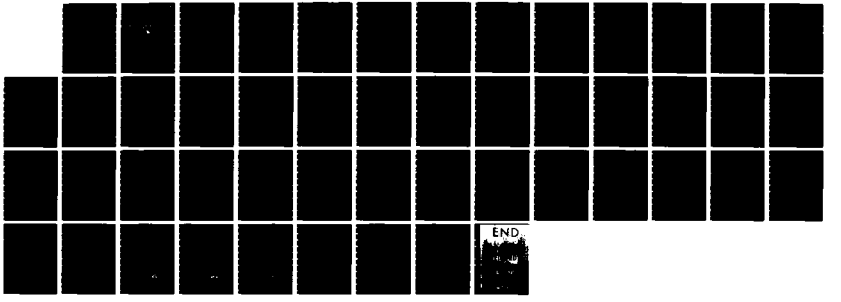
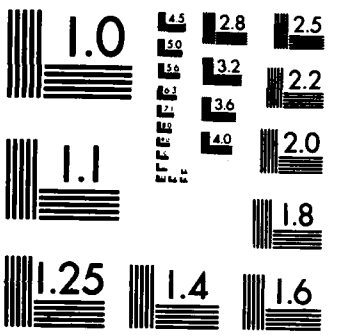
AD-A136 900 EFFICIENT COMPUTATION OF THE K-TERMINAL RELIABILITY OF DIRECTED ACYCLIC NETWORKS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA L CHAN SEP 83 1/1

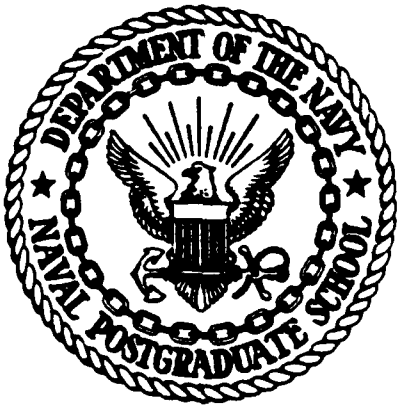
UNCLASSIFIED F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

EFFICIENT COMPUTATION OF THE K-TERMINAL RELIABILITY OF DIRECTED ACYCLIC NETWORKS

by

Lee, Chan

September 1983

Thesis Advisor: R. Kevin Wood

Approved for public release; distribution unlimited

84 01 17 101

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A136 [illegible]	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Efficient Computation of the K-terminal Reliability of Directed Acyclic Networks		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis September 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Lee, Chan		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE
		13. NUMBER OF PAGES 47
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Network reliability, Acyclic, Directed graph, Vertex, Edge, Reduction, Decomposition, Complexity

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Several topological reduction and decomposition techniques are developed to decrease the complexity of computing $R_{SK}(D)$, the source-to-K terminal reliability of an acyclic directed network D with independent component failures. $R_{SK}(D)$ is computed in $O(|V||E|)$ time when D can be completely reduced. When not completely reduced, a graph D' remains

#20 - ABSTRACT - (Continued)

such that $R_{sK}(D)=M\ R_{sK_1}(D_1)R_{sK_2}(D_2)\ \ldots\ R_{sK_m}(D_m)$ where M is a known constant and the D_i are one or more separable components of D'. A simple scheme, exponential in $|V_i-K_i|$ is given for computing $R_{sK}(D)$. When $|V_i-K_i|$ become too large, a truncated version of this scheme usually gives an excellent lower bound on $R_{sK_i}(D_i)$ and thus an excellent lower bound on $R_{sK}(D)$.

A program using these techniques has been coded in FORTRAN and tested on "complete" acyclic graphs and "street" networks with up to 100 vertices. Running on an IBM 3033AP under FORTRAN H (Extended), total CPU time for computing exact reliability is less than 3.5 seconds when $|V-K|\leq 10$ and $|V|\leq 100$.

Approved for Public Release; Distributed unlimited

Efficient Computation of the K-Terminal Reliability of Directed Acyclic Networks

by

Lee, Chan
Major, Republic of Korea A.F.
B.S., Republic of Korea A.F. Academy, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1983.

Author: Chan Lee

Approved by: R. Ken Wood
Thesis Advisor

V. Eslary
Second Advisor

Chairman, Department of Operations Research

K.T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

Several topological reduction and decomposition techniques are developed to decrease the complexity of computing $R_{sK}(D)$, the source-to-K-terminal reliability of an acyclic directed network D with independent component failures. $R_{sK}(D)$ is computed in $O(|V||E|)$ time when D can be completely reduced. When not completely reduced, a graph D' remains such that $R_{sK}(D) = M \cdot R_{sK_1}(D_1) \cdot R_{sK_2}(D_2) \cdots R_{sK_m}(D_m)$ where M is a known constant and the D_i are one or more separable components of D'. A simple scheme, exponential in $|V_i - K_i|$ is given for computing $R_{sK}(D)$. When $|V_i - K_i|$ become too large, a truncated version of this scheme usually gives an excellent lower bound on $R_{sK_i}(D_i)$ and thus an excellent lower bound on $R_{sK}(D)$.

A program using these techniques has been coded in FORTRAN and tested on "complete" acyclic graphs and "street" networks with up to 100 vertices. Running on an IBM 3033AP under FORTRAN H (Extended), total CPU time for computing exact reliability is less than 3.5 seconds when $|V-K| \leq 10$ and $|V| \leq 100$.

TABLE OF CONTENTS

I. INTRODUCTION

Often, in the design and production of complex system such as computer network, good system reliability is a critical requirement. Network reliability is rarely treated explicitly and quantitatively, however, largely because of problems of computational complexity and because few efficient computer programs exist for computing network reliability. Only in the last few years have research and development in this area provided usable results.

We wish, as efficiently as possible, to determine the network reliability for specific class of networks. The purpose of this thesis is to develop new reduction and decomposition techniques for computing network reliability and to show how computational complexity can be decreased by using these techniques.

The network model which is used in this thesis may be thought of as a communication network with directed edges, allowing only one way communication from tail vertex to head vertex. Additionally, we impose the restriction that the network have no directed cycles. Such a model may be appropriate in hierarchical communication networks such as command networks. Also, such networks may arise as subproblems while computing the reliability of general directed networks, Agrawal [1982].

Networks can be divided into various classes, e.g., directed, undirected, cyclic, acyclic, etc. A large amount of literature exists on each class. (See Agrawal [1982], Ball and Provan [1981], Satyanarayana [1982], Hagstrom [1980] and Satyanarayana and Prabahkar [1978] for directed networks; see Ball and Provan [1981], Johnson [1982], Satyanarayana [1982], Satyanarayana and Chang [1981], Satyanaryana and Wood [1982], Valiant [1978], and Wood [1982] for undirected

networks; see Agrawal [1982] and Satyanarayana and Prabhakar [1978] for cyclic versus acyclic directed networks.) The specific model we will consider is the directed acyclic network, denoted $D(V,E)$ where V is the set of vertices and E is the set of edges. Each edge is represented by an ordered pair of vertices $e=(u,v)$. The ordering implies that the flow of communication can move along e from u to v only.

Each edge e_i in the network functions with probability p_i and fails with probability $q_i = 1-p_i$. All edge failures are assumed to be independent events. For simplicity in the initial discussion, vertices are assumed to be completely reliable. However, the extension to unreliable vertices will be handled briefly.

Vertex u can send communication to vertex v in D if there is a directed path of working edges from u to v. One of the vertices s is designated as the "source" or the "root". With respect to the source, there are three different measures of reliability which are usually studied in directed networks:

(1) The source-to-terminal (ST) reliability, $R_{st}(D)$, is the probability that s can send communication to a specified vertex t in D.

(2) The source-to-all-terminal (SAT) reliability, $R_{sV}(D)$, is the probability that s can send communication to all vertices of D.

(3) The source-to-K-terminal (SKT) reliability, $R_{sK}(D)$, is the probability that s can send communication to every vertex in a specified set K (these vertices are referred to as "K-vertices") with, by convention, $s \in K$.

Of course, (1) and (2) are special cases of (3). We consider the general case (3) in this thesis.

Generally, the of problem computing network reliability is NP-hard. Certain reduction and decomposition schemes exist which reduce the size of the problem while preserving network reliability, e.g., Agrawal[1982], Shogan[1978] and Wood [1982], and, for special classes of networks, it is thereby possible to compute

reliability in polynomial time. In general, however, the solution time grows exponentially in the size of the network. In this thesis, we develop a method for computing reliability which, in general, is exponential in the size of D, but which is polynomial when the number of non-K-vertices, i.e., $|V-K|$, is fixed.

The rest of this thesis is outlined as follows.

Chapter 2 contains a necessary definitions and graph theoretic concepts.

Chapter 3 develops reduction and decomposition techniques to reduce the size of a network in order to decrease computation time. Some of these techniques are new.

Chapter 4 shows how to compute network reliability exactly when the number of non-K-vertices is moderate. A truncated version of this technique is used to obtain a lower bound on reliability when the number of non-K-vertices is too large. We demonstrate that this lower bound will usually be "good."

Chapter 5 describes an algorithm to compute $R_{sK}(D)$ or find a lower bound on $R_{sK}(D)$, and its implementation. The program is coded in FORTRAN with one main routine and 16 subroutines totaling about 1600 lines of code.

Chapter 6 gives computational experience for the algorithm run on an IBM 3033AP system /370 with the FORTRAN H (Extended) compiler.

Chapter 7 is a conclusion and gives suggestions for further research.

II. DEFINITIONS AND NETWORK RELIABILITY THEORY

In this chapter, we give the definitions and notation used throughout this thesis, and introduce basic network reliability theory.

