have terminal vertex newSink.}
             g.e[inSink].stop := newSink;
             g.inDegree[newSink] := g.inDegree[newSink] + 1;
             g.inDegree[g.sink] := g.inDegree[g.sink] - 1;
             end;  {Change the edge inSInk to have terminal vertex newSink.}
          g.nb[v] := (g.nb[v] + [newSink]) - [g.sink];
          end;
  g.vert := g.vert - [g.sink];
  g.sink := newSink;
  CleanSink(g);
  end;  {Contract}


function Connected (var g:graph) : boolean;
  {Determine if the sink can be reached from the source (they're connected).}
  var
      comp : GraphSet;    {Vertices so far reachable from the source}
      u : integer;        {Possible vertex in comp}
      oldSet : GraphSet; {Comp on the last pass through the graph}
      changed : boolean; {True when a new vertex is added to comp}

  begin {BFS}
  comp := [g.source];
  repeat
     oldSet := comp;
```

```
      changed := false;
      for u:=1 to g.maxVertex do
          if u in comp then
              comp := comp + g.nb[u];
      if comp <> oldSet then
      changed := true
   until not changed;
   Connected := g.sink in comp
   end;  {BFS}


procedure SinkEdge (var g:graph;var k:integer;var initVert:integer);
   {Find an edge k into the sink. Initial vertex is initVert.}
   var
       j : integer;     {Edge number}

   begin   {SinkEdge}
   for j:=1 to g.numEdges do
      if (g.e[j].stop = g.sink) then
         begin
         k := j;
         initVert := g.e[j].start
         end
   end;  {SinkEdge}


function Prob (g:graph) : real;
   {Returns the reliability of the graph g.}
   var
       reducible : boolean;   {True if the graph was just reduced}
       p : real;           {Factor for the probability of the reduced graph}
       markedEdge : integer;  {Edge used for factoring}
       probEdge : real;   {Probability of edge used for factoring}
       initVert : integer;    {Endpoint of factored edge}
       p1 : real;          {Probability of g with edge removed}

   begin  {Prob}
   p := 1.0;
   repeat
     reducible := false;
     CleanUp(g);
     if (g.source <> g.sink) and (g.indegree[g.sink] > 0) and
                              (g.outdegree[g.source] > 0)  then
        begin
        SourceSinkRed(g,reducible,p);
        if not reducible then
            begin  {No source or sink reduction was possible}
            BackSimplify(g,reducible),
            ForwardSimplify(g,reducible);
            InOutDeg1Red(g,reducible);
            end
        end
  until not reducible;
  if (g.source = g.sink) then
     Prob := p
  else if (g.indegree[g.sink] = 0) or (g.outDegree[g.source] = 0) then
     Prob := 0
  else
     begin {Factor the graph -- no more reductions are possible}
     SinkEdge(g,markedEdge,initVert);
     probEdge := g.e[markedEdge].pr;
     Delete(g, markedEdge);
     If not Connected(g) then
         p1 := 0
     else
         p1 := Prob(g);
     Contract(g,initVert);
     factorCt:= factorCt + 1;
     Prob :=p*( (1 - probEdge) * p1 + probEdge * Prob(g))
     end  {Factor}
   end;        {Prob}


begin {Main program}
sourceSinkCt := 0;
inOut1Ct := 0;
factorCt := 0;
getGraph(g);
FindDegree(g);
writeln(outfile, Prob(g) : 0 : 18, ' = probability ');
writeln(outfile,'Number of source/sink reductions     ', sourceSinkCt);
writeln(outfile,'Number of chain vertex reductions     ', inOut1Ct);
writeln(outfile,'Number of times factoring theorem is used  ', factorCt);
writeln(outfile);
close(outfile)
end   {Main program}.
```

F.3 Network Reliability Files

F.3.1 Network A Input File

```
39 40
1  2    1.0
3  4    0.3
5  6    0.7
7  8    0.5
9  10   0.8
11 12   1.0
13 14   0.3
15 16   0.7
17 18   0.5
19 20   0.8
21 22   1.0
23 24   0.3
25 26   0.7
27 28   0.5
29 30   0.8
31 32   0.8
33  34  0.7
35  36  0.3
37  38  1.0
39  7   1.0
39  3   1.0
39  5   1.0
39  1   1.0
8   27  1.0
4   27  0.6
4   9   0.3
6   21  1.0
6   17  1.0
2   23  1.0
2   25  1.0
2   27  1.0
10  19  0.6
10  21  0.7
18  29  1.0
24  29  0.7
26  29  1.0
20  29  0.6
22  29  1.0
30  11  0.3
30  13  0.6
30  15  0.7
12  27  0.6
14  27  0.6
16  27  0.3
28  37  0.6
```

```
28  35  0.6
28  33  0.3
38  31  0.3
36  31  0.6
34  31  0.7
32  40  1.0
```

F.3.2 Network A Output File

