


***k*-Terminal Reliability of Communication Networks**

By

Wael Hasan Abdullah Saafin

Supervisor

Dr. Bassam Kahhaleh



عميد كلية الدراسات العليا

Co-Supervisor

Prof. Jamil Ayoub

Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science in
Communications

Faculty of Graduate Studies
University of Jordan

October 1998

This thesis was successfully defended and approved on Oct. 3, 1998

Examination committee

Signature

Dr. Bassam Kahhaleh

Assoc. Prof. of Electrical Engineering (Computer) *B. Kahhaleh*

Dr. Jamil Ayoub

Prof. of Electrical Engineering (Communications) *Jamil Ayoub*

Dr. Mohamed K. Abdelazeez

Prof. of Electrical Engineering (Communications) *m.k. abdelazeez*

Dr. Souheil Odeh

Assist. Prof. of Electrical Engineering (Computer) *Souheil Odeh*

Dr. Mohammad Ibbini

Assoc. Prof. of Electrical Engineering (Communications) *M. Ibbini*

Dedication

*To the heroes of Palestine
wherever they fight,
however they sacrifice*

Wael H. Saafin

Acknowledgement

I would like to thank my supervisors: Prof. Jamil Ayoub and Dr. Bassam Kahhaleh; for their directions, assistance and patient hard working in all steps of writing this thesis. Also thanks for the examination committee.

I should not forget to thank the employees of the computer center in the Faculty of Engineering and Technology for their assistance.

List of Contents

| | |
|-----------------------|------|
| Dedication | iii |
| Acknowledgement | iv |
| List of Contents..... | v |
| List of Tables..... | vii |
| List of Figures..... | viii |
| Appendices..... | x |
| Abstract..... | xi |

Chapter 1 : Introduction

| | |
|--|---|
| 1.1 Communication Networks | 1 |
| 1.2 Reliability of Communication Networks..... | 6 |
| 1.3 k -Terminal Reliability..... | 7 |
| 1.4 Theoretical Concepts of a Graph | 9 |

Chapter 2 : Search Methods

| | |
|---|----|
| 2.1 Introduction..... | 14 |
| 2.2 Depth-First Search..... | 15 |
| 2.3 Breadth-First Search..... | 18 |
| 2.4 Backtracking..... | 21 |
| 2.5 Reliability Evaluation: General Survey..... | 24 |

Chapter 3: Proposed Reliability Algorithms

| | |
|---------------------------------|----|
| 3.1 Assumptions..... | 30 |
| 3.2 Monte Carlo Simulation..... | 31 |

| | |
|------------------------------------|----|
| 3.3 Algorithm 1, <i>ALG1</i> | 35 |
| 3.4 Algorithm 2, <i>ALG2</i> | 44 |

Chapter 4: Results

| | |
|--|----|
| 4.1 Algorithm Performance..... | 52 |
| 4.2 <i>ALG1</i> and <i>ALG2</i> Vs Previous Studies..... | 68 |
| 4.3 All <i>k</i> -Terminal Reliability..... | 75 |

Chapter 5: Conclusions

| | |
|-----------------------------|-----|
| References..... | 85 |
| Appendices..... | 90 |
| Abstract (in Arabic)..... | 106 |

List of Tables

| | | |
|------|---|-------|
| 4.1 | Reliability and Time vs NS for A -networks, $p_i = 0.9$ | 56 |
| 4.2 | Reliability and Time vs NS for A -networks, $p_i = 0.4$ | 57 |
| 4.3 | Time for one execution run for A -networks, $p_i = 0.9$ | 61 |
| 4.4 | Time for one execution run for A -networks, $p_i = 0.4$ | 61 |
| 4.5 | Reliability and Time vs NS for B -networks, $p_i = 0.9$ | 63 |
| 4.6 | Reliability and Time vs NS for K_2 -network, for $p_i = 0.95$ and $p_i = 0.1, k = 3$ | 65 |
| 4.7 | Time for one execution run for K_2 -network | 67 |
| 4.8 | Two values of NS and the corresponding R_K for K_2 network | 67 |
| 4.9 | Reliability and Time vs p_i for K_2, A_4 and B_4 networks | 68 |
| 4.10 | Reliability and Time vs N for A -networks using $ALG1,$ $ALG2$ and MAP | 72 |
| 4.11 | Reliability and Time vs NS for B_1 and B_2 networks using $ALG1, ALG2$ and $PRFA$ | 74 |
| 4.12 | All k -terminal set for K_1 network | 76-79 |

List of Figures

| | | |
|-----|---|----|
| 1.1 | A linear, simple, undirected graph, $G = (7,10)$ | 3 |
| 1.2 | k -terminal reliability problem | 9 |
| 2.1 | Depth-first search | 17 |
| 2.2 | Breadth-first search | 20 |
| 2.3 | A directed graph as a result of <i>BFS</i> | 22 |
| 2.4 | Reduce and factor process | 25 |
| 3.1 | Flow chart for <i>ALG1</i> | 36 |
| 3.2 | k -terminal reliability problem | 39 |
| 3.3 | <i>ALG1</i> solution for a sample network | 40 |
| 3.4 | <i>ALG1</i> solution for a sample network | 41 |
| 3.5 | Flow chart for <i>ALG2</i> | 45 |
| 3.6 | <i>ALG2</i> solution for a sample network | 46 |
| 3.7 | <i>ALG2</i> solution for a sample network | 47 |
| 3.8 | <i>ALG2</i> solution for a sample network | 48 |
| 4.1 | A simple, undirected graph $A_1 = (6,9)$ | 53 |
| 4.2 | Reliability and Time vs <i>NS</i> for A_1 -network for $p_i = 0.9$ and $p_i = 0.4$ | 58 |
| 4.3 | Reliability and Time vs <i>NS</i> for A_4 network for $p_i = 0.9$ and $p_i = 0.4$ | 59 |

| | |
|---|-------|
| 4.4 Reliability and Time vs NS for A_7 network for $p_i = 0.9$ and $p_i = 0.4$ | 60 |
| 4.5 B -networks | 62,63 |
| 4.6 Reliability and Time vs NS for B_1 and B_2 networks | 64 |
| 4.7 k -terminal reliability problem, $K_2 = (9,13)$ | 65 |
| 4.8 Reliability and Time vs NS for K_2 network, for $p_i = 0.95$ and $p_i = 0.1$ | 66 |
| 4.9 Time vs p_i for A_4 network | 69 |
| 4.10 Time vs p_i for B_2 network | 70 |
| 4.11 Time vs p_i for K_2 network | 71 |
| 4.12 Time vs N for A -networks, $p_i = 0.9$ | 73 |
| 4.13 Time vs N for A -networks, $p_i = 0.4$ | 74 |
| 4.14 K_J -network | 75 |

Appendices

| | |
|---|-----|
| Appendix 1: <i>f-t</i> To Connection-Matrix Transformer | 91 |
| Appendix 2: Incidence- T Connection-Matrix Transformer | 93 |
| Appendix 3: <i>ALG1</i> | 95 |
| Appendix 4: <i>ALG2</i> | 99 |
| Appendix 5: Network Generator | 103 |

Abstract

k-Terminal Reliability of Communication Networks

By

Wael Hasan Saafin

Supervisor

Dr. Bassam Z. Kahhaleh

Co-Supervisor

Prof. Jamil Ayoub

One of the important fields in communication networks is the reliability of these networks. The *k*-terminal reliability is our main concern in this thesis. It is the probability of finding a communication path among some specified set of terminals that should be connected with one another.

There are two general approaches to solve this problem; exact and approximate. Exact solution usually can not be applied except for small networks due to the impractical time it takes to obtain a solution.

Two algorithms that are of the approximation approach are suggested in this thesis. They introduce a solution for a generic network in feasible and practical time, while achieving good accuracy. In these two algorithms, Monte Carlo simulation is utilized, and a sufficient number of samples are used to get the reliability measure. In each considered sample, some links are assumed to fail depending on a given probability. For a given sample network, a search method is used to check if the *k*-terminal nodes are connected with each other by working links or not. In the first algorithm,

ALG1, we utilize the depth-first search, while in the second algorithm, *ALG2*, breadth-first search is utilized to check for the connectedness of the k -terminal nodes.