A. DEFINITIONS AND NOTATION

A directed network $D=(V,E)$ comprises two finite sets: V is the set of **vertices** and E is the set of **edges**. Each edge $e \in E$ corresponds to an ordered pair of vertices in the directed network. That is, $e=(u,v)$ where $u,v \in V$. The vertex u is called the **tail vertex** of edge e and vertex v is called the **head vertex** of edge e. For any two vertices u and v in D, if there exists an edge $e=(u,v)$, then u and v are said to be **adjacent** and edge e is **incident into** v and **incident out of** of u. The **indegree** of a vertex v, denoted **indeg(v)**, is the number of edges incident into v and the **outdegree** of a vertex v, denoted **outdeg(v)**, is the number of edges incident out of v. Two vertices are **connected** (or "communicate") if there exists a sequence of vertices and edges of the form $v_0, (v_0,v_1), v_1, (v_1,v_2),\ldots,(v_{m-1},v_m), v_m$. This sequence is a **path** of length m. If the $v_0=v_m$, the path is a **cycle**.

Consider a undirected graph $G(V,E)$ formed by ignoring the ordering of the edges in a directed graph $D(V,E)$. G is called the **underlying graph** of D. D is **connected** if its underlying graph G is connected. G is connected if there is an undirected path between all pairs of vertices. The **vertex connectivity** of D is the minimum number of vertices which must be deleted from G (along with adjacent edges, of course) in order to disconnect G or to create a trivial graph with only one vertex. D is **separable** if its vertex connectivity is one. Otherwise D is

nonseparable. In a separable graph, a vertex whose deletion disconnects the graph is called a **cut-vertex**.

Merging a vertex v into another vertex w means removing all edges between v and w and forming a single vertex v' by coalescing v and w such that any edge incident into (out of) v or w is now incident into (out of) v'. Merging v and w is sometimes also referred to as "identifying" v with w.

The SKT reliability of a directed network D, denoted $R_{sK}(D)$, is the probability that all K-vertices in D are connected by path of working edges and vertices from source vertex s. The probability that edge e_i works is $p_i = 1-q_i$. For now, we assume that all vertices work with probability one.

B. NETWORK THEORY AND SURVEY

1. NP-Completeness

We briefly discuss the complexity of the SKT reliability problem in terms of the theory of NP-completeness and in terms of practical computation. More detail can be found in Garey and Johnson [1979], Ball and Provan[1981].

Algorithms to solve various problems can be broadly classified into two categories, polynomial-time and exponential-time algorithms. An algorithm is a polynomial-time algorithm if for a problem of size n, its running time is bounded by a polynomial in n. Any algorithm that is not a polynomial-time algorithm is an exponential-time algorithm. In combinatorics, a decision problem P is said to belong to the class NP if given a tentative solution, the validity of the solution can be checked in polynomial-time. P is "NP-complete" if it is equivalent to the satisfiability problem of a conjunctive normal form boolean expression. A problem is NP-hard if it is at least as hard as an NP-complete problem.

A problem is #P-complete (number-P complete) if its solution is equivalent to counting (not enumerating) the number of solutions to an NP-complete problem. Any #P-complete problem is NP-hard.

2. Problem Complexity

Ball and Provan [1981] show that the ST reliability problem in an acyclic directed network is a #P-complete problem. This implies that the SKT problem is also #P-complete since problem instances restricted to $|K|=2$ are #P-complete. Even finding an ε-approximation to SKT reliability is an #P-complete problem.

In practice, these complexity results imply that a general problem of large size cannot be solved analytically in a reasonable amount of time. In fact, the time taken grows exponentially with size of the network. As will be seen, however, practical problems of fairly large size can be handled with the techniques developed here. Furthermore, even larger problems can be handled if one is willing to accept a "good" lower bound on exact reliability.

3. Reduction

Many reduction schemes have been developed to help solve various network reliability problems. These reductions reduce the size of the network in polynomial time while preserving reliability, i.e., they create a new, smaller, network G' from the original network G (G may directed or undirected and any number of definitions of reliability may be used.) such that $R(G)=M\ R(G')$ where M is a known constant factor. In this way, a polynomial amount of work is expended in order to save, it is hoped, an exponential amount of work during later, more general computations. In some special, but often practical cases, it is possible to completely reduce a network and effectively compute its reliability in polynomial time.

The reductions describe below give a flavor for the different types of reductions which have been developed. Parallel and series reductions are well-known schemes so they will not be discussed. The reductions described assume

that vertices are completely reliable. However, most can be extended to handle unreliable vertices.

(1) Polygon-To-Chain Reductions: A chain C in an undirected graph G is defined as a alternating sequence of vertices and edges beginning and ending with vertices such that all internal vertices have degree 2. Two parallel chains form a polygon and this polygon can be replaced by a chain in a reduction that preserves reliability in the K-terminal reliability problem, the analog to the SKT reliability problem in an undirected network. This reduction and its implications are developed by Wood [1982].

(2) 2-Neighbor Vertex Reduction: For the SKT reliability problem, any vertex u that has exactly two neighbors v and w can be eliminated from the network if $u \notin K$ or $\{u,v,w\} \in K$ and $u \neq s$.

(3) Parallel-Chain Reduction: Suppose E_{ij} is the set of all edges in a directed network D whose endpoints are v_i and v_j. In D, a chain C_i is an alternating sequence of vertices and non-empty sets of edges where the internal vertices are 2-neighbor vertices in D. Two chains C_1 and C_2 with identical endpoints are parallel chains. Parallel chains can be reduced to a single chain in the SKT reliability problem. Reductions (2) and (3) are developed by Agrawal [1982].

4. Decomposition

Sometimes it is possible to divide a given graph into a number of subgraphs, such that each subgraph can be analyzed separately and the results combined to obtain the reliability of the whole graph. We briefly describe some of these methods based on vertex connectivity.

(1) 2-Connected Graph Decomposition: Let G be a 2-connected graph with a pair of vertices $\{u,v\}$ whose deletion disconnects the graph. These vertices are called a "separating pair." This separating pair partitions G into two subgraphs G_1 and G_2 such that $G_1 \cup G_2 = G$, $G_1 \cap G_2 = \{u,v\}$, $E_1 \neq \phi$ and $E_2 \neq \phi$. It has been shown by Wood [1982] that, in the K-terminal reliability problem, each subgraph defines a chain

through a set of reliability computations on the subgraphs. The subgraphs may be replaced by the chains so as to preserve reliability. Agrawal [1982] has proven the analogous decomposition scheme for the SKT reliability problem on directed networks. (2) Strong-connected Decomposition: A strong component of a directed graph D is a maximal subgraph of D such that there exists a directed path from every vertex to every other vertex in the subgraph. When computing $R_{sK}(D)$, each strong component can be considered separately. Computing each component $R_{sK}(D_i)$ and multipling $R_{sK}(D_i)$ is whole graph reliability $R_{sK}(D)$. This decomposition has been developed by Shogan [1973] and Agrawal [1982].

Other decomposition schemes exist. Decomposition in 3-connected graphs is often possible, Rosenthal [1974]. Decomposition in 1-connected graphs will be discussed in chapter 3.

III. REDUCTION AND DECOMPOSITION

Any algorithm to solve a general network reliability problem has a running time which is exponential in the size of problem. If by some method, the size of the problem can be reduced, immense computational savings can be made. Therefore, in this chapter, we introduce reduction and decomposition schemes which will often reduce the size of the problem while always preserving reliability. Some of these techniques are specific to the SKT reliability problem in acyclic directed networks while others are of more general applicability.

Reduction schemes reduce a parameter of the size of the problem, usually edges or vertices, in order to decrease computational complexity while preserving network reliability. On the other hand, decomposition schemes split the whole graph into subgraphs, computes subgraphs reliabilities separately and then combines these reliabilities in some way to compute overall network reliability.

A. REDUCTIONS

In this section, we discuss various reduction schemes which reduce the size of D and, consequently, reduce the complexity of computing $R_{sK}(D)$. Under any reduction, reliability of the network remains invariant up to a known, constant factor, i.e., $R_{sK}(D) = M \cdot R_{sK'}(D')$ where M is the known factor, K' is the new set of K-vertices, and D' is the reduced network. In the following illustrations, all vertices are represented as circles with K-vertices being darkened or shaded.

The parallel and series reductions are standard, i.e., well-known and widely used. The three reduction schemes,

"indegree-one K-vertex contraction", "non-K-vertex deletion" and "neck vertex to K-vertex reduction" are newly developed reductions.

1. Parallel Edges Reduction

In a directed network, two parallel edges $e_a=(u,v)$ and $e_b=(u,v)$ may be replaced a single edge $e_c=(u,v)$ while preserving reliability if $p_c = p_a+p_b-p_ap_b$ and $M=1$.

2. Series Edges Reduction

In a directed network, a pair of edges $e_a=(u,v)$ and $e_b=(v,w)$ such that indeg(v)=outdeg(v)=1 are series edges. If $v \notin K$, edges e_a and e_b can be replaced by single edge $e_c=(u,w)$ while preserving reliability if $p_c = p_ap_b$ and $M=1$.