```
*** neta ***    11/4/91    19:03
Source vertex = 39   Sink vertex = 40

  1 ----   2    rel =     1.0000
  3 ----   4    rel =     0.3000
  5 ----   6    rel =     0.7000
  7 ----   8    rel =     0.5000
  9 ----  10    rel =     0.8000
 11 ----  12    rel =     1.0000
 13 ----  14    rel =     0.3000
 15 ----  16    rel =     0.7000
 17 ----  18    rel =     0.5000
 19 ----  20    rel =     0.8000
 21 ----  22    rel =     1.0000
 23 ----  24    rel =     0.3000
 25 ----  26    rel =     0.7000
 27 ----  28    rel =     0.5000
 29 ----  30    rel =     0.8000
 31 ----  32    rel =     0.8000
 33 ----  34    rel =     0.7000
 35 ----  36    rel =     0.3000
 37 ----  38    rel =     1.0000
 39 ----   7    rel =     1.0000
 39 ----   3    rel =     1.0000
 39 ----   5    rel =     1.0000
 39 ----   1    rel =     1.0000
  8 ----  27    rel =     1.0000
  4 ----  27    rel =     0.6000
  4 ----   9    rel =     0.3000
  6 ----  21    rel =     1.0000
  6 ----  17    rel =     1.0000
  2 ----  23    rel =     1.0000
  2 ----  25    rel =     1.0000
  2 ----  27    rel =     1.0000
 10 ----  19    rel =     0.6000
 10 ----  21    rel =     0.7000
 18 ----  29    rel =     1.0000
 24 ----  29    rel =     0.7000
 26 ----  29    rel =     1.0000
 20 ----  29    rel =     0.6000
 22 ----  29    rel =     1.0000
 30 ----  11    rel =     0.3000
 30 ----  13    rel =     0.6000
 30 ----  15    rel =     0.7000
 12 ----  27    rel =     0.6000
 14 ----  27    rel =     0.6000
 16 ----  27    rel =     0.3000
 28 ----  37    rel =     0.6000
 28 ----  35    rel =    0.6000
 28 ----  33    rel =    0.3000
 38 ----  31    rel =    0.3000
 36 ----  31    rel =    0.6000
 34 ----  31    rel =    0.7000
 32 ----  40    rel =    1.0000
Number of edges = 51   Number of vertices = 40
0.15043267200 = probability
Time = 0.44 seconds
Number of source/sink reductions     7
Number of chain vertex reductions     37
Number of times factoring theorem is used  2
```

F.3.3 Network B Input File

```
46 47
14 15 0.80
14 17 0.80
18 17 0.50
18 19 0.50
16 15 0.80
16 19 0.50
16 30 0.60
 8 30 0.80
 8 43 0.50
11 27 0.80
31 30 0.50
31 44 0.60
31 45 0.60
20 21 0.50
17 21 0.70
22 21 0.50
23 24 0.50
23 25 0.70
23 26 0.70
34 35 0.50
34 37 0.50
34 38 0.50
25 30 0.70
25 39 0.50
32 30 0.70
28 29 0.80
33 30 0.80
36 35 0.60
36 37 0.60
36 38 0.60
36 39 0.60
36 40 0.60
36 41 0.60
36 42 0.60
36 43 0.60
36 44 0.60
36 45 0.60
 2 14 0.70
 4 18 0.15
 6 16 0.03
13 31 0.04
15 20 0.40
19 22 0.01
21 23 0.70
24 34 0.11
26 32 0.06
27 28 0.09
```

```
29 33 0.18
30 36 0.07
46  2 1.00
46  4 1.00
46  6 1.00
46  8 1.00
46 11 1.00
46 13 1.00
35 47 1.00
37 47 1.00
38 47 1.00
39 47 1.00
40 47 1.00
41 47 1.00
42 47 1.00
43 47 1.00
44 47 1.00
45 47 1.00
```

F.3.4 Network B Output File

```
*** netrel ***   11/4/91   21:25
Source vertex = 46   Sink vertex = 47

 14 ----  15    rel =    0.8000
 14 ----  17    rel =    0.8000
 18 ----  17    rel =    0.5000
 18 ----  19    rel =    0.5000
 16 ----  15    rel =    0.8000
 16 ----  19    rel =    0.5000
 16 ----  30    rel =    0.6000
  8 ----  30    rel =    0.8000
  8 ----  43    rel =    0.5000
 11 ----  27    rel =    0.8000
 31 ----  30    rel =    0.5000
 31 ----  44    rel =    0.6000
 31 ----  45    rel =    0.6000
 20 ----  21    rel =    0.5000
 17 ----  21    rel =    0.7000
 22 ----  21    rel =    0.5000
 23 ----  24    rel =    0.5000
 23 ----  25    rel =    0.7000
 23 ----  26    rel =    0.7000
 34 ----  35    rel =    0.5000
 34 ----  37    rel =    0.5000
 34 ----  38    rel =    0.5000
 25 ----  30    rel =    0.7000
 25 ----  39    rel =    0.5000
 32 ----  30    rel =    0.7000
 28 ----  29    rel =    0.8000
 33 ----  30    rel =    0.8000
 36 ----  35    rel =    0.6000
 36 ----  37    rel =    0.6000
 36 ----  38    rel =    0.6000
 36 ----  39    rel =    0.6000
 36 ----  40    rel =    0.6000
 36 ----  41    rel =    0.6000
 36 ----  42    rel =    0.6000
 36 ----  43    rel =    0.6000
 36 ----  44    rel =    0.6000
 36 ----  45    rel =    0.6000
  2 ----  14    rel =    0.7000
  4 ----  18    rel =    0.1500
  6 ----  16    rel =    0.0300
 13 ----  31    rel =    0.0400
 15 ----  20    rel =    0.4000
 19 ----  22    rel =    0.0100
 21 ----  23    rel =    0.7000
 24 ----  34    rel =    0.1100
 26 ----  32    rel =    0.0600
 27 ----  28    rel =    0.0900
 29 ----  33    rel =    0.1800
 30 ----  36    rel =    0.0700
 46 ----   2    rel =    1.0000
 46 ----   4    rel =    1.0000
 46 ----   6    rel =    1.0000
 46 ----   8    rel =    1.0000
 46 ----  11    rel =    1.0000
 46 ----  1[illegible]    rel =    1.0000
 35 ----  47    rel =    1.0000
 37 ----  47    rel =    1.0000
 38 ----  47    rel =    1.0000
 39 ----  47    rel =    1.0000
 40 ----  47    rel =    1.0000
 41 ----  47    rel =    1.0000
 42 ----  47    rel =    1.0000
 43 ----  47    rel =    1.0000
 44 ----  47    rel =    1.0000
 45 ----  47    rel =    1.0000
Number of edges = 65   Number of vertices = 40
0.60116629875 = probability
Time = 58.00 seconds
Number of source/sink reductions     2838
Number of chain vertex reductions     4584
Number of times factoring theorem is used  3128
```

F.3.5 Network C Input File