We propose a general solution for the k -terminal reliability problem without any restriction on the size or the topology of the network under study.

Chapter 1

INTRODUCTION

In this chapter a brief introduction is given as a background for this study. Section 1.1 starts with communication networks, then some methods of mathematical modeling are stated. Section 1.2 covers the k -terminal reliability issue, while Section 1.3 states the k -terminal reliability problem. Finally, Section 1.4 covers some theoretical concepts required by the following chapters.

1.1 Communication Networks :

A communication network consists of transmission paths that connect various nodes such as switching centers, concentrators and terminals. This network can be represented by a graph, which is independent of the nature of communication paths (radio, cable,...etc.) and independent of the

functions accomplished at the nodes (switching, concentration, information entry, information receipt,...etc.).

A graph may be considered to be directed or undirected depending on the direction of information flow through its edges or links. A link is said to be undirected if information flow can be in both directions, and is represented by a line without arrows, connecting two terminal nodes. If information flow is dictated to one direction through each link of the graph then the links, and hence the whole graph, are said to be directed with an arrow added on each edge so as to indicate direction of information flow.

This thesis is concerned with linear, simple and undirected graphs. A graph is said to be simple if it has neither self-loops; i.e no edge has the same vertex as both of its end vertices, nor parallel edges, i.e no two edges have the same end vertices.

An example of a linear, simple and undirected graph is drawn in Figure 1.1. This network contains seven terminals or nodes that are connected in the shown manner by ten links. As a graph, it contains seven vertices that are connected, in the shown manner, by ten edges. Clearly, this graph is undirected, since no arrows are drawn, and it is simple.

Graphs are named by giving both the number of vertices and edges. It is written as $G=(V,E)$ where V and E are the number of vertices and edges respectively. In our example, in Figure 1.1, the graph is labeled $G=(7,10)$. In this figure, the vertices, v_k , are numbered from 1 to 7 for future references.

An edge e_k may be numbered independently of the vertices, or it may be represented as an unordered pair (v_i, v_j) where v_i and v_j are called the end vertices of e_k . In Figure 1.1 edge 8 has vertices 1 and 7 as end vertices, i.e $e_8 = (v_1, v_7) = (v_7, v_1)$

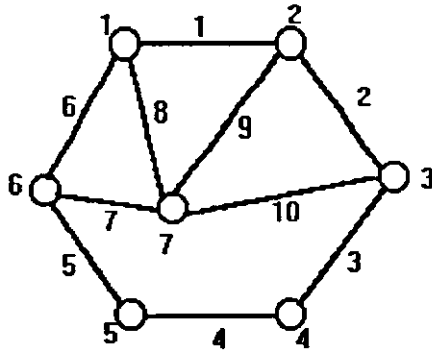


Figure 1.1: A linear, simple, undirected graph, $G = (7,10)$

The previous graphical representation is an important step in formulating the mathematical model of a communication network. A trivial representation would be

$$G = \{e_1, e_2, e_3, \dots, e_E\}$$

where E is the number of edges. Each edge e_k has two end vertices: one of them is stored as the k th entry of one array say f array, and the other one is stored as the k th entry of a second array say t array. The previous example is then given by:

$$G = \{e_1, e_2, e_3, \dots, e_{10}\}$$

$$= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

$$f = \{1, 2, 3, 4, 5, 6, 6, 1, 7, 7\}$$

$$t = \{2, 3, 4, 5, 6, 1, 7, 7, 2, 3\}$$

493336

Note that the first array consists of only a counter that counts from 1 to E . Hence, mathematically we can refer only to f and t arrays so as to get a complete information about the concerned graph. Also, it makes no difference for an edge which one of its vertices goes to f and which one goes to t . What is important is to write the end vertices of an edge e_k as the k th entry of both f and t . This is a result of the undirectivity of the graph.

493336

Another mathematical representation is the incidence-matrix representation. The incidence matrix contains only two possible elements: one or zero, i.e. it is a binary matrix or $(0,1)$ -matrix. Rows denote the vertices while columns denote the edges of the graph. The vertex v_k is said to be incident to the edge e_j if v_k is an end vertex of the edge e_j . In our example, v_2 and e_9 are incident to each other. If there are three edges e_m, e_n , and e_o incident to a vertex v_k then the k th row of the incidence matrix will be all zeros except for the m th, n th and o th columns, which will have the value of one. In general

$$A = [a_{ij}]$$

$$a_{ij} = \begin{cases} 1 & \text{if edge } e_j \text{ is incident on vertex } v_i \\ 0 & \text{otherwise} \end{cases}$$

Back to our example, its incidence matrix $A(G)$ would be:

$$\begin{array}{c}
 \\
 v_1 \\
 v_2 \\
 v_3 \\
 v_4 \\
 v_5 \\
 v_6 \\
 v_7
 \end{array}
 \begin{array}{cccccccccc}
 e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} \\
 \left[\begin{array}{cccccccccc}
 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
 \end{array} \right]
 \end{array}$$

Note that every column of A has exactly two ones, since each edge is connecting two vertices only. The number of ones in a given row equals the degree of the corresponding vertex, i.e. the number of edges incident to that vertex. Note that the dimension of the incidence matrix is $V \times E$.

Another important matrix representation of graphs is based on the idea of adjacency between vertices. Two vertices v_i and v_j are said to be adjacent if they are both incident on one common edge e_k , that is v_i and v_j are the end vertices of the edge e_k . In our example, v_6 and v_7 are adjacent since they are the end vertices of edge e_7 . A matrix representation of this type is called the adjacency matrix or connection matrix. It is a $V \times V$ square matrix.

In general;

$$X = [x_{ij}]$$

where

$$x_{ij} = \begin{cases} 1 & \text{if there is an edge between vertices } v_i \text{ and } v_j \\ 0 & \text{otherwise} \end{cases}$$

Now back to our example, the adjacency matrix is

$$\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{array} \begin{bmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The adjacency matrix is symmetrical as a result of the undirectivity of its graph. Also it has zero diagonal entries due to the simplicity of its graph, i.e. no self-loops and no parallel edges. The number of ones in a row, or a column, is equal to the degree of the corresponding vertex. For example, vertex v_7 has a degree of four since four ones can be seen in the 7th row or the 7th column due to symmetry.

There are other matrix representations of graphs. However in this thesis, the above mentioned representations are sufficient. Given any one of these matrix representations, it is possible to construct a geometric graph G without any ambiguity, and having the same information as its mathematical representation. However, some applications may prefer one representation over the others, i.e. for simpler programming, to get better performance, etc.

1.2 Reliability of Communication Networks:

An important step when designing a communication network is to calculate its expected reliability, that is the survivability of data when transferred through this network. Also, it is important for improving existing working networks, since the reliability analysis would give the designer indicative information such as where to add new links or help to change the network topology so as to get better performance.

Reliability problems are of two major types: deterministic and probabilistic. In deterministic reliability, no probability of failure is assigned to links or nodes, and we are concerned with some sets of nodes or links that may affect the network functionality. This type of problems measures the cohesion and the connectivity of networks (Cravis, 1981).

Probabilistic reliability problems assume that the links or nodes may fail at random. The reliability here depends on both topology of the specified network and the individual reliability of each link and node in the network.

Another classification of reliability problems that depends on the terminals or nodes of interest is given as follows

- Terminal-pair reliability

- All-terminal reliability
- k -terminal reliability

The terminal-pair, or two-terminal, reliability problem is to determine the probability of successful communication between two specified nodes in a network, given the probability of success for each communication link in the network.

Computing the reliability of a generic network is computationally difficult, and an exact solution is usually unattainable. Faced with this computational difficulty, an assumption is usually made that all types of components are perfectly reliable except for one, and usually the link component is assumed to be that one. In this thesis, references to some previously published solutions are given for the purpose of comparing them to existing algorithms.

In the all-terminal, or overall terminal, reliability problem, the objective is to find the probability that every pair of nodes can communicate with each other using working links, assuming perfectly reliable nodes, as in the case of terminal-pair problem, and given the reliability of each communication link in the network. Such a measure is needed in some networks where all terminals must be in touch with one another all the time. The problem of optimum design of link topology is then to maximize reliability or to minimize the cost, both requiring to calculate or estimate the network reliability.