3. Indegree-One K-vertex Contraction

Let $e_i=(u,v)$ be an edge in D such that $v \in K$ and indeg(v)=1. In the SKT reliability problem, communication from s to vertex $v \in K$ can only only occur via edge e_i so this edge must work if the network is to work. Consequently, edge e_i can be contracted such that $R_{sk}(D) = M \cdot R_{sk}(D')$ and $M=p_i$

where $D' = \{ V - u - v + w, E - e_i \}$

$$w = u \cup v$$

$$K' = \begin{cases} K-v+w, \text{ if } u \notin K \\ K-v-u+w, \text{ if } u \in K. \end{cases}$$

The validity of this reduction is easily shown. Let $e_i(u,v)$ be the edge described above in the acyclic network D. Let F_i denote the event that e_i is working, and let $\bar{F}_i$ denote the complementary event. Since $R_{sK}(D)$ is just a probability, the rules of conditional probability can be applied to obtain

$$R_{sK}(D) = p_i R_{sK}(D \mid F_i) + q_i R_{sK}(D \mid \bar{F}_i)$$

$$= p_i R_{sK}(D \mid F_i) \quad \text{since } R_{sK}(D \mid \bar{F}_i)=0$$

$= p_i R_{sK}(D')$

since no new paths are created by contraction.

4. Non-K-Vertex Deletion

In the SKT reliability problem, an outdegree zero non-K-vertex can be deleted from the network without changing network reliability since such a vertex is irrelevant. In this case $R_{sK}(D) = M\ R_{sK}(D-v)$ and $M=1$.

5. Neck Vertex to K-vertex Reduction

Suppose u and v are two vertices in D such that $u \in K$, $v \notin K$ and all paths from s to u include v. v is called a ***neck*** vertex. It may or may not be a cut-vertex. Since s must communicate with v before it communicates with u, $R_{sK}(D) = M\ R_{s(K+v)}(D)$ where $M=1$. Thus, the neck vertex to K-vertex reduction simply changes K to $K+v$. See Figures 3.1a and 3.1b. (All figures are given in Appendix A.)

The neck vertex to K-vertex is termed a "reduction" because it reduces the parameter $|V-K|$ in D. As will be seen in chapter 4, the general algorithm for computing $R_{sK}(D)$ runs in time proportional to $2^{|V-K|}$ so this is a reduction according to our definition. Furthermore, the neck vertex to K-vertex reduction will often allow other reductions or decomposition to take place.

6. Other Reductions

Numerous other reductions exist as exemplified in chapter 2, section B3. However, none of these reductions has been implemented because of their complexity, and therefore, no other reductions will be described here.

B. DECOMPOSITION

Sometimes it is possible to decompose a given network D into a number of subnetworks. Each subnetwork can then be analyzed separately and the results combined in some way to obtain overall network reliability.

Decomposition can result in substantial savings because, in general, the time to analyze the whole system is exponential in size of the system. Effort needed to compute $R_{sK}(D)$ using decomposition is the sum of efforts needed for each subproblem plus the effort, normally polynomial in the size of D, in performing the decomposition and recombining the result. Without decomposition, the necessary work is roughly the product of the effort needed to solve the individual problems.

1. *Cut-Vertex Decomposition*

A connected, undirected, network $G(V,E)$ has a cut-vertex v if there exist components (subgraphs) C_1 and C_2 such that $C_1 \cap C_2 = v$ and $C_1 \cup C_2 = G$. Any communication from one component to the other must pass through v. In this case we also say that v separates C_1 and C_2. A network which has a cut-vertex is "separable" and one which has none is called "nonseparable." A directed network D is separable if its underlying graph G is separable.

Let D have nonseparable components $C_1, C_2, \ldots, C_n$. Any component C_i which has only one cut-vertex is called a ***pendant*** component. If a pendant component, C_i, contains no K-vertices except possibly its cut-vertex, then it is completely irrelevant with respect to reliability since none of its edges are in any path from the source to a K-vertex. Thus, this component, excluding its cut-vertex, can be deleted from the network while preserving network reliability in the SKT reliability problem. This type of deletion may be recursively applied until only relevant components remain.

Cut-vertices are not efficiently found in a directed network if each edge is represented in one direction, e.g., if all edges incident into each vertex are stored contiguously. However, if edges are stored both by tail vertex and head vertex, then this is exactly the representation of the underlying undirected network. Thus, an efficient algorithm for finding cut-vertices in an undirected graph may be used to find cut-vertices in D. A slight modification to the

DFS (depth-first search) algorithm due to Tarjan [1972] can find all cut-vertices in $O(|\mathcal{E}|)$ time.

Once all cut-vertices have been found and all irrelevant components removed, any cut-vertices remaining which are not K-vertices are neck vertices and may be added to the set K. It is fairly obvious that each component must work by itself if D is to work. Since edge reliabilities are independent it follows that $R_{sK}(D) = R_{sK}(D_1)\, R_{sK}(D_2) \cdots R_{sK}(D_m)$ where the D_i are the nonseparable, relevant components of D. This is identically equal to a network which has the same components as D but whose only cut-vertex is the source. A more rigorous demonstration of this identity can be made by applying the decomposition scheme described next, "moving edges to source," to all edges incident out of cut-vertices. Figure 3.2 demonstrates the results of cut-vertex decomposition applied to a network D with five components, two of which are irrelevant. In this case, $R_{sK}(D) = R_{sK}(D_1) \cdot R_{sK}(D_2) \cdot R_{sK}(D_3)$.

2. Moving Edges to Source

In this section, we discuss a technique which does not fit neatly under the rubric of decomposition or reduction since it only moves edges from one place to another in the network. However, by moving certain edge tails to the source s, new parallel reductions, series reductions or contractions may be made possible along with cut-vertex decompositions. Consider an edge $e = (u, v)$ such that $u \in K$. Edge e can be made incident out of any other K-vertex $u' \in K$ without affecting reliability as long as no cycles are created. A simple conditioning argument can be used to show this, Agrawal [1982]. The source is the obvious candidate to receive the edge tails since it has no edges incident into it and therefore no cycles can possibly be created. Figure 3.3 shows an example where this operation creates a cut-vertex and allows a parallel reduction.

IV. COMPUTING SKT RELIABILITY IN AN ACYCLIC DIRECTED NETWORK

This chapter gives a general method for computing SKT reliability in an acyclic directed network. The method is based on repeated computation of SAT reliability of certain subgraphs of D and, consequently, we first discuss SAT reliability. After deriving a method for computing SKT reliability, a simple lower bound becomes apparent and several heuristic arguments are given indicating that the bound's accuracy should be good. Actual computational examples are reserved for the next chapter.

A. SOURCE-TO-ALL TERMINAL RELIABILITY

Let $D(V,E)$ be an acyclic directed network and let $q_i=1-p_i$ be the failure probability for any edge $e_i \in E$ and let E_j be the set of edges incident into v_j. Also, let the vertices be numbered such that $\{v_1,v_2,\cdots,v_n\}$ is an an acyclic ordering of V. Then, SAT reliability may be expressed as

$$R_{sV}(D) = P(\, s \text{ communicates with all } v \in V-v_n \text{ and at least one edge into } v_n \text{ works }) \tag{4.1}$$

since there no edges incident out of v_n by the acyclic ordering

$$= R_{s(V-v_n)}(D-v_n)\left[1-\prod_{e_j\in E_n} q_j\right] \quad \text{by independence}$$

$$= R_{s(V-v_{n-1}-v_n)}(D-v_{n-1}-v_n)\left[1-\prod_{e_j\in E_{n-1}} q_j\right]\left[1-\prod_{e_j\in E_n} q_j\right]$$

$$= \cdots$$

$$= \prod_{v_i\in V}\left[1-\prod_{e_j\in E_i} q_j\right]$$

Another way to see this is to use the fundamental topological property that $R_{sV}(D)$ is unchanged if any edge (u,v) with $u \in K$ is replaced by an edge (w,v) with the same reliability and $w \in K$ has a lower number than u in an acyclic ordering of D. Consequently, in the SAT problem, we may move all edges so that they come out of the source just like the "moving edge to source" operation in section 3.2. Equation (4.1) is then trivially true for a network where all edges come out of the source. (See Figure 4.1). The complexity of computing $R_{sV}(D)$ is easily seen to be $O(|E|)$ if all edges are with common head vertices are stored contiguously.

B. SOURCE-TO-K-TERMINAL RELIABILITY

Now we consider the SKT reliability problem in D. Let $D(V,E)$ be a acyclic directed network with $K \subseteq V$. By simple state-space partitioning,

$$
\begin{aligned}
R_{sk}(D) = {} & P(\, s \text{ communicates with all } v \in V) \\
& + \sum_{v_i \notin K} P(\, s \text{ communicates with all } v \in V - v_i \text{ but not with } v_i) \\
& + \sum_{v_i, v_j \notin K} P(\, s \text{ communicates with all } v \in V - v_i - v_j \text{ but not with } v_i \text{ or } v_j) \\
& + \ldots \\
& + P(\, s \text{ communicates with all } v \in K \text{ but not with any } v \in V - K)
\end{aligned}
$$

But,

$P(\, s$ communicates with all $v \in V\,) = R_{sV}(D)$

$P(\, s$ communicates with all $v \in V - v_i$ but not with $v_i\,) = R_{s(V-v_i)}(D)$. $P(\,$ No edge into v_i works $)$

$P(\, s$ communicates with all $v \in V - v_i - v_j) = R_{s(V-v_i-v_j)}(D)$. $P(\,$ No edge into v_i or v_j works except an edge $e = (v_i, v_j)$ or $e = (v_j, v_i))$

etc.,

In general, then,

$$R_{sK}(D) = R_{sV}(D) + \sum_{v_i \notin K} \left[\left(\prod_{e_j \in E_i} q_i \right) R_{sV}(D - v_i) \right] \tag{4.2}$$

$$+ \sum_{v_i, v_j \notin K} \left[\left(\prod_{e_j \in E_{ij}} q_j \right) R_{sV}(D - v_i - v_j) \right]$$

$$+ \sum_{v_i, v_j, v_k \notin K} \left[\left(\prod_{e_j \in E_{ijk}} q_j \right) R_{sv}(D - v_i - v_j - v_k) \right]$$

$$+ \ldots$$

where E_i is the set of edges going into v_i, E_{ij} is the set of all edges going into v_i or v_j except edges going between v_i and v_j, E_{ijk} is the set of all edges going into v_i, v_j or v_k except edges going between pairs of vertices in $\{v_i, v_j, v_k\}$, etc.