```
78 79
20 21 0.90
22 21 1.00
13 14 1.00
23 21 0.60
10 15 0.30
16 17 0.60
18 19 1.00
24 21 0.90
25 21 1.00
19 21 0.70
26 27 0.90
26 39 0.60
26 42 0.30
26 60 0.60
26 62 0.70
28 29 1.00
28 30 1.00
28 31 0.60
28 32 0.30
28 33 0.60
28 34 0.70
28 35 0.90
28 36 1.00
28 37 1.00
28 38 0.60
40 39 0.30
40 42 0.60
40 43 0.70
40 44 0.90
40 45 1.00
40 46 1.00
40 47 0.30
41 39 0.60
41 42 0.30
41 43 0.60
41 44 0.70
41 45 0.90
41 46 1.00
41 47 1.00
31 51 0.60
31 52 0.30
48 49 0.60
48 51 0.70
48 52 0.90
48 53 1.00
48 56 1.00
50 49 0.60
```

```
50 53 0.30
50 54 0.60
61 60 0.70
63 62 0.90
36 55 1.00
57 56 1.00
58 59 0.60
 2 20 0.30
 4 22 0.70
 6 13 0.50
 8 23 0.80
12 16 0.30
14 18 0.70
15 24 0.50
17 25 0.80
21 26 0.70
27 28 0.70
29 40 0.50
30 41 0.80
32 48 0.30
33 50 0.70
34 61 0.50
35 63 0.80
37 57 0.30
38 58 0.70
39 64 0.50
42 65 0.80
44 66 0.30
45 67 0.70
46 68 0.50
47 69 0.80
51 70 0.30
52 71 0.70
53 72 0.50
54 73 0.80
56 74 0.30
59 75 0.70
60 76 0.50
62 77 0.80
78  2 1.00
78  4 1.00
78  6 1.00
78  8 1.00
78 10 1.00
78 12 1.00
43 79 1.00
49 79 1.00
55 79 1.00
64 79 1.00
65 79 1.00
66 79 1.00
```

```
67 79 1.00
68 79 1.00
69 79 1.00
70 79 1.00
71 79 1.00
72 79 1.00
73 79 1.00
74 79 1.00
75 79 1.00
76 79 1.00
77 79 1.00
```

F.3.6 Network C Output File

```
Source vertex =         78   Sink vertex =          79

 20 ----  21    rel =    0.9000
 22 ----  21    rel =    1.0000
 13 ----  14    rel =    1.0000
 23 ----  21    rel =    0.6000
 10 ----  15    rel =    0.3000
 16 ----  17    rel =    0.6000
 18 ----  19    rel =    1.0000
 24 ----  21    rel =    0.9000
 25 ----  21    rel =    1.0000
 19 ----  21    rel =    0.7000
 26 ----  27    rel =    0.9000
 26 ----  39    rel =    0.6000
 26 ----  42    rel =    0.3000
 26 ----  60    rel =    0.6000
 26 ----  62    rel =    0.7000
 28 ----  29    rel =    1.0000
 28 ----  30    rel =    1.0000
 28 ----  31    rel =    0.6000
 28 ----  32    rel =    0.3000
 28 ----  33    rel =    0.6000
 28 ----  34    rel =    0.7000
 28 ----  35    rel =    0.9000
 28 ----  36    rel =    1.0000
 28 ----  37    rel =    1.0000
 28 ----  38    rel =    0.6000
 40 ----  39    rel =    0.3000
 40 ----  42    rel =    0.6000
 40 ----  43    rel =    0.7000
 40 ----  44    rel =    0.9000
 40 ----  45    rel =    1.0000
 40 ----  46    rel =    1.0000
 40 ----  47    rel =    0.3000
 41 ----  39    rel =    0.6000
 41 ----  42    rel =    0.3000
 41 ----  43    rel =    0.6000
 41 ----  44    rel =    0.7000
 41 ----  45    rel =    0.9000
 41 ----  46    rel =    1.0000
 41 ----  47    rel =    1.0000
 31 ----  51    rel =    0.6000
 31 ----  52    rel =    0.3000
 48 ----  49    rel =    0.6000
 48 ----  51    rel =    0.7000
 48 ----  52    rel =    0.9000
 48 ----  53    rel =    1.0000
 48 ----  56    rel =    1.0000
```

```
50 ----  49    rel =    0.6000
50 ----  53    rel =    0.3000
50 ----  54    rel =    0.6000
61 ----  60    rel =    0.7000
63 ----  62    rel =    0.9000
36 ----  55    rel =    1.0000
57 ----  56    rel =    1.0000
58 ----  59    rel =    0.6000
 2 ----  20    rel =    0.3000
 4 ----  22    rel =    0.7000
 6 ----  13    rel =    0.5000
 8 ----  23    rel =    0.8000
12 ----  16    rel =    0.3000
14 ----  18    rel =    0.7000
15 ----  24    rel =    0.5000
17 ----  25    rel =    0.8000
21 ----  26    rel =    0.7000
27 ----  28    rel =    0.7000
29 ----  40    rel =    0.5000
30 ----  41    rel =    0.8000
32 ----  48    rel =    0.3000
33 ----  50    rel =    0.7000
34 ----  61    rel =    0.5000
35 ----  63    rel =    0.8000
37 ----  57    rel =    0.3000
38 ----  58    rel =    0.7000
39 ----  64    rel =    0.5000
42 ----  65    rel =    0.8000
44 ----  66    rel =    0.3000
45 ----  67    rel =    0.7000
46 ----  68    rel =    0.5000
47 ----  69    rel =    0.8000
51 ----  70    rel =    0.3000
52 ----  71    rel =    0.7000
53 ----  72    rel =    0.5000
54 ----  73    rel =    0.8000
56 ----  74    rel =    0.3000
59 ----  75    rel =    0.7000
60 ----  76    rel =    0.5000
62 ----  77    rel =    0.8000
78 ----   2    rel =    1.0000
78 ----   4    rel =    1.0000
78 ----   6    rel =    1.0000
78 ----   8    rel =    1.0000
78 ----  10    rel =    1.0000
78 ----  12    rel =    1.0000
43 ----  79    rel =    1.0000
49 ----  79    rel =    1.0000
55 ----  79    rel =    1.0000
64 ----  79    rel =    1.0000
65 ----  79    rel =    1.0000
```