1.3 k -Terminal Reliability:

The k -terminal reliability problem is the general case of the three major problems. Its solution includes solutions for both terminal-pair and all-

terminal problems. In the k -terminal problem, a network is given along with the reliability of each communication link. Nodes are assumed to be perfectly reliable, as well as the k terminals of special interest. The requirement is to calculate the probability of successful communication between any two terminals from the specified set of k terminals. Note that if the set of k terminals includes only two elements then the problem becomes a terminal-pair problem, while if the set of k terminals includes all the terminals in the network then the k -terminal problem becomes an all-terminal reliability problem.

For example, Figure 1.2(a) shows a graph G that represents the original communication network. It has 9 vertices and 13 edges hence $G = (9,13)$, the reliability of each edge e_i in E is given and it is equal to 0.9.

In this example the network edges have the same reliability. Even in the general case where edges have different edge reliabilities, the used methodology remains the same. For this example, the k terminals of interest are specified to be vertices one, four and nine.

What is required now is to calculate the probability of nodes one, four and nine being connected together via any working edges independent of the states of other nodes and edges that may not affect this connection. Naturally, investigating the graphical model, one may find many paths that connect the k nodes together in a single tree. Two of these trees are shown in Figure 1.2(b). If, for example, the edges found in one of these two trees have reliability of one, then the k -terminal reliability is one irrespective of, say edge (1,6). However, a 100% reliability is only theoretical and is stated here only to clarify the idea.

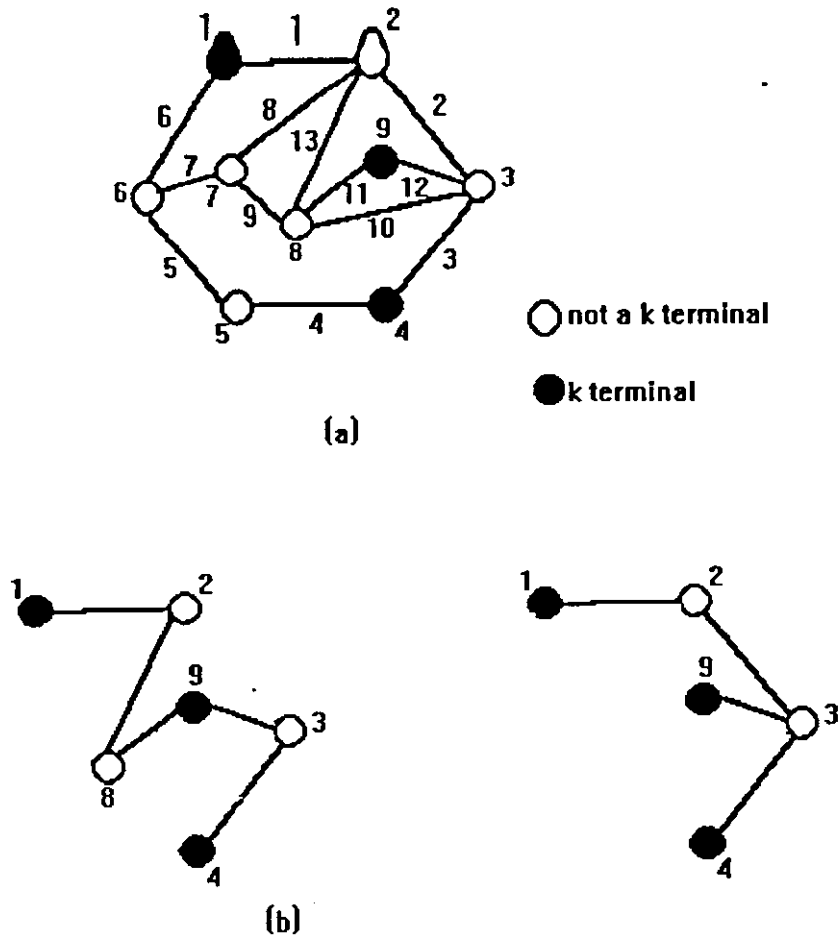


Figure 1.2: k -Terminal reliability problem

(a) $G=(9,13)$, a network of 3 k -terminal nodes (1,4 and 9)

(b) two possible paths connecting the k -terminal nodes.

1.4 Theoretical Concepts of a Graph:

Before stating any solution methods, there are many definitions and concepts that should be firstly clarified.

A labeled graph is a graph in which each vertex is assigned a unique name or label. Two vertices v_i and v_j are said to be adjacent if they are the end vertices of some edge e_k in G , i.e. they are adjacent if there is an edge connecting them together. Two edges e_i and e_j are said to be adjacent if they share an end vertex v_k , i.e. they are adjacent if $e_i = (v_m, v_k)$ and $e_j = (v_m, v_k)$ irrespective of the vertices v_m and v_n .

A vertex v_i and an edge e_j are said to be incident with, or on to, each other if v_i is an end vertex of the edge e_j , i.e. $e_j = (v_i, v_m)$ irrespective of the vertex v_m .

A walk, through a graph, is a finite alternating sequence of vertices and edges beginning and ending with vertices, such that each edge is incident with the vertices preceding and following it. A special case of a walk is when no vertex appears more than once. This special walk is called a path, or simple path, whose length is the number of edges in that path. A graph is connected if there is at least one path between every pair of vertices in G . A graph may have different paths with different lengths while connecting the same set of nodes. A path of minimum number of edges is called a shortest-path that interconnects some set of vertices.

A tree is a connected graph without any circuits, i.e. it does not have any closed walks in which there is at least one vertex that appears more than once. By its nature, there is one and only one path between every pair of vertices in a tree T . Clearly, a tree of (n) vertices has $(n-1)$ edges.

A binary tree is defined as a tree in which there is exactly one vertex of degree two while every other vertex is of degree one or three. The vertex of degree two is the root of the tree. Note that the degree of a vertex is equal to the number of edges incident to it. In a binary tree, a vertex v_i is said to be at level l_i if v_i is at a distance of l_i from the root while the root is assumed to

be at level zero. The distance between two levels is the minimum number of edges connecting between them (Deo, 1990).

A spanning tree T , is a tree of a connected graph G provided that T contains all vertices of G . Note that a spanning tree is only defined for a connected graph since a tree is always connected while in a disconnected graph of V vertices, one cannot find a connected subgraph with V vertices. Finding a spanning tree of a connected graph is done by simply deleting an edge from each circuit, if any, found in the graph, leaving a connected, circuit-free graph that contains all the vertices of G .

A disconnected graph consists of two or more connected subgraphs, each of these connected subgraphs is called a component.

A cut set is a set of edges whose removal from an originally connected graph G , leaves it disconnected, or it is the minimum set of edges in a connected graph whose removal increases the number of components in the graph by one.

Chapter 2

SEARCH METHODS

This chapter covers the main search methods that are utilized in this thesis. These search methods are needed so as to get around the fact that exact solutions for reliability problem are, in general, not feasible to get, and approximation methods depend completely on these methods. Binary search is investigated in Section 2.1, depth-first search and breadth-first search are discussed in Sections 2.2 and 2.3, respectively. Backtracking which may be considered as a help tool is studied in Section 2.4.

In Section 2.5 some previous studies are discussed including both exact and approximation types. This chapter is intended to provide the required background necessary to investigate and discuss the proposed algorithms which are stated in the next chapter.

$$\max l_{\max} = (n-1)/2$$

A pendant vertex is a vertex that hasn't any adjacent vertices of higher levels in the binary tree. In analysis of algorithms, we are generally interested in computing the sum of the levels of all pendant vertices. This quantity, known as the path length of a tree, can be defined as the sum of the path lengths from the root to all pendant vertices.

The importance of the path length of a tree lies in the fact that this quantity is often directly related to the execution time of an algorithm; an important factor in choosing the best algorithm to follow.

2.2 Depth-First Search:

Depth-first search is a technique that is widely used for finding solutions to problems in combinatorial theory and artificial intelligence (Tarjan, 1972). Suppose G is a graph which we wish to explore. Initially, all the vertices of G are unexplored. We start from some vertex of G and choose an edge, from the set of incident edges, to follow. Traversing this edge leads to a new vertex. We continue in this way; and at each step we traverse an unexplored edge leading from an already reached vertex. A traversed edge leads to some vertex, either new or already reached before. Whenever we run out of edges leading from old vertices, we choose some unreached vertex, if any exists, and begin a new exploration from this new vertex. Eventually, we will traverse all the edges of G each exactly once. Such a process is called a search of G .