The complexity of computing $R_{sK}(D)$ is, via the above formula, exponential in the number of non-K-vertices. The total number of additive terms in equation (4.2) is $\sum_{i=0}^{n_1} C_i^{|V-K|} = 2^{|V-K|}$ where C_i^n is the number of ways to choose i elements out of n elements. Computing SKT reliability thus requires $O(|E|2^{|V-K|})$ time using equation (4.2). Of course, if the number of non-K-vertices is fixed as $|E|$ changes, computation is effectively $O(|E|)$.

C. A LOWER BOUND ON SKT RELIABILITY

Since computation increases exponentially in the worst case, we may be satisfied with an approximation to SKT reliability which requires less time to compute than exact reliability. Since all terms in equation (4.2) are positive, any subset of these terms provides an approximation to SKT reliability which is a lower bound. Also, the product formed by lower bounds on independent components of a network yields a lower bound on overall network reliability. The subset of terms we choose to form this lower bound consists of terms enumerated in the order given in equation (4.2) until some specified computational limit is reached. The motivation for this scheme being fairly accurate is given below.

Suppose $p_i = p$ for all $e_i \in E$ and p is close to 1. Consider the first few terms of Equation (4.2).

$$R_{sV}(D) = \prod_{v_i \in V} \left[1 - q^{|E_i|}\right] \sim 1.0$$

$$\left(\prod_{e_j \in E_i} q_j\right) R_{s(V-v_i)}(D-v_i) = q^{|E_i|} \prod_{v_i \in V-v_i} \left[1 - q^{|E_i|}\right] \sim q^{|E_i|}$$

$$\left(\prod_{e_j \in E_{ij}} q_j\right) R_{s(V-v_i-v_j)}(D-v_i-v_j) \leq q^{|E_i|+|E_j|-1} \prod_{v_i \in V-v_i-v_j} \left[1 - q^{|E_i|}\right] \sim q^{|E_i|+|E_j|+1}$$

Thus, the contribution of the terms to the total reliability sum diminishes rapidly unless there are a significant number of edges connecting non-K-vertices.

Another reason that the lower bound should often be good is that terms with many vertices deleted may be identically zero. For example, consider an ST reliability problem where the shortest path from s to t has l edges in it. This path must contain $l-1$ non-K-vertices and so $R_{s(V-v_i-v_j-\ldots-v_k)}(D-v_i-v_j-\ldots-v_k) \equiv 0$ when $|V-\{v_i,v_j,\ldots,v_k,s,t\}| < l-1$. Of course, other terms may be identically zero, too. For example, in any K-terminal problem, the term corresponding to $D-v_i-v_j$ will be identically zero if all paths from s to some K-vertex v must include either v_i or v_j.

Computational experience in Chapter V lends credence to the accuracy of this lower bound.

V. ALGORITHM AND DATA STRUCTURE

The algorithm presented here for reliability evaluation of directed acyclic networks is based on the theory developed in chapter 3 and chapter 4. The program is intended for general use so it was coded in the widely available language, FORTRAN. This language is available on most computer systems and is still one of the most popular languages among operations research analysts.

A. ALGORITHM

The objective in this section is to develop an efficient algorithm for computing SKT reliability in a directed acyclic network. Initially, polynomial-time reductions and decomposition are used to reduce network parameters and decompose the the network into subnetworks, if possible. If reliability is not completely computed via reliability-preserving reductions, the general SKT reliability expression is computed as described in chapter 4. If a specified CPU time limit is reached for any component, the computation is truncated and a lower bound on reliability is produced.

There is a one main algorithm and numerous subroutine algorithms in the program. We discuss the main program and several important subroutine algorithms for understanding. The structure of the main program is based on a step by step approach which avoids redundant work. Comments will be given in curly brackets.

Main

Input: A directed acyclic network D with vertex set V, $|V|\geq 2$, edge set E, $|E|\geq 2$, and set $K\subseteq V$, $|K|\geq 2$. Edge reliability p_i for each edge e_i in E.

Output: $R_{sK}(D)$, exact reliability, if CPU time less than specified amount, otherwise a lower bound.

(1) {Initialize} M = 1.0, bound=.false.

(2) Read all edge data and create network data structure

(3) {Change neck vertices to K-vertices} K = **Neck** (D,K)

(4) {Perform reductions for whole network} **Reduct** (D,K,M)

(5) If remaining vertices $|V| \leq 1$, go to end.

(6) $\{D_1,D_2,\ldots,D_t\}$= **Decomp**(D)

(7) For each subnetwork D_i

(a) {Initialize} lowbnd=.false., set time = 0.

(b) {Move edge tails to source} For each edge $e=(v_i,v_j)$ with $v_i\in K$ {move v_i to s} **Move**(e)

(c) Construct the data structure for subnetwork D_i

(d) {Reduce subnetwork} **Reduct** (D_i,K_i,M)

(e) { Initialize } sumpro=0

(f) { Find SAT reliability } sumpro = **Rsa**(D_i),

(g) Count the remaining non-K-vertices for D_i, i.e., if vertex number > 0, count=count+1

(h) If count $\geq$ 1, { find the terms in $\sum\left[(\prod q_i)R_{sa}(D-v_i,\ldots,-v_l)\right]$ }

1) {Find the next combination of non-K-vertices}

$\{v_{j_1},v_{j_2},\ldots,v_{j_i}\}$ = **Comb**(j,i)

2) { Find $\prod q_i$ for $E_{j_1 j_2 \cdots j_i}$ } M' = **Multi**$(v_{j_1},v_{j_2},\ldots,v_{j_i})$

3) sumpro=sumpro + $M'\cdot$ **Rsa**$(D-v_{j_1}-v_{j_2}-,\ldots,-v_{j_i})$

4) If elapsed CPU time $\geq$ time limit then lowbnd=.true. and

bound=.true.

5) If lowbnd=.false. and all combination have not been enumerated then go to (8.h.1)

(i) $M = M \times$ sumpro

(8) If bound=.false. print "Exact reliability is" M, otherwise print "Lower bound on reliability is" M

End Main

function Neck (D,K)

Input: D,K

Output: New set of K-vertices

(This routine finds all neck vertices and changes them to K-vertices)

(1) Put all $v_i \notin K$ into Q_n

(2) $k = |K|$

(3) While $Q_n \neq \phi$

(a) Remove v from Q_n

(b) Search all K-vertices in $D - v_i$ using Depth-First Search

(c) k'= number of K-vertices reached

(c) If $k' < k$ then $K = K \cup \{v\}$

(3) Return (K).

End of Neck

subroutine Reduct(D,K,M)

Input: D,K,M

Output: Reduced network D with modified K and M

(This routine performs all non-K-vertex reductions, series-parallel reductions and indegree-one K-vertex contractions)

(1) Construct $Q_d = \{ v \in V-K \mid \text{outdeg}(v)=0 \}$

(2) While $Q_d \neq \phi$

(a) For each vertex v_j incident to v_i, if outdeg(v_j)=1 and $v_j \in V-K$ then add v_j to Q_d.

(b) **Nonkre** (v_i)

(3) Construct Q_s = { $v_i \in V-K$ | indeg(v_i)= outdeg(v_i)=1 } and Q_c = { $v_i \in K$ | indeg(v_i)=1 }

(4) For each v_i

(a) Check all outgoing edges, if $e(v_i,v_j), e(v_i,v_j)$ are found then **Parall**(e)

(b) If parallel reductions creates new series or contraction vertices, put these into Q_s or Q_c, respectively.

(5) While $Q_s \neq \phi$ and $Q_c \neq \phi$

(a) If $Q_s \neq \phi$ **Series**($Q_s(i)$)

1) If series reductions create new paraliel edges, do parallel reduction. **Parall**(e)

2) If parallel reductions create new series or contraction vertices, put these into Q_s or Q_c, respectively.

(b) If $Q_c \neq \phi$ **Contra**($Q_c(i)$)

1) If the contraction creates a new contraction vertex, put these into Q_c

2) If the contraction creates new parallel edges, do parallel reduction **Parall**(e)

3) If parallel reductions create new series or contraction vertices, put these into Q_s or Q_c, respectively

(6) Return (D,K,M)

End Reduct

subroutine Decomp(D)

Input: D

Output: Subnetworks $D_1, D_2, \ldots, D_m$

(1) While not all cut-vertices have been found

(a) Using DFS, find pendant component D_i with cut-vertex v_i

(b) If D_i has no K-vertices except possibly v_i then delete D_i

(2) Return $(D_1, D_2, \ldots, D_m)$

End Decomp

After reading in the data and creating the network data structure, the algorithm begins by changing any neck vertices to K-vertices by a call to **Neck** at step (3). Next, a call to **Reduct** is made at step (4). **Reduct** first removes all outdegree zero non-K-vertices. Through the queue mechanism, it is ensured that all initial outdegree zero non-K-vertices are deleted along with any which are created as the reduction proceeds. A similar method is used for the other reductions schemes.

The other reductions, indegree-one-K-vertex contractions and parallel and series reductions, may recursively create new reducible vertices or edges. The newly created reducible vertices are put into a queue instead of immediately being reduced to avoid redundant work. Parallel edges are reduced immediately, however. The network of Figure 5.1 is completely reduced by reductions alone and illustrates how a reduction can create other reducible edges or vertices. A call to **Reduct** would result in the set of reductions given below. This network would be completely reduced and its reliability completely calculated by these reductions.