```
 66 ----  79    rel =    1.0000
 67 ----  79    rel =    1.0000
 68 ----  79    rel =    1.0000
 69 ----  79    rel =    1.0000
 70 ----  79    rel =    1.0000
 71 ----  79    rel =    1.0000
 72 ----  79    rel =    1.0000
 73 ----  79    rel =    1.0000
 74 ----  79    rel =    1.0000
 75 ----  79    rel =    1.0000
 76 ----  79    rel =    1.0000
 77 ----  79    rel =    1.0000
Number of edges =       109   Number of vertices =        73
0.615699291229248047 = probability
Number of source/sink reductions         31884
Number of chain vertex reductions         70208
Number of times factoring theorem is used       32239
```

Appendix G. *Pascal Programs*

G.1 Convert

```
Program Convert;
const
   maxarc=125;

type
   arcarray = array[1..maxarc] of integer;
   nodearray = array[1..maxarc,1..2] of integer;
   probarray = array[1..maxarc] of real;

var
   p,c:arcarray;
   prob:probarray;
   pcarc,node:nodearray;
   arclimit,head,tail,arc,nodectr,nbrin,fin,k, kc,a,b,i,j,m:integer;
   source,sink:integer;
   top,out,inp :text;

Procedure Inprob;
var i,j,k:integer;

begin
reset(inp);
for i:= 1 to maxarc do
prob[i]:=0;
while not eof(inp) do
readln(inp,k,prob[k]);
arclimit:=k;
close(inp);
end;

Procedure RI;
var i,j:integer;

begin
assign(top,'net.top');
reset(top);
nodectr:=1;

for i:= 1 to maxarc do
  begin
  p[i]:=0;
  c[i]:=0;
  for j:= 1 to 2 do
     begin
     pcarc[i,j]:= 0;
     node[i,j]:=0;
     end;
  end;

i:=1;
while not eof(top) do
  begin
  readln(top,a,b);
  pcarc[i,1]:=a;
  pcarc[i,2]:=b;
  i:=i+1;
  end;
nbrin:=i-1;
close(top);
end;


Procedure Create;
var i,j:integer;

  Procedure Findp;
     var i:integer;
     begin
     kp:=0;
     for i := 1 to  nbrin do
     if pcarc[i,2] = arc then
       begin
       kp:=kp+1;
       p[kp]:=pcarc[i,1];
       end;
     end;

  Procedure Findc;
     var i:integer;
     begin
     kc:=0;
     for i:= 1 to nbrin do
     if pcarc[i,1] = arc then
        begin
        kc:=kc+1;
        c[kc]:=pcarc[i,2];
        end;
     end;

begin
nodectr:=1;
for j:= 1 to nbrin do
begin
```

```
   arc:=pcarc[j,1];
   if node[arc,1] = 0 then
      begin;
      head:=nodectr;
      Findp;
      for m:=1 to kp do
        for i:=1 to nbrin do
        if pcarc[i,1] = p[m] then
          begin
          node[pcarc[i,2],1]:=head;
          node[pcarc[i,1],2]:=head;
          end;
      nodectr:=nodectr+1;
      end;
   if node[arc,2] = 0 then
      begin;
      tail:=nodectr;
      Findc;
      for m:=1 to kc do
          for i:= 1 to nbrin do
          if pcarc[i,2] = c[m] then
            begin
            node[pcarc[i,1],2]:=tail;
            node[pcarc[i,2],1]:=tail;
            end;
      nodectr:=nodectr+1;
      end;
end;
nodectr:=0;
for i:= 1 to nbrin do
   for j:=1 to 2 do
      if node[i,j] > nodectr then
         nodectr := node[i,j];
for i := 1 to nbrin do
  if (node[i,1] > 0) and (node[i,2] = 0) then
     begin
     node[i,2] := nodectr+2;
     fin := i;
     end;
for i := 1 to fin do
   if node[i,1] = 0 then
   node[i,1] := nodectr+1;
end;
(* Main Program *)

Begin
   assign(out,'netrel');
   assign(inp,'prob.f');
   rewrite(out);
   Inprob;
   RI;
```

```
    Create,
    source:=nodectr+1;
    sink:=nodectr+2;
    writeln(out,source:2,' ',sink:2);
    for m:= 1 to fin do
      writeln(out,node[m,1]:2,' ',node[m,2]:2,' ',prob[m]:3:2);
    close(out);
end.
```

G.2 Capinv

```
Program Capinv;

const
    maxpath = 200;
    maxarc = 85;

type
    network = array[1..maxpath,1..maxarc] of integer;
    path = array[1..maxpath] of integer;
    arc = array[1..maxarc] of real;

var
    arcpath : network;
    cap,prob : arc;
    arclimit, pathlimit : integer;
    bud,inv : real;
    pfile, cfile,pathfile,gams :text;

Procedure Getprob; {Reads in arc probabilities into array prob[i]}
var i,k:integer;

begin
   reset(pfile);
   arclimit:=0;
   for i:=1 to maxarc do
   begin
   prob[i]:=0;
   end;
   while not eof(pfile) do
      begin
      readln(pfile,k,prob[k]);
      arclimit:=arclimit+1;
      end;
   close(pfile);
end;