There are many ways of searching a graph, depending on the way in which search edges are selected. Consider the following choice rule:

when selecting an edge to traverse, always choose an edge emanating from the vertex most recently reached.

A search which uses this rule is called a depth-first search. The set of old vertices with possibly unexplored incident edges may be stored on a stack.

Let us take G as a connected undirected graph. A search of G imposes a direction on each edge of G , defined by the direction in which the edge is traversed when the search is performed. Thus G is converted into a directed graph G' . The set of edges which lead to a new vertex when traversed during the search defines a spanning tree of G' . All edges that make up the palm tree of the original graph are usually called fronds of G , while each edge (v,w) , which is not part of the spanning tree, connects vertex v to one of its ancestors w , i.e. a vertex that has been previously visited in the search method.

The previous statements may be translated into the following algorithm.

Input: An undirected graph $G = (V,E)$ represented by the adjacency set for each vertex $v \in V$.

Output: A partition of E into a set T of tree edges $(v \rightarrow w)$ and a set B of back edges $(v \dashrightarrow w)$.

Step1: let $v_i = 1$

Step2: for an appropriate node w_i that lies in the adjacency list of v_i , assume $w_{i i} = w_{i j} = w_i$ then do steps 3 and 4.

Step3: if w_i is not yet numbered then construct $v_i \rightarrow w_{i i}$ and number w_i .

Step4: else construct $v_i \dashrightarrow w_{i j}$.

Step5: let $v_i = w_{i i}$

Step6: If yet there is an unnumbered vertex then go to step 2.

Step7: stop.

Figure 2.1 shows an example to clarify the algorithm.

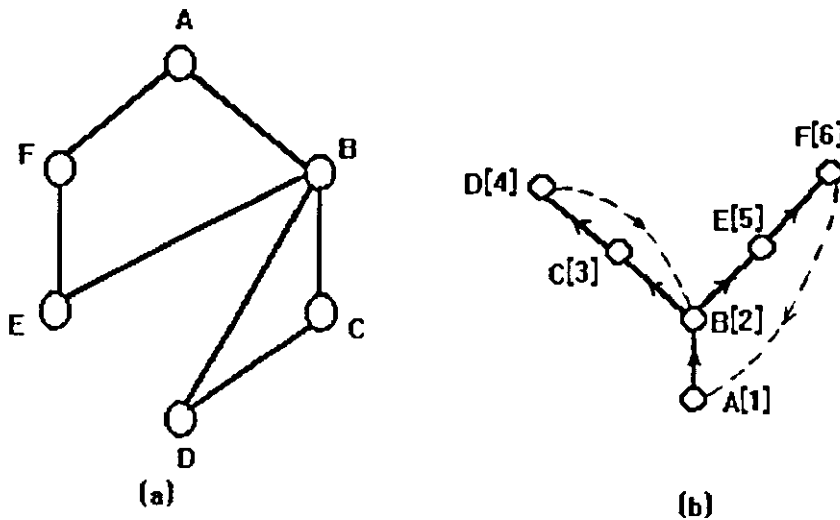


Figure 2.1: Depth-first search (a) $G = (6,7)$. (b) *DFS* palm tree, number in [] specify the vertex number as search procedure done

Let us start at a vertex A , and give it the number $v = 1$. At this node we will choose one of the adjacent vertices, i.e. one of (A,B) and (A,F) edges. Let us take the former one to construct the first $v \rightarrow w$, leading to vertex B at which v should be updated. At this vertex, three new possible incident edges may be followed. Let us choose edge (B,C) leading to vertex C . At this vertex, there are two adjacent vertices: the first is still unnumbered and can be reached through edge (C,D) , and the other is already numbered which is vertex B . Edge $C \rightarrow D$ is constructed and the new vertex we will be at is D . Now at vertex D only one new incident edge exists leading to the already

numbered vertex B so $D \rightarrow B$ is constructed. Hence B is an ancestor of vertex D . We still at vertex D where there are no incident edges, so we should go back through our spanning tree to the last numbered vertex C which also hasn't any new incident edges. Continue to vertex B where there is a new and sufficient edge (B, E) which is constructed leading to vertex E . Then $E \rightarrow F$ and $F \rightarrow A$ in the same manner.

A brief study of both the algorithm and the figure shows that the algorithm must terminate because each vertex can be numbered only once. Also, each edge in the graph is explored at most twice; a property that makes the search time linear in V and E . The edges $v \rightarrow w$ run from smaller numbered vertices to larger numbered ones. The edges $v \leftarrow w$ run from larger numbered vertices to smaller numbered ones. If edge $v \rightarrow w$ is constructed, edge $w \rightarrow v$ is not constructed, and edge $v \leftarrow w$ is not constructed later, because of the test performed in step 3. Thus each edge in the original graph is directed in one and only one direction.

In this thesis, the depth-first search method is utilized to develop an algorithm that finds the k -terminal reliability. This algorithm, named *ALG1*, is investigated in Chapter 3.

2.3 Breadth-First Search:

An opposite approach to the depth-first search method is the breadth-first search, which tries to find the shortest path between two specified terminals.

Initially, all vertices of the given graph G are unexplored. we start from some vertex of G and scan all edges incident on this vertex, then move to an

adjacent vertex . At the new vertex we scan all the incident edges and so on until all vertices in the graph are scanned or searched. Unlike depth-first search, breadth-first search is not naturally recursive. Moreover, in breadth-first search each vertex is visited only once. Thus, if the target of the search is to determine if two terminals are connected or not then this search follows the shortest path, i.e. the minimum number of edges between the two terminals of interest. As is the case in depth-first search, breadth-first search is carried out once for each connected component of the graph. The following algorithm is based on the breadth-first search method.

| |
|--|
| <p>Input: An undirected graph $G = (V, E)$ represented by adjacency sets for each $v_i \in V$</p> <p>Output: A partition of E into a set T of tree edges.</p> |
| <p>Step1: let $v_i = 1$</p> <p>Step2: for all w_i that lies in the adjacency list of v_i and not yet numbered ; construct $v_i \rightarrow w_i$ and number them.</p> <p>Step3: choose one of the new w_i say w_c ; then $v_i = w_c$</p> <p>Step4: if there is an unnumbered vertex yet then go to Step 2.</p> |

To illustrate this search technique, the same example used for illustrating the depth-first search, is redrawn in Figure 2.2.

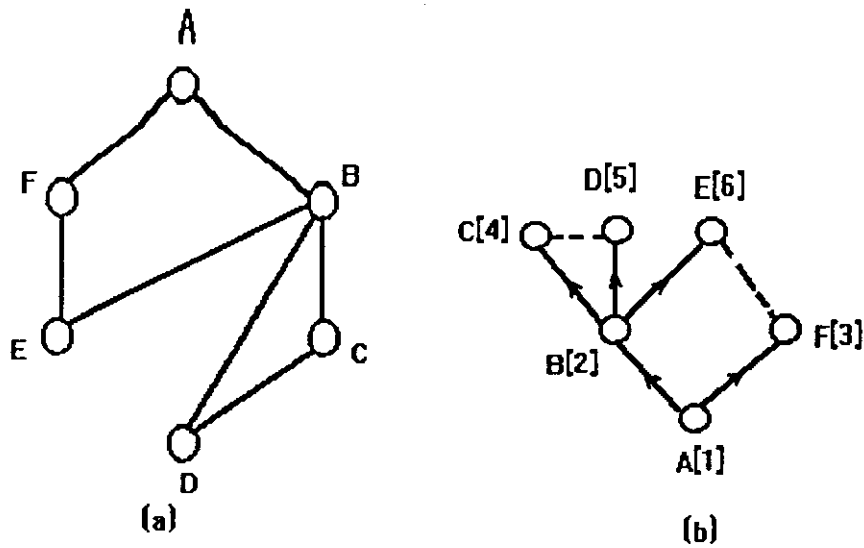


Figure 2.2 : Breadth-first search (a) $G = (6,7)$ (b) *BFS* tree with numbers in [] denotes the order of vertex exploration

Let us start at vertex A , and give it number 1 as shown in the figure. Both B and F vertices are adjacent to A ; so we will number them by 2 and 3 respectively. From the up-to-date vertices, let us choose B so as to be the next v . At B we will look for the remaining unnumbered adjacent vertices where there are three of such vertices: C , D and E ; number them with the numbers 4, 5 and 6. At this step, the number that we reached is six and equals the number of vertices for the whole graph, and since each vertex can be explored only and only once; so the search should terminate since all nodes were explored. However, as we can see, not all of the edges were explored; such as the edge connecting both C and D or that connecting E with F , drawn in Figure 2.2 as broken lines. Naturally, a simple modification may be made if one wants to investigate edges of such type so as to get two types of edges: fronds and back edges, as the case in depth-first search.