(1) Reduce parallel edges e_3, e_4 to e_3

(2) contract vertex v_6

(3) series vertex v_4

(4) contract vertex v_2

(5) reduce parallel edges e_3, e_5 to e_3

(6) series vertex v_7

(6) series vertex v_7

(7) reduce parallel edges e_{10}, e_{11} to e_{10}

(8) series vertex v_8

(9) reduce parallel edges e_{10}, e_{12} to e_{10}

(10) contract vertex v_9

(11) reduce parallel edges e_2, e_8 to e_2

(12) contract vertex v_3

(13) reduce parallel edges e_3, e_7 to e_3

(14) contract vertex v_5

After reductions are performed, a network will decompose into subcomponents if it contains any cut-vertices. This is done at step (6) of **Main** by a call to **Decomp**. The rest of the calculations are performed iteratively for each component D_i under step (7). First, all edges with tail-vertices in K are moved to the source. This may produce additional reductions, so another call to **Reduct** is made for the subcomponent. Then, the terms corresponding to Equation 4.2 (for that component) are produced until all such terms have been enumerated or the CPU time limit is reached. Note that this CPU limit is based on the CPU time used in calculating subcomponent reliability and is not based on total CPU time.

If the CPU time limit is reached for any of the subcomponents, the final value obtained by the algorithm is a lower bound on reliability. Otherwise, the value obtained is exact reliability. Choosing a CPU time limit will depend on computer

facilities available and the network under analysis. Figure 6.5 shows the reliability increment as a function of CPU time. The lower bound obtained using a reasonable CPU time limit will not differ much from the exact reliability for this large scale network example

B. DATA STRUCTURE

There a many ways to represent a network on a digital computer. The most straightforward is to use an adjacency matrix: If vertices v_i and v_j are adjacent, M(i,j)=1, and M(i,j)=0 otherwise. Matrices of the above type are, in practice, extremely sparse and this representation is very inefficient in terms of space. It is often inefficient in terms of execution time, too, particularly when one is interested in operations that require retrieving information from all vertices adjacent to a particular vertex, e.g., **Neck, Decomp, Reduct,** and **Rsa**. In this situation, since most of the elements of M(i,j) are null, the algorithm will spend a great amount of time retrieving and comparing zero values. For a static network, an efficient representation is a packed matrix. In our algorithm, the network is dynamic, since it is reduced and decomposed many times during execution. Therefore, a multi-linked data structure (multiple linked lists with links to other data structures such as arrays) is used to represent the network. Every vertex has a linked list of adjacent vertices, which, besides informing which vertices are adjacent to it, also tells whether or not the vertex belongs to set K. Two such lists are kept for each vertex, an "adjacent out" list and an "adjacent into" list. Of course, each adjacency entry corresponds to an edge. So, in addition, there are pointers indicating the addresses where information about edges is kept.

Figure 5.3 illustrates this data structure. The vertex v_1 only can send communication to vertices v_2, and v_3, and vertex v_2 can receive the

communication from vertex v_1 and send the communication to vertices v_3 and v_4, and so on. This structure is a very efficient data structure for dynamic networks.

C. ALGORITHMIC COMPLEXITY

It is important to determine the complexity of the various parts of the algorithm so that overall complexity can be understood. In this section, we analyze the worst-case complexity of the subroutines, as written, and thus, find the complexity of the overall algorithm. Not all of the subroutines have been written as efficiently as possible because of the difficulty in programming such routines and because of limited time. Furthermore, for problems of moderate size, it is unlikely that much efficiency would be gained using more sophisticated routines, since, in practice, most of the routines written seem to operate more efficiently than the worst-case analysis indicates.

There are three different reduction schemes included in the algorithm: non-K-vertex deletion, indegree-one K-vertex contraction and series and parallel reductions. However, the last three of these interact directly.

The non-K-vertex deletion, steps (1) and (2) in REDUCT, requires $O(|E|)$ operations in the worst case. For example, consider a complete acyclic graph which has only one K-vertex, the source. Here, one vertex at a time would be deleted from the network, but, in the process, every edge would be examined exactly once until a single isolated vertex remained. The initial queue-building is of no consequence since it is $O(|V|)$.

Steps (3) through (5) constitute the other three reductions. Queue-building again may be disregarded since it is at an $O(|V|)$ operation. Initial parallel reductions may require $O(|E|)$ operations using a bucket sort technique to check for edges with common end vertices. The central step of the algorithm, step (5), requires $O(|V|^2)$ time. This is true since (1) at most $|V|$ series reductions or

contractions can ever occur (2) each such reduction requires one $O(|V|)$ check for a newly created parallel edge, and (3) once identified, actually carrying out the parallel reduction requires only constant time. Overall then, REDUCT requires $O(|V|^2)$ time.

There are two different decomposition schemes used, cut-vertex decomposition and neck-vertex decomposition. The complexity of the cut-vertex algorithm is $O(|E|)$ since it is based on the $O(|E|)$ depth-first search. The complexity of the neck-vertex algorithm will be $O(|V-K||E|)$ since finding neck-vertices is based on repeated $O(|E|)$ searches on $D-v$ for each non-K-vertex v. In the worst case then, this decomposition requires $O(|V||E|)$ time. However, if we fix the number of non-K-vertices as described in chapter 4, then the complexity of neck-vertex algorithm will effectively be $O(|E|)$.

It follows from the above discussion that the total complexity of the reduction and decomposition algorithms is $O(|V||E|)$ normally, and $O(|V|^2)$ when non-K-vertices are fixed.

After reductions, computing SKT reliability requires $O(|E|2^{|V-K|})$ time. This is obviously true since $2^{|V-K|}$ combinations of non-K-vertices must be produced and an $O(|E|)$ SAT computation made for each combination. When the number of non-K-vertices is fixed, the computation becomes $O(|E|)$. The entire algorithm therefore requires $O(|E|2^{|V-K|})$ time in general but only $O(|V|^2)$ time with non-K-vertices fixed.

VI. COMPUTATIONAL EXPERIENCE

In order to test the efficiency of the SKT reliability algorithm, we used two kinds of networks, "complete" acyclic networks, and acyclic "street" networks. A complete acyclic network is a graph $D(V,E)$ such that $V=\{v_1,v_2,\cdots,v_n\}$ and $e=(v_i,v_j)\in E$ if and only if $i<j$. Any such network with n vertices has exactly $\frac{n(n-1)}{2}$ edges. A network is called an $m\times n$ "street" network if (1) its vertices can be laid out in an m rows by n columns rectangular grid, (2) for all vertices except those in the last column there is an edge incident out each vertex directed to the right, and (3) for all vertices except those in the first row, there is an edge incident out of each vertex directed upward. When $n=m$ case the street network is called "square" (See Figure 6.1).

In the complete network we assigned v_1 to be the source and fixed v_n as a K-vertex. This insures that all edges and vertices are relevant. Other K-vertices were chosen randomly as required. In the street network, all corner vertices were fixed as K-vertices, the lower left-hand corner vertex assigned as the source and all other K-vertices chosen randomly.

A. SOURCE-TO-ALL-TERMINAL RELIABILITY COMPUTATION

As described in chapter 4, the complexity of computing SAT reliability is $O(|E|)$. Since repeated computation of SAT reliability is the key to computing general SKT reliability, it is interesting to see how much CPU time is required for a single SAT computation. Figure 6.2 shows the CPU time used as a function of the number of vertices in the two types of test networks. $|V|$ is increased from 15 to 60 for the complete network and from 9 to 144 for the square street network. Of course, as a function of $|V|$, $|E|$ is quadratic for the complete network ranging

from 105 to 3160. $|E|$ increases linearly as a function of $|V|$ for the street network ranging from 12 to 284. From the figure, it is clear that SAT reliability can be computed very rapidly.

B. SOURCE-TO-K-TERMINAL COMPUTATION

Here we investigate the work required by the algorithm for computing general SKT reliability and the efficiency gained by using reductions and decomposition. Initial reductions and decomposition require $O(|V||E|)$ time. After reductions and decomposition, the complexity of computing SKT reliability is $O(\max_{i \in C}\{|2^{|V_i - K_i|}||E_i|\})$ where C is the set of separable components of D and where $V_i - K_i$ is the set of non-K-vertices in component i. Consequently, we expect an exponential growth in the worst case for SKT computations. Figure 6.3 shows the growth in CPU time as the number of non-K-vertices is increased, with and without reductions and decomposition.

For the complete network, the number of non-K-vertices ranged from 3 to 14 out of a total 20 vertices. The reductions and decomposition do not reduce computation much here since the complete network is so densely connected. On the other hand, computation times are significantly reduced when reductions and decomposition are used in the street network, primarily because many non-K-vertices are changed to K-vertices. Notice that the computation time required by the street network without reductions and decomposition is almost the same as for the complete network. This shows how complexity is dependent mostly on the number of non-K-vertices.

C. COMPUTATION WITH NON-K-VERTICES FIXED

If the number of non-K-vertices is fixed, i.e. $|V-K|=c$ for some constant c, then the complexity of computing SKT reliability is $O(|V|^2)+O(c|E|) = O(2^c|E|)$ which is $O(|V|^2)$. Thus, the complexity is polynomial instead of exponential. Of

course, if reductions and decomposition are not used, the complexity becomes $O(|E|)$ which is better than $O(|V|^2)$. However, in practice, the reductions and decomposition are very efficient and do not seem to increase computation time. Figure 6.4 shows the increase in computation for both test networks as $|V|$ is increased.