Procedure Getpaths; {Reads network paths into array arcpath[i,j]}
var i,j : integer;

begin {read paths into array arcpath}
    reset(pathfile);
    pathlimit:= -1;
    for i := 1 to maxpath do
       begin {initialize}
       for j := 1 to maxarc do
```

```
          arcpath[i,j]:=0;
       end; {initialize}
    while not eof(pathfile) do
       begin {read path number}
       read(pathfile,i);
       pathlimit:=pathlimit+1;
       while not eoln(pathfile) do
          begin {read arcs(j) in path(i)}
          read(pathfile,j);
          arcpath[i,j] := 1;
          end; {read arcs}
       end; {read path number}
       close(pathfile);
    end; {read paths into array arcpath}

Procedure Getcap; {Reads in arc capacities into array cap[i]}
var i,k:integer;

begin
   reset(cfile);
   for i:=1 to maxarc do
   cap[i]:=0;
   while not eof(cfile) do
      begin
      readln(cfile,k,cap[k]);
      end;
   close(cfile);
end;

Procedure Makegams;
   var i,j:integer;

   begin
   assign(gams,'capinv.gms');
   rewrite(gams);
   writeln(gams,'$OFFSYMXREF OFFSYMLIST');
   writeln(gams);
   writeln(gams,'SETS');
   writeln(gams,'   I   arcs  /1 * ',arclimit,'/');
   writeln(gams,'   J   paths /1 * ',pathlimit,'/;');
   writeln(gams);
   writeln(gams,'PARAMETERS');
   writeln(gams);
   writeln(gams,'   U(I)  arc capacities');
   writeln(gams,'   /');
   for i:= 1 to arclimit do
   writeln(gams,i:2,cap[i]:6);
   writeln(gams,'   /');
   writeln(gams);
   writeln(gams,'   P(I)  arc probabilities');
   writeln(gams,'   /');
```

```
for i:=1 to arclimit do
writeln(gams,i:2,prob[i]:5:2);
writeln(gams,'   /');
writeln(gams);
writeln(gams,'   A(I,J)  arc-path matrix');
writeln(gams,'   /');
for j := 1 to arclimit do
begin
   for i := 1 to pathlimit do
   begin
   if arcpath[i,j] > 0 then
   writeln(gams,j:2,'.',i:1,arcpath[i,j]:2);
   end
end;
writeln(gams,'   /;');
writeln(gams);
   writeln(gams,'SCALAR C cost of incr arc cap by 1  / ',inv:4,' /;');
writeln(gams,'SCALAR B total budget available  / ',bud:8,' /;');
writeln(gams);
writeln(gams,'PARAMETER R(J)  path reliabilities ;');
writeln(gams);
writeln(gams,'   R(J) = PROD(I $ A(I,J), P(I)) ;');
writeln(gams);
   writeln(gams,'VARIABLES');
writeln(gams,'   X(I)  arc cap increase ');
writeln(gams,'   F(J)  flow on path J ');
writeln(gams,    Z     network lower bound maxflo ;');
writeln(gams);
writeln(gams,'POSITIVE VARIABLES F,X ;');
writeln(gams);
writeln(gams,'EQUATIONS');
writeln(gams);
writeln(gams,'   MAXFLO');
writeln(gams,'   PATHFLO(I)');
writeln(gams,'   BUDGET ;');
writeln(gams);
writeln(gams,'MAXFLO .. Z =E= SUM(J, R(J)*F(J)) ;');
write(gams,'PATHFLO(I)$(U(I) ne 0) ..');
write(gams,'SUM(J $ A(I,J), F(J)) =L= U(I)+X(I) ;');
writeln(gams);
writeln(gams,'BUDGET .. SUM(I, C*X(I)) =L= B ;');
writeln(gams);
writeln(gams,'MODEL CAPINV  /ALL/ ;');
writeln(gams);
writeln(gams,'OPTION LIMROW = 0');
writeln(gams);
writeln(gams,'SOLVE CAPINV USING LP MAXIMIZING Z ;');
writeln(gams);
writeln(gams,'DISPLAY X.L,F.L ;');
close(gams);
end;
```

```
(* Main Program *)

Begin
assign(pfile,'prob.f');
assign(pathfile,'path.f');
assign(cfile,'cap.f');
write('Enter cost of increasing 1 unit of capacity  ');
read(inv);
writeln;
write('Enter total budget  ');
read(bud);
Getprob;
Getpaths;
Getcap;
Makegams;
End.
```

G.3 Relinv

```
Program Hueristic;

const
    maxpath = 200;
    maxarc = 85;

type
    network = array[1..maxpath,1..maxarc] of integer;
    path = array[1..maxpath] of integer;
    arc = array[1..maxarc] of real;

var
    arcpath : network;
    impind,prob : arc;
    card, np : path;
    arclimit, pathlimit : integer;
    cap,inv : real;
    pfile, pathfile,gams :text;

Procedure Getprob; {Reads in arc probabilities into array prob[i]}
var i,k:integer;

begin
   reset(pfile);
   arclimit:=0;
   for i:=1 to maxarc do
   begin
   prob[i]:=0;
   end;
   while not eof(pfile) do
      begin
      readln(pfile,k,prob[k]);
      arclimit:=arclimit+1;
      end;
   close(pfile);
end;


Procedure Getpaths; {Reads network paths into array arcpath[i,j]}
var i,j : integer;

begin {read paths into array arcpath}
    reset(pathfile);
    pathlimit:= -1;
    for i := 1 to maxpath do
       begin {initialize}
       card[i]:=0;
      for j := 1 to maxarc do
         arcpath[i,j]:=0;
      end; {initialize}
   while not eof(pathfile) do
      begin {read path number}
      read(pathfile,i);
      pathlimit:=pathlimit+1;
      while not eoln(pathfile) do
         begin {read arcs(j) in path(i)}
         read(pathfile,j);
         arcpath[i,j] := 1;
         card[i] := card[i] + 1;
         end; {read arcs}
      end; {read path number}
      close(pathfile);
   end; {read paths into array arcpath}