It is easy to show that the time required by a breadth-first search is in the same order as that required by a depth-first search which is linear, not

exponential, in the size of G . This is usually the best that one can expect from a graph algorithm, since it is reasonable to assume that each vertex and each edge must be processed. However, one search technique may be preferable over another, that is, it may give us a more efficient implementation. For example, breadth-first search is usually used where shortest-path routes are needed, since it travels through the minimum number of edges and/or nodes on the contrary of the case in depth-first search. On the other hand, depth-first search may be best used in connectivity problems that are of the deterministic type of reliability problems.

2.4 Backtracking:

Backtracking is an important method that should be utilized so as to best utilize search methods including both depth-first and breadth-first search methods. In depth-first search we may reach to a vertex that hasn't any unexplored edges incident with it, while still the graph is not completely explored. An example of this case is when reaching vertex D in Figure 2.1. There was no new incident edges while still edges (B,E) and (E,F) were unexplored. At this case we should backtrack our previous walk and return to the previously numbered vertices with the last being the first in backtracking, where at each vertex an adjacency list is checked if any new incident edges were found, in that example, we returned to C then to B for which edge (B,E) was incident.

While our previous discussion was restricted to depth-first search, backtracking is also important when used together with breadth-first search,

especially when the shortest path is required. The following is a backtracking algorithm for finding a shortest-path between two terminal points: source (s) and destination (t).

Input: A directed graph (resulted from the breadth-first search)
with distance of each vertex from the source is known.

Output: Backtracking tree (from t to s)

Step1: set $i = d(t)$ and assign $v_i = t$

$d(t)$ is the distance of vertex (t) from the source (s)

Step2: find a vertex (u) adjacent to v_i and with $d(u) = i-1$

assign $v_{i-1} = u$

Step3: if $i = 1$, stop

else decrement i and go to step 2

Let us discuss the same previous example which we searched using breadth-first search and let the source, s to be node A while the destination, t be vertex D , noting that at this special case, shortest $s-t$ path, a slight modification should be made on the graph numbering that finds the distance $d(v_i)$ with $d(s) = 1$, then vertices adjacent have $d(.) = 2$, etc. As can be seen in Figure 2.3, $d(t) = d(D) = 3$.

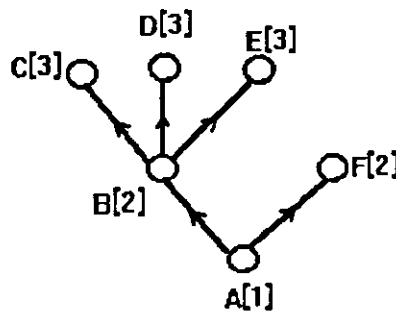


Figure 2.3 : A directed graph as a result of *BFS*, number in [] is the distance of the node w.r.t node A

Following the algorithm above we want u adjacent to t or D with $d(u) = 2$ which is vertex B , then assign $v_i = B$ which has an incident edge that leads to vertex A of $d = 1$ which is the source, i.e backtracking terminates and the path followed from D to A was $D \rightarrow B \rightarrow A$, which is a shortest path that has the minimum number of edges.

Hence, backtracking algorithm carries out a systematic search, looking for solutions to some problem and it allows one to find the way through the network without danger of going round and round in circles.

In our study we utilized all the three previous algorithms with some modifications that may simplify solution or save time. We propose two algorithms *ALG1* and *ALG2*. The former depends thoroughly upon depth-first search together with the help of backtracking. In this algorithm depth-first search is utilized to look in the given network to check if a connection is found between all the k -terminal nodes. If yes then no need to continue in the search, and a decision is made to be connected. In the latter one, breadth-first search was used where we take the first k -terminal node of interest then we find the shortest-path between it and the next k -terminal node. If a path exists then the recently reached k -node is considered to be the new source while the next k -node is the destination and backtracking is done, and so on until ensuring the connection of all the k -terminals of interest, then the decision is made to be connected. If at any stage, no such path exists, then the decision is made to be disconnected.

2.5 Reliability Evaluation : General Survey

Many algorithms were published to solve the k -terminal reliability problem, most of them depend on evaluation of reliability polynomial, utilizing reduction and factorizing theorems. (Wood,1986) states a recursive factoring algorithm for exact computation of this problem, and it applies the formula:

$$R_K(G) = p_i R(G_K * e_i) + q_i R(G_K - e_i)$$

$$p_i = 1 - q_i = \text{reliability of edge } e_i$$

$$G_K * e_i = G_K \text{ with edge } e_i \text{ being contracted}$$

$$G_K - e_i = G_K \text{ with edge } e_i \text{ being deleted}$$

$$R_K(G) = \text{The } k\text{-terminal reliability.}$$

A contraction of an edge means to unify both the end vertices of that edge in one vertex while keeping all edges incident to both vertices as incident to the new vertex.

Although other algorithms may have differences, such as that found in (Baily and Kulkarni, 1986), Wood algorithm follows the main general procedure and hence we will take it with more specialty. The main idea is to begin with some graph, then do any possible reductions and/or factoring keeping in mind that every alteration on each graph has its equivalent variation in the reliability polynomial of the original graph. For example, if two edges e_a and e_b are parallel, i.e they have the same end vertices, then both of them may be replaced by just one edge e_c such that $p_c = 1 - q_a q_b$. If they are in series, i.e they have one common vertex that is of degree two then they may be replaced by an edge e_c such that $p_c = p_a p_b$. There are many other types of reductions, such as degree-2 reduction, polygon-to-

chain reductions, bridge contraction, degree-3 reduction, etc. (Resende, 1986), (Page and Perry, 1994).

To get a clearer idea, let us discuss the example discussed by Wood, redrawn in Figure 2.4, from which we can state the k -terminal reliability, with $k=2$ by the following relationship:

$$R_k(G) = p_4 [(1 - q_3 q_5)(1 - q_1 q_2)(1 - q_6 q_7) + q_3 q_5 (p_1 p_7 + p_2 p_6 - p_1 p_2 p_6 p_7)] + q_4 [p_1 p_3 + p_2 - p_1 p_2 p_3][p_5 p_7 + p_6 - p_5 p_6 p_7]$$