D. LOWER BOUND COMPUTATION

As described in chapters 4 and 5, a lower bound on SKT reliability is obtained when the normal SKT reliability computation is truncated. Two heuristic arguments were given in chapter 4 indicating that this lower bound should be fairly good. Figure 6.5 shows, for actual computations, how the lower bound approaches the exact network reliability very quickly. The test networks were assigned different values of p_i to show that the accuracy of the bound is not much affected by p_i. It appears that a CPU time limit can be used successfully for truncating SKT computation to obtain a good lower bound when exact computation is not feasible.

VII. CONCLUSION

This thesis has explored the source-to-K-terminal reliability analysis problem in acyclic directed networks. We have shown that the polynomial-time reductions and decompositions reduce an exponentially complex problem to a polynomially complex problem under certain conditions. Even when these conditions do not hold, surprisingly large networks can be analyzed. The theoretical and computational results obtained here can be used by researchers in analyzing the reliability of directed acyclic networks and, in some cases, for the partial reliability analysis of general directed networks.

We assumed that all vertices work perfectly and that edge reliabilities are constant while the network system is alive. These assumptions may not be acceptable in a real-world situation. It may be necessary to treat vertex reliabilities explicitly and, in some way, handle component reliabilities changing as a function of time. This will complicate reliability analysis but does not make analysis infeasible.

The extension to unreliable vertices is not hard. All K-vertices must work if the system is to work and thus, for unreliable K-vertices, we obtain network reliability as usual and then multiply by the probability that all K-vertices function. Unreliable non-K-vertices may be handled in several ways. One of the simplest ways is to split each unreliable non-K-vertex into two perfectly reliable vertices with an unreliable edge between the two vertices with a failure probability equal to the failure probability of the original vertex. All edges incident into the original vertex are made incident into the tail vertex of the new edge and all edges incident out of the original vertex are made incident out of the new head vertex. However,

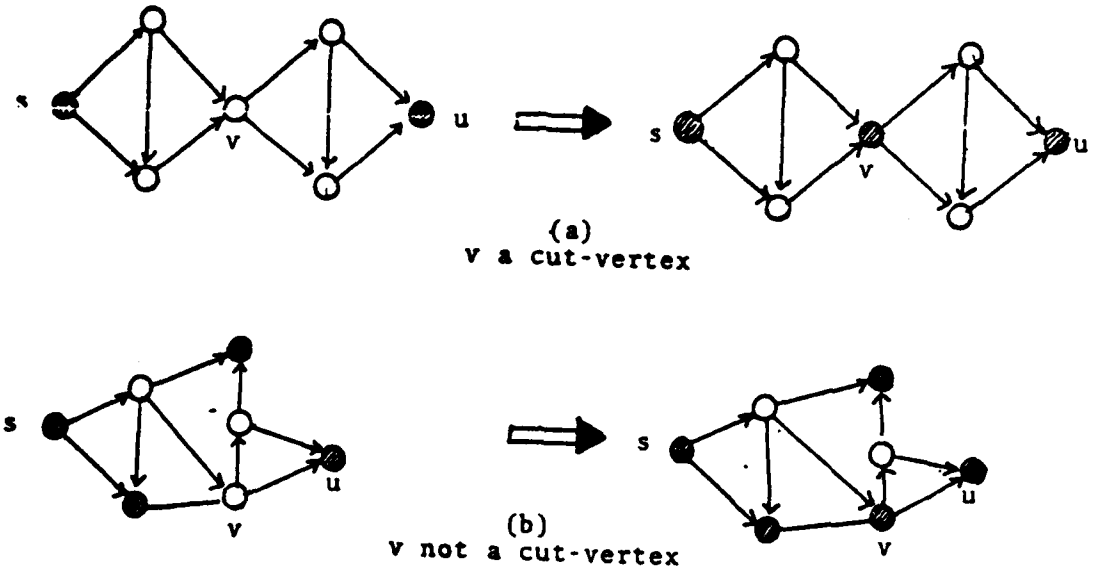
it is also possible to redefine the reduction and decomposition schemes and redefine the general reliability expression to directly handle unreliable non-K-vertices. For example, the series reduction can include the vertex reliability so that $p_c = p_a p_b p(v_i)$. Also, the term in the general reliability expression associated with "no edges in $E_{ij\ldots k}$ work" can be replaced with a term which expresses "no edges into v_i work or v_i fails, and no edges into v_j work or v_j fails, and ... except edges going between pairs of vertices in $\{v_i, v_j, \cdots, v_k\}$. This is not hard to compute.

Reliability changing as a function of time can be handled in one of two ways. The easiest way is to just input component reliabilities at different times and compute network reliability at these times. Interpolation can be used to approximate values between the times for which reliability is actually evaluated. Another possibility is to produce a symbolic reliability expression that can be repeatedly evaluated by a computer at different times, thus avoiding repeated reductions, decompositions, etc. This might require large amounts of storage but, if reliability can be calculated at all in a reasonable amount of time, it should be feasible. Once the price is paid, computing reliability for any given point in time should be very efficient.

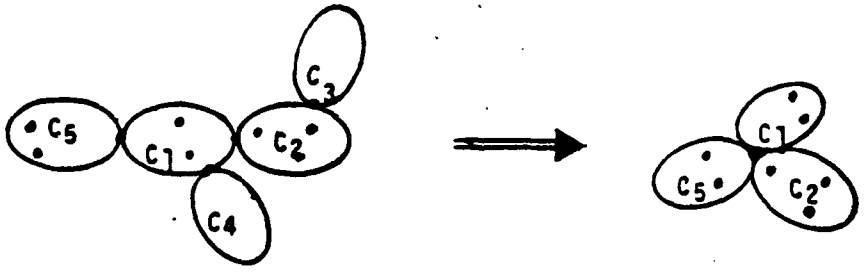
As modern computer, power and other network systems become larger and more complicated, reliability becomes a more important part of design criteria. Unfortunately, it is obvious that as networks become more complicated they also become more difficult to analyze. Further research should concentrate on ways to streamline computations and to find accurate but easily computable bounds. The method described in chapter 4 for computing SKT reliability could be improved if only non-zero terms were generated; An enumeration scheme similar to that of Satyanaryana and Prabhakar [1978] should help in this respect. The lower bound described in this thesis is a good start on finding accurate and easily computable

bounds, but every lower bound should be accompanied with an upper bound so that accuracy can be checked. Finding a good upper bound should be a priority for future research.

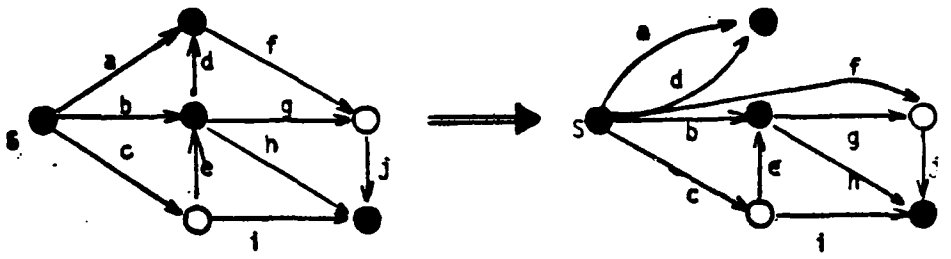
APPENDIX A



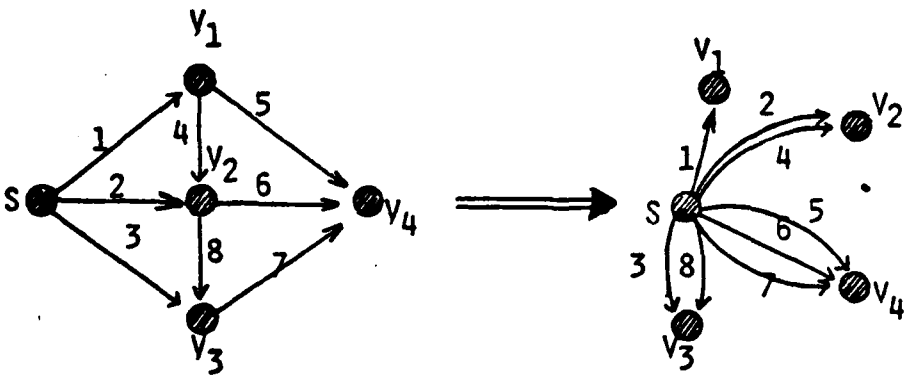
Neck Vertex to K-vertex Reduction
<Figure 3.1>