  Procedure Computenp ;
  var i,k: integer;
  begin
  for k:= 1 to pathlimit do {initialize np}
  np[k]:=0;
  for k:= 1 to pathlimit do
     for i:= 1 to pathlimit do
     if card[k] = card[i] then
        np[k]:= np[k]+1;
  end;

Procedure Computeii ;
  var i,j:integer; cardsq:path;

  begin
  for j:= 1 to arclimit do {initialize impind}
     impind[j]:= 0;
  for i:= 1 to pathlimit do {square card array}
     cardsq[i]:= card[i]*card[i];
  for j:= 1 to arclimit do
     for i:= 1 to pathlimit do
        impind[j]:= impind[j] +
(arcpath[i,j]/(np[i]*cardsq[i]))*1000;
  end;

Procedure Makegams ;
  var i:integer;

  begin
  assign(gams,'relhuer.gms');
  rewrite(gams);
  writeln(gams,'$OFFSYMXREF OFFSYMLIST');
  writeln(gams);
  writeln(gams,'SETS');
  writeln(gams,'   I   arcs  /1 * ',arclimit,'/;');
  writeln(gams);
  writeln(gams,'PARAMETERS');
  writeln(gams);
  writeln(gams,'   R(I)  arc reliability index');
  writeln(gams,'   /');
  for i:= 1 to arclimit do
  writeln(gams,i:2,impind[i]:8:4);
  writeln(gams,'   /');
  writeln(gams);
  writeln(gams,'   P(I)  arc probabilities');
  writeln(gams,'   /');
  for i:=1 to arclimit do
  writeln(gams,i:2,prob[i]:5:2);
  writeln(gams,'   /;');
  writeln(gams);
  writeln(gams,'SCALAR C cost of incr arc rel by .1  / ',inv:4,' /;');
  writeln(gams,'SCALAR B total budget available  / ',cap:8,' /;');
  writeln(gams);
  writeln(gams,'VARIABLES');
  writeln(gams,'   X(I)  arc rel increase ');
  writeln(gams,'   Z     objective for rel index ;');
  writeln(gams);
  writeln(gams,'POSITIVE VARIABLE X;');
  writeln(gams);
  writeln(gams,'EQUATIONS');
  writeln(gams);
  writeln(gams,'   MAXINDEX');
  writeln(gams,'   REL(I)');
  writeln(gams,'   BUDGET ;');
  writeln(gams);
  writeln(gams,'MAXINDEX .. Z =E= SUM(I, R(I)*(P(I)+X(I))) ;');
  writeln(gams,'REL(I) .. P(I)+(.1*X(I)) =L= 1 ;');
  writeln(gams,'BUDGET .. SUM(I, C*X(I)) =L= B ;');
  writeln(gams);
  writeln(gams,'MODEL RELHUER  /ALL/ ;');
  writeln(gams);
  writeln(gams,'OPTION LIMROW = 0');
  writeln(gams);
  writeln(gams,'SOLVE RELHUER USING LP MAXIMIZING Z ;');
  writeln(gams);
  writeln(gams,'DISPLAY X.L ;');
  close(gams);
  end;

(* Main Program *)