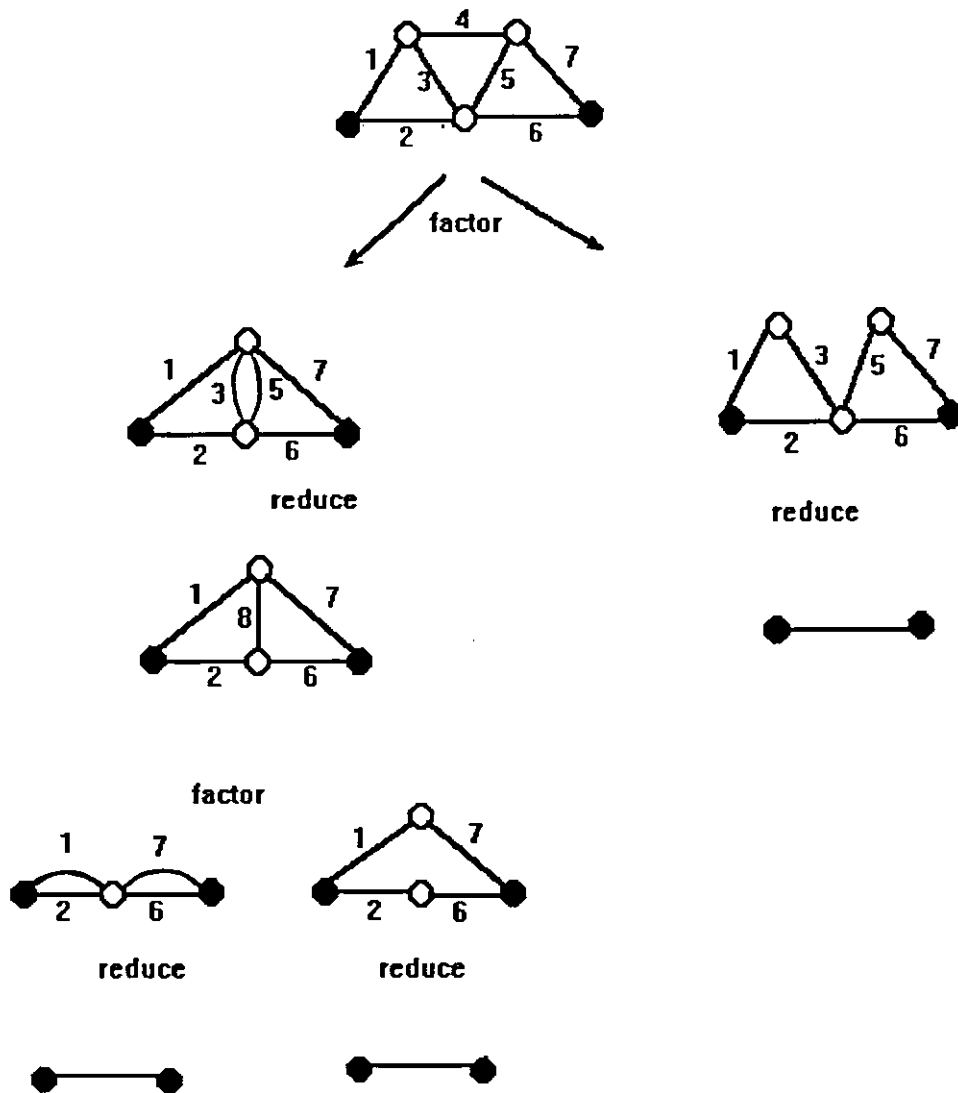


Figure 2.4 : Reduce and Factor Process

Hence, the exact k -terminal reliability is found. However, there is an important point to note, that is the process of factoring and reduction may result in a different tree if the edge, say e_4 , is not chosen to be contracted and deleted in the first step, and hence the number of factorizations may also differ. This imposes on us to use some strategy to determine the best edge to contract that results in minimum number of leaves in the search structure, i.e that minimizes the number of factorizations needed.

Another similar algorithm is that published by Page and Perry, which uses the factoring theorem to find the k -terminal reliability. We will name it as *PRFA* for future references (Page and Perry, 1988).

However, even if a program takes into its account all reductions mentioned above, with an optimum choice of the edge to be contracted, it is still not general, i.e it cannot be applied to all possible topologies of networks. Also, it cannot be used for large networks since it may take long time periods to find the reliability, if it can be found. Hence the need for another method, although may be not exact, that can be general is of special importance, and this is the goal of our work.

(Ayoub and Shahbaz, 1996) illustrates an algorithm, called *MAP*, that deals with a probabilistic graph of N nodes and L links, link l_i fails with probability p_i . In the following is a brief description of the methodology used in this algorithm.

Given a graph G , its nodes and links are first numbered in a certain order. The links are split and numbered into those of a spanning tree G_T , and those remaining are extracted from G as follows: as nodes are numbered from 1 to N , the links of G_T are numbered l_i ; $i = 1, 2, \dots, N-1$ such that $f(l_i) \leq i$ and $t(l_i) = i + 1$. When a numbered node is reached that does not have an adjacent node, he backtracks through the numbered nodes until a node that has an adjacent node is reached. As the N th node is numbered, $N-1$ links of G_T are

also numbered. The case $f(l_i) < i$ occurs only when $t(l_i)$ is selected after backtracking. The remaining links of G are numbered arbitrarily l_i ; $i = N, N+1, \dots, L$. For the sample network G_j , the sample state vector $X_j = \{ l_1, l_2, \dots, l_L \}$ is generated.

Before stating the *MAP* algorithm, let us clarify some notations used in it.

ρ = number of components in G_j

$g[n_i]$ = the component that node n_i belongs to

$\theta[g_i]$ = number of nodes in component g_i

$f(l_i), t(l_i)$ = end vertices of edge l_i

| | |
|---|--|
| Input: State vector of sample G_j (X_j) | |
| Output: k -terminals are all in one component or not | |
| Step0: initialization | |
| $\rho = 1, g[f(l_1)] = 1, \theta[g_i] = 1, i = 1$ | |
| Step1: (phase 1) if (l_i) is up | |
| $g[t(l_i)] = g[f(l_i)]$ | |
| $\theta[g[f(l_i)]] = \theta[g[f(l_i)]] + 1$ | |
| Step2: if (l_i) is down | |
| $\rho = \rho + 1$ | |
| $g[t(l_i)] = g_\rho$ | |
| $\theta[g_\rho] = 1$ | |
| Step3: $i = i + 1$ | |
| if $i < N$ go to step 1 | |
| if $i = N$ and $\rho = 1$, go to step 9 | |
| if $i = N$ and $\rho > 1$, go to step 4 | |
| Step4: (phase 2) $T[g_k] = g_k$; $k = 1, 2, \dots, \rho$ (Define tables) | |
| Step5: if l_i is down go to step 7 | |

| | |
|--------|---|
| | $S = T[g[f(l_i)]]$ |
| | $B = T[g[t(l_i)]]$ |
| | if $S = B$ go to step 7 |
| Step6: | $\theta[S] = \theta[S] + \theta[B]$ |
| | $\theta[B] = 0$ |
| | for $m = 1, 2, \dots, \rho - 1$ |
| | if $T[g_m] = B$, set $T[g_m] = S$ |
| | $\rho = \rho - 1$ |
| Step7: | $i = i + 1$ |
| | if $i \leq L$ go to step 5 |
| Step8: | for nodes $n_i, i=1, 2, \dots, N$ |
| | $g[n_i] = T[g_n]$ |
| Step9: | check if all the k terminals lie in one component |
| | if yes then $\phi(X_j) = 0$ |
| | else $\phi(X_j) = 1$ |
| Stop. | |

The last algorithm that we will discuss here is the Modified Dotson Algorithm (Yoo and Deo,1988) which is a terminal-pair reliability algorithm, a special case of the k -terminal reliability problem. In this algorithm the network is modeled by an adjacency matrix, then the breadth-first search is employed so as to find the shortest paths between the source and destination. A reliability polynomial is derived next and used to calculate the terminal-pair reliability. The vertices are numbered from 1 to n and edges from 1 to m , the network is represented by its adjacency matrix.

Chapter 3

PROPOSED RELIABILITY ALGORITHMS

3.1 Assumptions:

As mentioned previously, computing the reliability of a generic network is computationally difficult, and an exact solution is unattainable for many problems. Faced with this computational difficulties some assumptions are usually considered so as the reliability evaluation methods become applicable. The work developed in this thesis will also take into account these assumptions, which are:

- The location of each network node is given
- Nodes are perfectly reliable
- Each p_i is fixed and known
- Each link is bi-directional
- There are no redundant links in the network
- Links are either operational (up), or failed (down).

- The failures of links are mutually statistically independent
- The k -terminal set is specified.

However, there is still another problem which is unattainability of exact solution for many problems. Moreover, the long computational time is another problem. Faced with this problem, we should look for an estimate of the reliability measure that gives an accurate, although not exact, value computed in short time period while keeping the used methodology as general as possible. This approach is followed in this thesis.

After stating the above assumptions, some notations that are used in our discussion must be presented next. These are

$G = (N, M)$ = The graph to be investigated with N nodes, or vertices, and M links, or edges.

K = Number of terminals in the k -terminal set.

$R_K(G)$ = The k -terminal reliability.

p_i = Reliability of edge e_i .