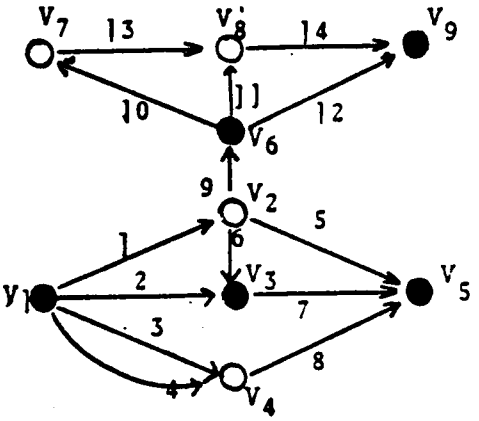
Decomposition
<Figure 3.2>



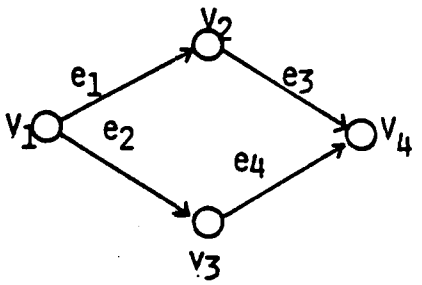
Moving Edges to Source
<Figure 3.3>

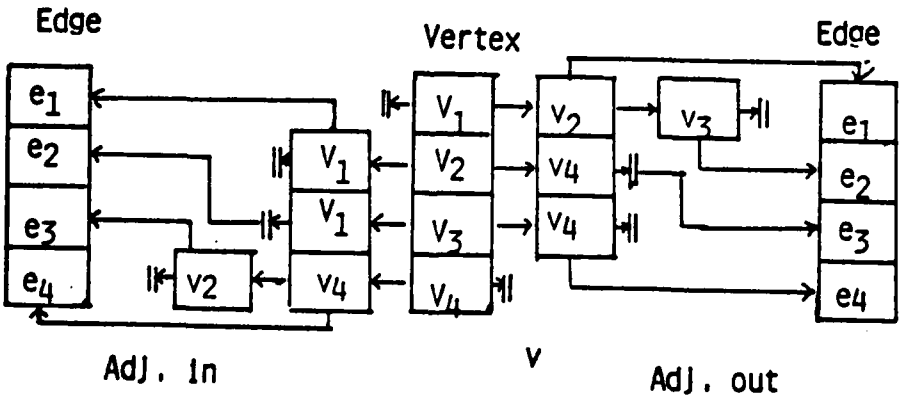


All Terminal Reliability $R_{sV}(D_1) = R_{sV}(D_2)$
<Figure 4.1>

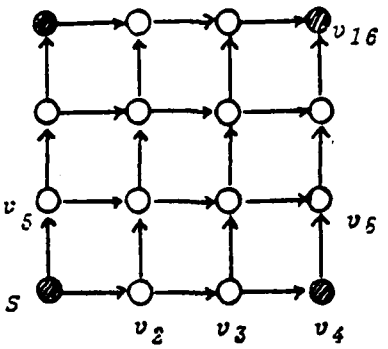


Completely Reducible Network by Reduction Schemes
<Figure 5.1>

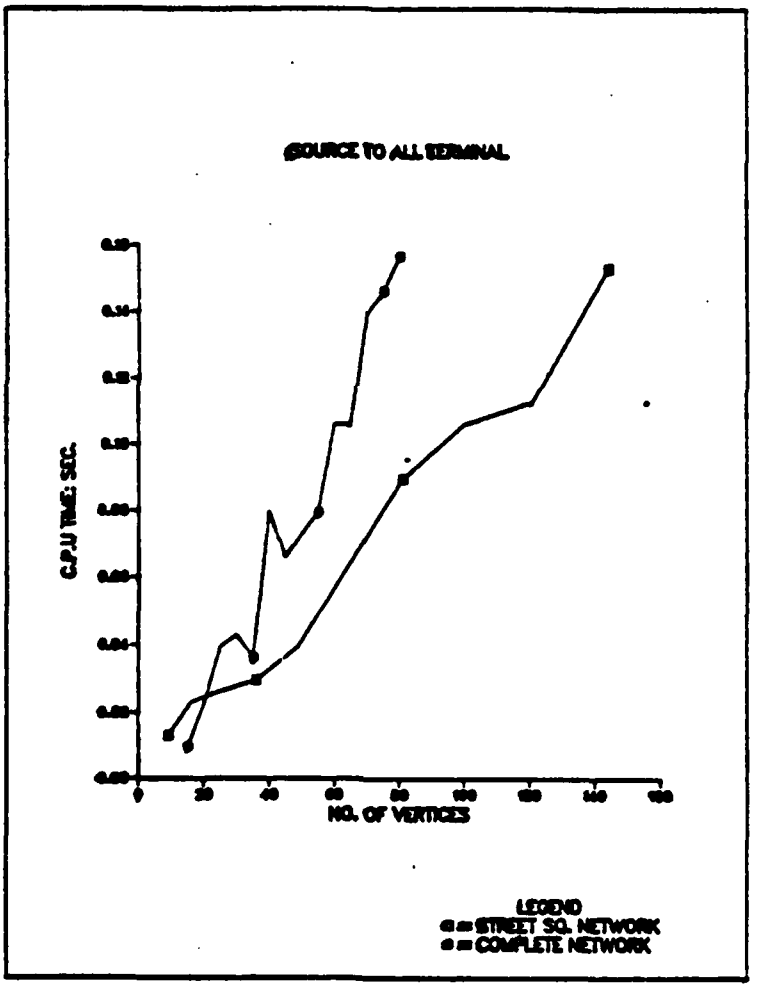




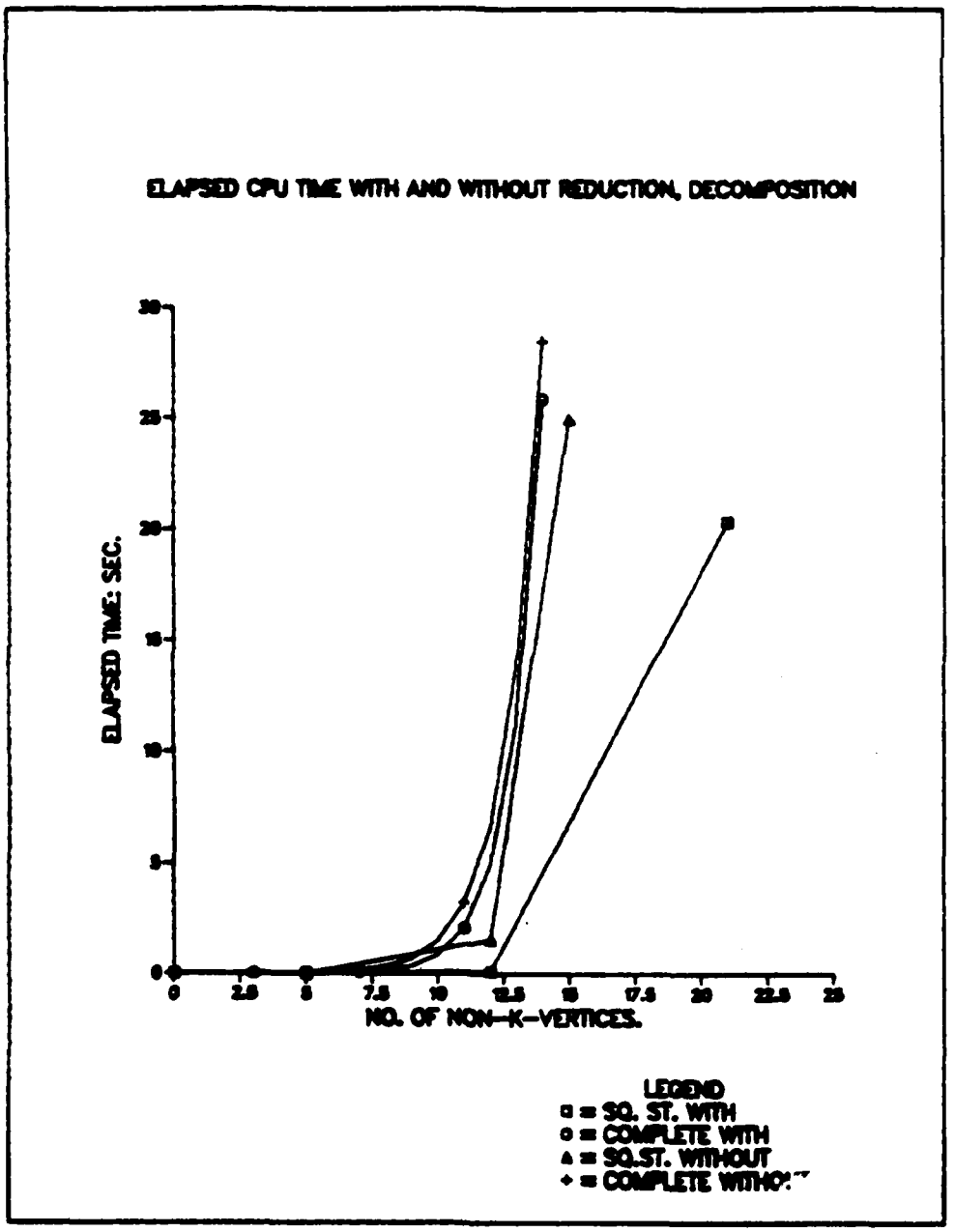
Multilist Data Structure
<Figure 5.2>



4×4 Square "Street" Network
<Figure 6.1>

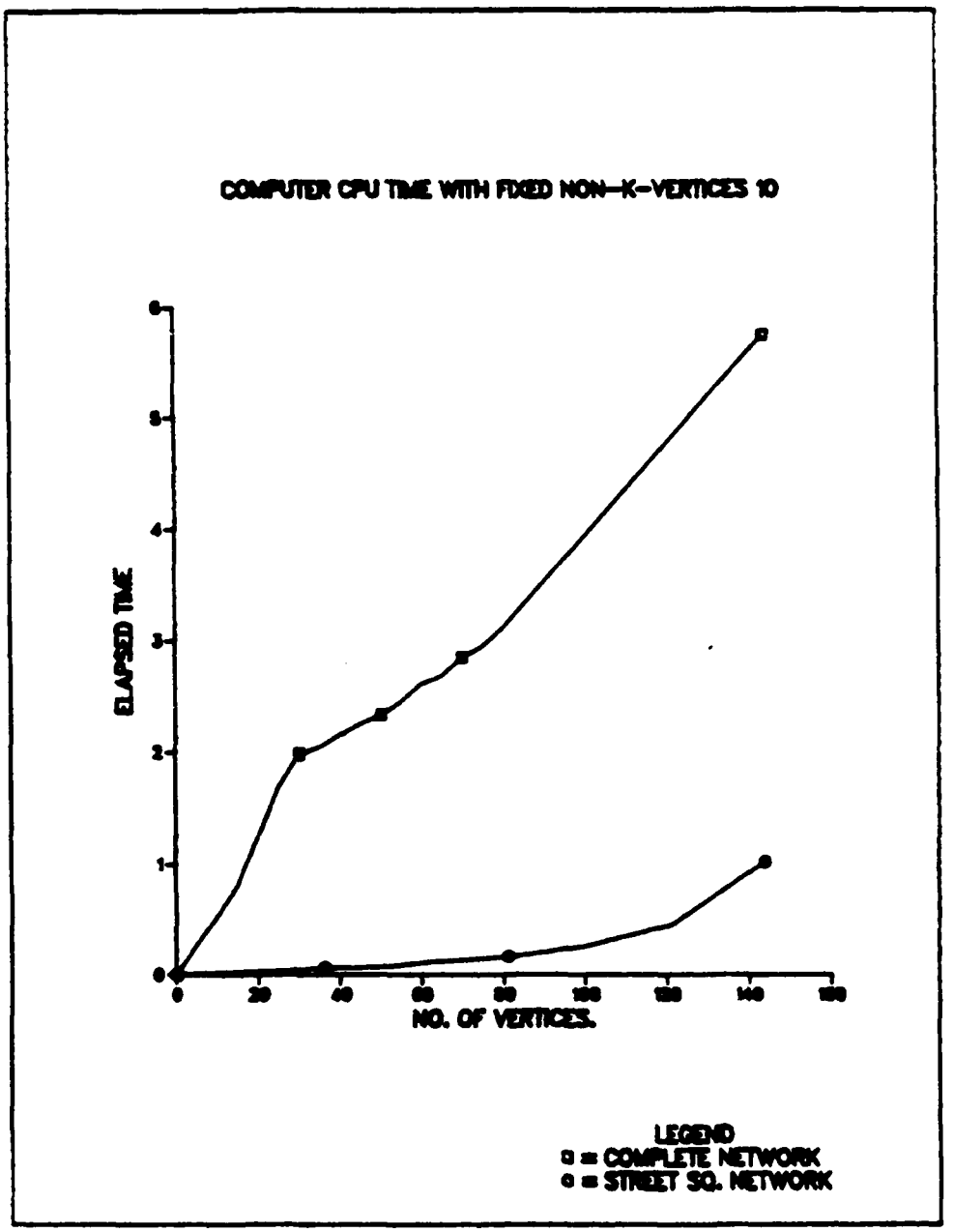


Source-To-All Terminal Network
< Figure 6.2 >



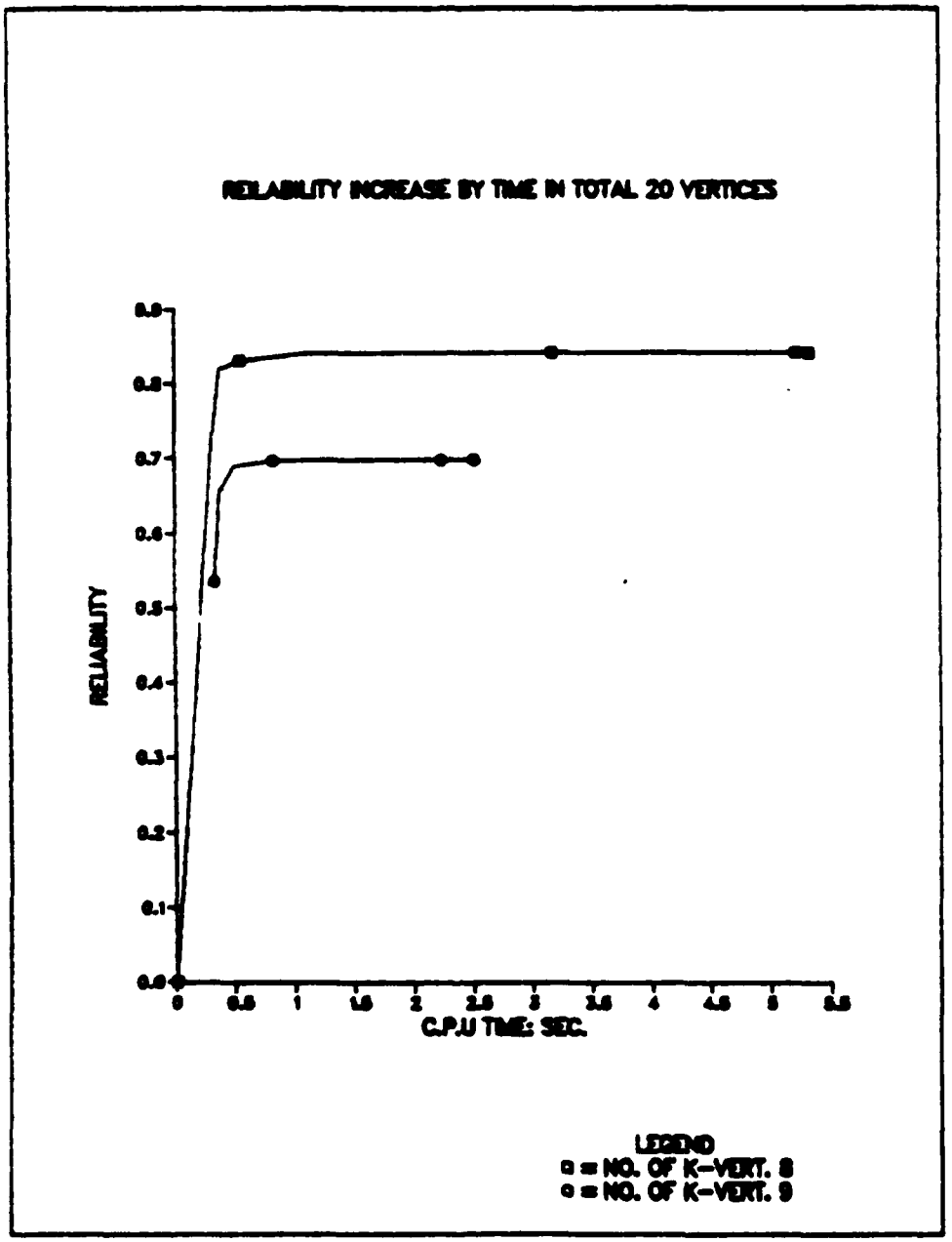
Souce-To-K Terminal Network

< Figure 6.3 >



Source-To-K Terminal with Fixed Non-K-Vertices

< Figure 6.4 >



Network Reliability Increment

< Figure 6.5 >

LIST OF REFERENCES

Agrawal, A. Reliability Analysis of Rooted Directed Networks, Ph.D. Thesis, IEOR. University of California, Berkeley. 1982.

Ball, M.O. and Provan, J.S. "The Complexity of Computing Cuts and of Computing the Probability that a Graph is Connected," Working Paper 81-002, University of Maryland at College Park. 1981.

Garey, M.R. and Johnson, D.S. Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., San Francisco, California. 1979.

Hagstrom, Jane N. Combinatoric Tools for Computing Network Reliability, Ph.D. Thesis, Dept. of IEOR, University of California, Berkeley. 1981.

Johnson, R. Some Combinatorial Aspects of Network Reliability Ph.D. Thesis, Dept. of IEOR, University of California, Berkeley. 1982.

Rosenthal, A. Computing Reliability of Complex Systems, Ph.D. Thesis, University of California, Berkeley. 1974.