Begin
assign(pfile,'prob.f');
assign(pathfile,'path.f');
```

```
write('Enter cost of increasing .1 units of survivability  ');
read(inv);
writeln;
write('Enter total budget  ');
read(cap);
Getprob;
Getpaths;
Computenp;
Computeii;
Makegams;
End.
```

Appendix H. *Graphical Network Analyzer (GNA) User's Guide (for Sun workstations)*

by Andrew Jaffee August 22, 1991

Contents

• Starting GNA

Type <gna> at your UNIX system's prompt. The network drawing screen and menu will appear. Click left with the mouse to get started.

• Drawing normal nodes

With the mouse, click on <node>, then click on <place normal>. A window will appear in the lower left-hand corner of the screen. This is the input box. The input box appears when you need to input data and disappears when you're done. First, you will be prompted to enter the node's reliability. Enter it and press <return>. If you entered a percentage of less than 1.0, you will be prompted to enter the cost of increasing one unit of reliability in the node. Type it and press <return>. You will be prompted to enter the node's capacity. Enter * if the node has infinite capacity or some real number if less than infinite and press <return>. If the capacity is finite, you will be prompted to enter the cost per unit increase of capacity in the node. Enter it and press <return>. You will be prompted to enter whether this node is continuous(c) or discrete(d). Enter c or d and press <return>. If you pressed d, you will be asked to enter the node's capacity increase. Enter it and press <return>. Now place the cursor on the screen where you want to position the node and click. The node will appear. To label the node, click <new>, then <label>. Place the cursor where you want the label and click left.-To draw a new node, click on <new>, then <place normal>, and repeat the steps described earlier. When finished drawing nodes, click on <quit>. The main menu will reappear.

• Drawing queuing nodes

With the mouse, click on <node>, then click on <place queue>. The input window will appear in the lower left-hand corner of the screen. First, you will be prompted to enter the node's service rate. Enter it and press <return>. Then you will be prompted to enter the node's service rate variability. Enter it and press <return>. Then you will be prompted to enter the number of servers at the node. Enter it and press <return>. Now place the cursor on the screen where you want to position the node and click. The node will appear. To label the node, click <new>, then <label>. Place the cursor where you want the label and click left. To draw a new node, click on <new>, then <place queue>, and repeat the steps described earlier. When finished drawing nodes, click on <quit>. Then main menu will reappear.

• Drawing arcs

Click on <arc> and then on <place>. The input window will appear. Enter arc data as you did in Drawing normal nodes. Now move the cursor over the origin node for this arc and double-click. Double-click on the terminal node for this arc and the arc will appear. To label the arc, click <new>, then <label>. Place the cursor where you want the label and click left. To draw another arc, click on <new>, then <place>, and repeat the steps described earlier. When finished drawing arcs, click on <quit>. Then main menu will reappear.

• Drawing routes between queuing nodes

From the main menu, click on <path>. The input window will appear prompting you to enter the route's arrival rate and variability parameter. Enter them. Double-click on the first node in the path, then double-click on the second node, double click and the third node, ... When you've double-clicked on the last node in the path, click on <done> in the route dialog box. To draw a cycle, double click twice on the same node. Then you can draw another path by clicking on <new> and then <path> or exit by clicking on <quit>.Undoing your last draw operation (queues and nodes)

• Undo

If you draw a node or arc badly, accidentally, or you change your mind, you can fix it. From your menu, pick the <undo> command, and the last node or arc drawn will be deleted from the network drawing area. The network data base will also be corrected. Remember, this operation will only remove your most recently drawn node or arc.

• Saving the network you've drawn

Click on <save>, the input window will appear prompting you to "ENTER FILE NAME:". Type the file name and hit <return>. The network will be saved in the file you named in the current directory.

• Analyzing the network with an operations research (OR) model : FORMULA

You can analyze your network using the FORMULA model and the LP/MIP 83 and GINO linear/nonlinear/mixed-integer programming packages. From the main menu, pick <new> then <model>. The model menu will appear. Pick <formula> and then <convert>. The input window will appear. You will be prompted to enter the budget for improving the capacity and the budget for improving the reliability of the network drawn on screen. Now open a DOS window and enter api at the DOS prompt. The ARITY/PROLOG interpreter will appear. Type consult('formula2.ari'). at the ?- prompt and hit <return>. Then type go. and hit <return>. FORMULA will prompt you to enter your GNA data file name. Type form_in.ari and hit <return>. Now you will see a menu giving you several choices as to how to analyze your network. Pick 1 for Find all paths and calculate reliabilities. You will be asked where you want output to go. Pick 2 for File. To see all paths through the network graphically, see Displaying all paths through the network below. FORMULA now asks you if you want continue execution, type y and the main menu will reappear. Pick 3 for Generate the Lower Bound Formulation. You will be asked where you want output to go. Pick 2 for File. Now open another DOS window and enter the command bottle at the DOS prompt (this is a batch file containing the command line lp83 output3.lp marginanalysis yes > lp83.out). This invokes the program LP83 to find, among other things, all bottlenecks in your network. To see all bottlenecks graphically, see Displaying all bottlenecks (arcs/nodes) in the network below. FORMULA now asks you if you want continue execution, type y and the main menu will reappear. Pick 5 for Generate Investment Strategy Model I.

You will be asked where you want output to go. Pick 2 for File. In your other DOS window enter the command capacity at the DOS prompt (this is a batch file containing the command line lp83 output5.lp marginanalysis yes > lp83.out). This invokes the LP83 program to find, among other things, all arcs in the network that need capacity improvement and the suggested capacity improvements. To see these suggested improvements graphically, see Displaying capacity improvements below. FORMULA now asks you if you want continue execution, type y and the main menu will reappear. Pick 7 for Generate Investment Strategy Model 3. You will be asked where you want output to go. Pick 2 for File. In your other DOS window enter the command gino. A colon will appear. Type retr output7.nlp and hit <return>. Type divert and hit <return>. You'll be prompted for a file name. Type gino.out and hit <return>. Type go and hit <return>. When the colon reappears, type quit and hit <return>. These actions will solve the Model 3 formulation to find network elements that need reliability improvement along with suggested improvements. To see these suggested improvements graphically, see Displaying reliability improvements below. You have several other options when running FORMULA. You can generate the maximum flow and upper bound formulations. You can also generate three investment strategy models for improving network performance. These formulations can be solved using the commercial packages LP/MIP 83 and GINO. Please consult the FORMULA, LP/MIP 83, and GINO documentation to see how to use these packages.

- Displaying all paths through the network