NS = Number of executions or samples.

ns = The sample number (variable)

ϕ = Result of the ns search (variable)

phi = Counter of connected samples (variable)

GS = Sample network (variable)

NS , ns , ϕ , phi , and GS are explained more in the next sections.

3.2 Monte Carlo Simulation :

One important method of simulation is the Monte Carlo simulation. In this method, to calculate the reliability of a network, a number of executions,

called samples, is done for the original network taking into account probabilistic variations. In other words, for each execution, random errors, whose percentage depends on each specified edge reliability, are generated leading to a new network that may have different number of edges and hence different topology. For this sample network we want to check if our k terminals of interest are connected by at least one path of working edges or not. If yes then this sample network results in a connected graph and hence the number of connected samples is increased by one. Otherwise then the number of unconnected samples is incremented by one, and so on until sufficient number of samples is taken that gives a sufficiently accurate reliability measure.

A single execution represents only one of the many possible pathways in the model. Thus for a single execution we have no way to tell whether the result is an extreme case or about average. Repeated executions of the simulation provide distributions of values at each state and a distribution of pathways. From these distributions we can infer what the averages, the extremes and the shapes of the distributions if they were of our interest.

How reliable are the estimates we make from a limited number of executions? Two executions may give us a great deal more information than a single execution. One hundred executions may give us hints of the extremes and of the shapes of the ultimate distribution. A thousand executions may give us results that approximate smooth curves, but a second thousand may give different curves. These possibilities raise the very practical question: How long should we run a simulation model? If we continue executing the simulation process, we hopefully will get, after some number of executions, some results that will be fixed around some asymptote. Hence one may do number of executions that are larger than recommended and then starts to lower this number to minimize it while still

giving results within the specified accuracy and/or a sufficient specified form (Barton, 1970).

A sufficient number of executions needs a stopping criteria. Such a criteria may depend on the accuracy, i.e when the difference between the approximate value and the exact one is less than some pre-defined error. This method needs to know the exact measure which is not always available. Another criteria depends on the shape of variations in the estimated measure, i.e. it compares between successive estimated measures and if the differences between them is within a specified value then it stops executions. In mathematical terms:

$$R_K(m+n) - R_K(m) \leq d_1$$

where d_1 is an appropriate fraction that indicates that the estimated reliability measure is almost constant and any additional executions may not add new information of substantial importance. The value of n may be any integer greater than or equal to one. In some cases the percentage change is calculated to serve as a stopping criteria, using the following equation:

$$\frac{R_K(m+n) - R_K(m)}{R_K(m+n)} \leq d_2\%$$

In our work, for each sample done, a random number is generated for each edge in the sample. If this number is higher than the reliability of that edge then it is considered to be a failed connection. Otherwise, nothing happens for that specified edge. The search procedures are utilized next to check the connectedness of the k -terminal nodes. One important point here is that each sample network is statistically independent of other sample networks.

After doing sufficient executions then the k -terminal reliability is simply the mean of the probability distribution $Y(ns)$ on NS . Then consider NS as an urn from which a ball ns can be drawn with probability $p(ns)$. Now let Z denotes the mean value of the random variable $Y(ns)$ where

$$Y(ns) = \frac{z(ns)}{p(ns)}$$

and

$$Z = \sum_{ns=1, \dots, NS} z(ns) = \sum_{ns=1, \dots, NS} p(ns) \frac{z(ns)}{p(ns)} = E\{Y\}$$

The variance and coefficient of variation of Y are

$$\text{var}\{Y\} = \sum_{ns=1, \dots, NS} p(ns) Y^2(ns) - E^2\{Y\}$$

$$\delta\{Y\} = \sqrt{\sum_{ns=1, \dots, NS} \frac{p(ns) Y^2(ns)}{E\{Y\}^2} - 1}$$

From basic statistics, for N statistically independent choices from NS with probabilities $p(u)$; we have an unbiased estimate of $E\{Y\}$. Hence the reliability measure is found by the following equation

$$\hat{Y}_s = \frac{1}{N} \sum_{ns=1}^N Y(ns)$$

with variance and coefficient of variation being equal to (Elperin, Gertsbakh and Lomonosov, 1991)

$$\text{Var}\{\hat{Y}_s\} = \frac{1}{N} \text{Var}\{Y\}$$

$$\delta_s = \frac{\delta_Y}{\sqrt{N}}$$

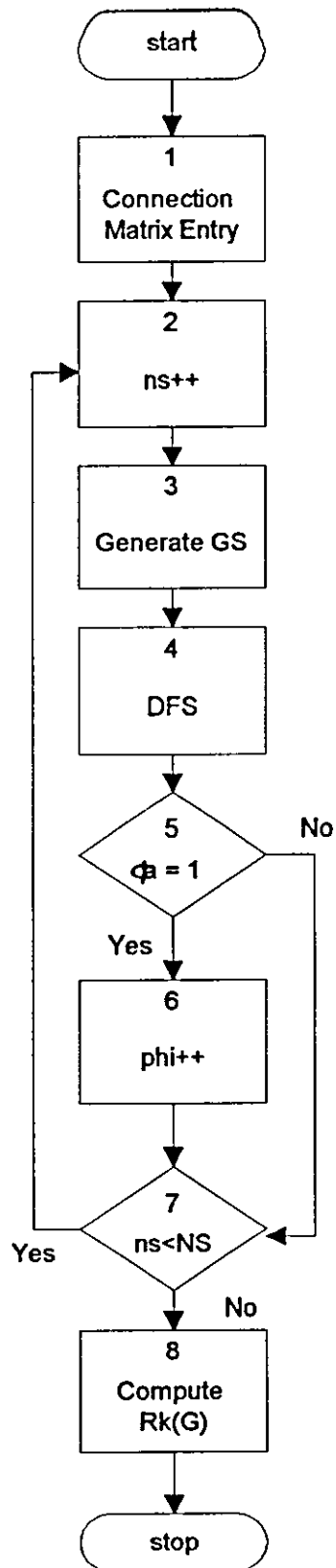
All previously published results utilized graph searching techniques other than the depth-first and the breadth-first. Our proposed algorithms in this thesis are the first to use these two techniques in the k -terminal reliability problem.

3.3 ALGORITHM 1, *ALGI*

This algorithm depends thoroughly on the depth-first search with the aid of Crude Monte Carlo simulation. Before discussing the specifics of this algorithm, let us take a look at the headlines of this algorithm shown by the flow chart in Figure 3.1.

The first block is concerned with the mathematical representation of the graph to be studied. The network under study should be presented to the program in the adjacency- or connection-matrix representation. Hence programs that convert from $f-t$ representation and/or incidence-matrix representation to the connection matrix representation were written and are given in Appendices 1 and 2 respectively.

Why to use the connection matrix representation instead of the others? There are three points of considerations that make this choice be desirable. First, it uses the smallest dimension ($V \times V$) among other matrix representations. For the incidence matrix the dimension is ($V \times E$), where usually the number of edges E is greater than the number of vertices V . Hence the connection matrix is smaller in size. This smaller size leads to a lower number of locations to be searched in the search process and hence less time spent during each execution. The binary nature of the connection matrix makes it the optimum choice when considering the memory size with respect to all other matrix representations. The $f-t$ representation deals with two arrays each of length E and entries that may take values from 1 to E . Thus, the required number of bits depends on E , while in a connection matrix each entry requires just one bit and hence saving memory space.

Figure 3.1: Flow chart for *ALG1*

Another attractive characteristic of the connection matrix is its symmetry around its diagonal. This characteristic feature enables us to get all the information about the network under study from less than half of this matrix. Only the entries that lie over the diagonal of the connection matrix are read since the entries in the lower half are just a mirrored image of the upper one. This lowers the number of entries to be investigated during each execution run, and hence execution time. This property reduces the number of investigated entries from V^2 to $V(V-1)/2$.

The second block in the flow chart of Figure 3.1 works as a counter to maintain for the number of samples or executions. The third block generates a sample network for each execution; that is, it takes every edge in the graph G , and generates a random number p for it. If this p is greater than or equal to the reliability of that edge p_i , then it is considered a failed edge, or down, otherwise it is considered to be connected or up. This operation is done for all edges in G resulting in a new network that have the same number of vertices but possibly different number of edges. We call this network a sample graph GS .