Satyanarayana, A. "A Unified Formula for Analysis of Some Network Reliability Problem," IEEE Trans. Reliability, R-31, 23-32. 1982.

Satyanarayana, A and M.K. Chang, M.K. "Network Reliability and the Factoring Theorem," Networks, to appear. Also, ORC 81-12, Operational Research Center, University of California, Berkeley. 1981.

Satyanarayana, A. and Prabhakar, A. "New Topological Formula and Rapid Algorithm for Reliability Analysis of Complex Systems," IEEE Trans. Reliability, R-27, 82-100. 1978.

Satyanarayana, A. and Wood, R.K. "Polygon-To-Chain Reductions and Extensions for Reliability Evaluation of Undirected Networks," ORC 82-4, Operation Research Center, University of California, Berkeley. 1982.

Shogan, A.W. "A Decomposition Algorithm for Network Reliability Analysis," Networks, Vol.8, No 3, 231-251. 1978.

Tarjan, R. "Depth-First Search and Linear Graph Algorithms," SIAM J. Comput., pp 146-160. Vol. 1, 1972.

Valiant, Leslie G. "The Complexity of Enumeration and Reliability Problem," SIAM J. Computing, 8, 410-421. 1979.

Wood, R.K. Polygon-To-Chain Reductions and Extensions for Reliability Evaluation of Undirected Networks, Ph.D. Thesis, Dept. of IEOR, University of California, Berkeley. 1982.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Asst. Professor R.K. Wood Code 55Wd Department of Operations Research Naval Postgraduate School Monterey, California 93943	2
4.	Professor J.D. Esary Code 55Ey Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
5.	Major; Lee, Chan 19 Tong 3 Ban 387-1 Kongdug 2 Dong Mapoku Seoul, Korea.	5
6.	Dr. Avinash Agrawal Tymnet 2710 Orchard Parkway San Jose, CA 95134	1
7.	Prof. Richard Barlow Operations Research Center Etcheverry Hall University of California, Berkeley Berkeley, CA 94720	1

END

FILMED

2-84

DTIC