  Now move the mouse back from the DOS window into the GNA window. Pick <new> then <paths>. The first path through the network will appear. Its reliability and flow will be shown in the paths dialog box. Click <next> to see the next path or <done> to quit. Pick <quit> to <return> to the main menu.

- Displaying all bottlenecks (arcs/nodes) in the network

  Now move the mouse back back from the DOS window into the GNA window. Pick <new> then <bottlenecks>. All bottlenecks will be displayed in the network. Each node or arc that is a bottleneck will have the following symbol next to it. Click on <done> in the bottlenecks dialog box to continue.Displaying capacity improvements. Now move the mouse back back from the DOS window into the GNA window. Pick <new> then <capacity>. All nodes and arcs that need capacity improvement will be displayed in the following manner. The labels of the elements being improved will reflect the new values. GNA displays network nodes and arcs in sizes that correspond to their capacities. An improved element will have the newly sized element displayed in green with the original element superimposed over it. For example, Click on <done> in the capacity improvements dialog box to continue. Displaying reliability improvements. Now move the mouse back back from the DOS window into the GNA window. Pick <new> then <reliability>. All nodes and arcs that need reliability improvement will be displayed in the following manner. The labels of the elements being improved will reflect the new values. Elements will be colored as follows: reliability (p) >= 0 and p <= 0.5 will be red, p >= 0.5 and p < 1.0 will be orange, and p = 1.0 will be black. Click on <done> in the reliability improvements dialog box to continue.

- QNA (Queuing Network Analyzer)

You can analyze a queuing network using QNA. After you've draw your queuing network, with nodes and routes, pick <model> from the main menu. From the model menu that appears, pick <qna>. The qna menu will appear. Go to another UNIX window and enter the command qna < qna.in > qna.out. QNA will execute and place output containing the analysis of your network into a file named "qna.out".

- Displaying all bottlenecks in the network

Now move the mouse back back from the UNIX window into the GNA window. Pick <new> then <bottlenecks>. All bottlenecks will be displayed in the network as they were under FORMULA (above). Click on <done> in the bottlenecks dialog box to remove bottleneck symbols.

- Displaying customers in node queues

Pick <new> then <queues> to display the customers in each node's queue. You will see dots lined up in each queue representing the number of customers in the queue. Visually, queues will look like the following: Click on <done> in the queues dialog box to remove queue symbols.Displaying other congestion/performance measuresSimilarly, you can click on <times>, <rates>, <visits>, and <traffic> to see sojourn times at nodes, arrival and departure rates to and from nodes, expected number of visits to nodes, and traffic intensities at nodes, respectively. When finished viewing each, click on <done> in the respective dialog box.

- Displaying overall network performance information

Pick <new> then <performance>. A window will appear displaying textual and symbolic representations of network throughout, flows in and out of the network, expected total number of customers in the network, and expected total sojourn time in the network. Click on <close> when done viewing this data.

- Navigating in GNA windows

You can change the vantage point from which you see the graphics images on screen. This can be done to back away from an image to see the whole thing, or to zoom in to get a closer look. You can also rotate images around three axes. To do these things, pick <camera> from the main menu. The menu options should be self-explanatory.Exiting GNA: Click on <quit> from the main menu.

Bibliography

1. Abraham, J.A. "An Improved Algorithm for Network Reliability,"*IEEE Transactions on Reliability, Vol. R-28, No. 1*: 58-61 (April 1979).
2. Aggarwal, K.K. "Integration of Reliability and Capacity in Performance Measure of a telecommunication Network," *IEEE Transactions on Reliability, Vol. R-34, No. 2*: 184-186 (June 1985).
3. Aggarwal, K.K. and K.B. Misra "A Fast Algorithm for Reliability Evaluation," *IEEE Transactions on Reliability, Vol. R-24, NO. 1*: 83-85 (April 1975).
4. Arity Corporation, *The Arity/Prolog Language Reference Manual.* Concord, Massachusetts, 1988.
5. Bailey, T. G.*Response Surface Analysis of Stochastic Network Performance.* MS thesis, AFIT/GOR/ENS/88D-01. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB, OH, December 1988 (AFD-A202561).
6. Borland International, *The Turbo Pascal Version 5.0 User and Reference Guides.* Scotts Valley, California, 1989.
7. Brooke, A. and others. *GAMS a User's Guide.* Redwood City, CA: The Scientific Press, 1988.
8. Brown, D. B. "A Computerized Algorithm for Determining the Reliability of Redundant Configurations," *IEEE Transactions on Reliability, Vol. R-20, No. 3*: 121-124 (August 1971).
9. Buchanan, J. T. and H. G "Daellenbach. A Comparitive Evaluation of Interactive Solution Methods for Multiple Objective Decision Models," *European Journal of Operational Research, Vol. 29, No. 3*: 353-359 (June 1987).
10. Committee on the Next Decade in Operations Research. "Operations research: The Next Decade," *Operations Research, Vol. 36, No. 4*: 619-637 (July-August 1988).
11. Evans, G. W. "An Overview of Techniques for Solving Multiobjective Mathematical Programs," *Management Science, Vol. 30, No. 11*: 1268-1282 (November 1984).
12. Gaught, W. L. *Improving Reliability in a Stochastic Communication Network.* MS thesis, AFIT/GSO/ENS/90D- 10. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB OH, December 1990.
13. Heidtmann, K. D. "Smaller Sums of Disjoint Products by Subproduct Inversion,"*IEEE Transactions on Reliability, Vol. 38, No. 3*: 305-311 (August 1989).

14. Jaffee, A. and Y. Chan *The Graphical Network Analyzer: A Computer Graphics Package for Stochastic and Deterministic Network Analysis in Operational Sciences.* Preliminary report presented at the 4th International Conference on Computer Graphics and Descriptive Geometry, Miami, Florida, 1990.

15. Jain, S.P. and K. Gopal, "A Hueristic Method of Link Reliability Assignment for Maximal Reliability,"*Microelectronics Reliability, Vol. 30, No. 4*: 673-679 1990.

16. Li, V. O. K. and J. A. Sylvester "Performance Analysis of Networks with Unreliable Components," *IEEE Transactions on Communications, Vol. COM-32, No. 10*: 1105-1110 (October 1984).

17. Locks, M. O. "A Minimizing Algorithm for Sum of Disjoint Products,"*IEEE Transactions on Relaibility Vol. R-36, No. 4*: 445-453 (October 1987).

18. Page, L. B. and J. E. Perry "Reliability of Directed Networks using the Factoring Theorem," *IEEE Transactions on Reliability, Vol. 38, No. 5*: 556-562 (December 1989).

19. Provan, S. J. and M. O. Ball "Computing Network Reliability in Time Polynomial in the Number of Cuts," *Operations Research, Vol. 32, No. 3*: (May-June 1984).

20. Seo, F. and M. Sakawa. *Multiple Criteria Decision Analysis in Regional Planning.* Dordrecht, Holland: D. Reidel Publishing Company, 1988.

21. Shin, W. S. and A. Ravindran "Interactive Multiple Objective Optimization: Survey I — Continuous Case," *Computers and Operations Research," Vol. 18, No. 1*: 97-114 (1991).

22. Wood, K. R. "Factoring Algorithms for Computing K-Terminal Network Reliability," *IEEE Transactions on Reliability, Vol. R-35, No. 3*: 269-278 (August 1986).

23. Yim, E. *Improving the Survivability of a Stochastic Communication Network.* MS thesis, AFIT/GOR/ENS/88D-01. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH,December 1988 (AD-A202872).

24. Yim, E. and others. *Exact and Approximate Improvement to the Throughput of a Stochastic Network.* Submitted to Joint Meeting of the Operations Research Society and the Institute of Management Science, Vancouver, B. C. Canada, May 1989.

25. Zeleny, M. *MCDM:Past Decade and Future Trends.* Grenwich, Connecticut: JAI Pres Inc, 1984.