After generating the sample, a test should be made to see if the k -terminal nodes of interest are still connected with each other or not, in spite of the deletion of some edges. This is what the fourth block does. It utilizes the depth-first search method to do so. It starts with a vertex that belongs to the k -terminal set then continue to search until all k -terminal nodes are explored, or no path is found to connect them. The former case leads to a 'connected' decision, while the latter leads to a 'disconnected' decision. A 'connected' sample assigns the value one for ϕ , while a 'disconnected' sample assigns a zero value for it, and hence no change in phi .

This procedure constitutes one execution run, which is a point in the NS execution points. Another execution run is done, and so on to accumulate NS

points. After doing all NS execution runs, the k -terminal reliability is the value of ϕ divided by the number of samples NS , and hence the reliability measure is estimated.

One further point that we did not mention in the previous discussion, that is the utilization of backtracking method, if needed, in the search process. Backtracking technique is used to avoid looping and hence decrease the search time.

After this general look, we will take an example and discuss it numerically. We will use the network shown in Figure 3.2, and do two samples that are generated by the computer to see the results. The connection matrix of G is:

$$G = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Now, and after specifying GS , a search is done. It starts at the k -terminal that holds the name v_1 and the k -terminal counter counts one, starting at adjacent vertices. It continues to v_2 then v_3 and v_8 . At v_8 there is an edge that leads to an ancestor which is constructed then it continues to v_7 , v_6 and v_5 ; there is no adjacent vertices found so far hence backtracks to v_8 from which an edge that leads to v_9 is found. Since v_9 is a k -terminal node, the k -terminal counter is incremented. Now no adjacent vertices; backtracking is done but it does not help since there is still another k -terminal v_4 that is isolated from the others. Hence the result is a disconnected decision, i.e. $\phi = 0$, and the value of phi is not incremented.

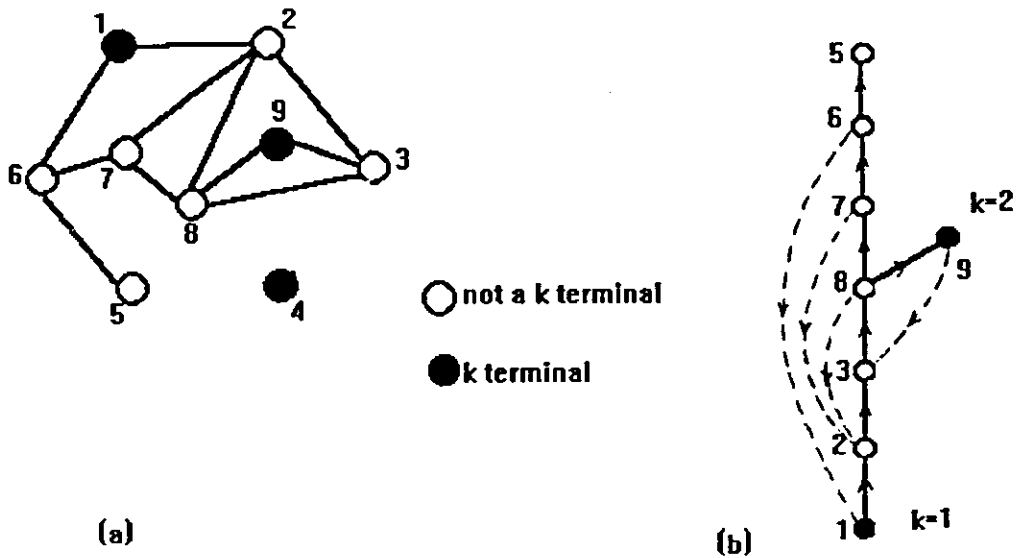


Figure 3.3: *ALGI* solution for a sample network (a) Sample network, GS .

(b) Search tree.

The second sample to discuss deletes edges 1, 4, 5, 8, and 13 assuming that all of them are down, resulting in GS shown in Figure 3.4 and represented by:

$$GS = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 & 1 & 1 \\ & & & 0 & 0 & 0 & 0 & 0 \\ & & & & 0 & 0 & 0 & 0 \\ & & & & & 1 & 0 & 0 \\ & & & & & & 1 & 0 \\ & & & & & & & 1 \end{bmatrix}$$

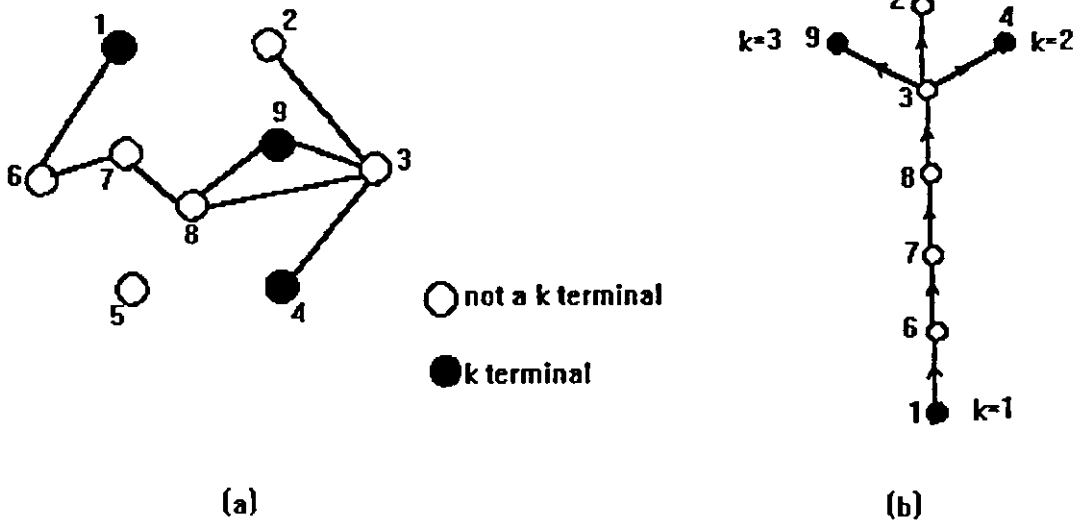


Figure 3.4: *ALG1* solution for a sample network (a) Sample network GS .
(b) Search tree.

The search starts at v_1 , the first k -terminal to be explored, and hence the k -terminal counter counts one. Then through the depth-first search method, it

continues to v_6, v_7, v_8 and v_3 . Then, from vertices adjacent to v_3 , v_2 is chosen leading to a point where no new adjacent edges are found so we backtrack to v_3 . At v_3 edge (3,4) leads to a k -terminal, hence the k -terminal counter is incremented. Then we backtrack to v_3 from which we go to v_9 , the third k -terminal of interest. The k -terminal counter is incremented a third time and is checked for reaching its limit, which is three for this example. The program is then terminate the search, although some vertices may still be unexplored, and give a 'connected' decision, i.e. $\phi = 1$. The value of phi is now incremented by one.

Note that if we take a quick look at GS in the second sample, it tells us that the graph is disconnected since vertex v_5 is isolated from the whole graph. However, the search gave a connected decision since we are interested in terminals v_1, v_4 and v_9 only and all these three vertices lie in one component. Hence the decision was 'connected' after the search was ended.

These two samples are just examples contributing some information about the reliability of the set of k nodes. The computer continues to do the NS execution runs given the required accuracy of the reliability estimate, in possible finite time duration.

Now, after discussing this algorithm, we will present it in a formal manner to simply writing it as a computer program.

| |
|--|
| <p>Input: An undirected network with a known reliability p_i for each edge.</p> <p>Output: The k-terminal reliability $R_K(G)$</p> |
| <p>Step0: initialization</p> <p style="padding-left: 40px;">$0 \rightarrow ns, 0 \rightarrow phi$</p> <p>Step1: enter the connection matrix of the network.</p> |

Step2: $ns+1 \rightarrow ns$

generate random number p for each edge e_i in G

if $p \geq p_i$ then assume e_i to be down

else e_i is up

Step3: $0 \rightarrow i, 0 \rightarrow k, 1 \rightarrow no$

select a k -terminal to be $w(1)$

Step4: $w(no) \rightarrow v$

$no + 1 \rightarrow no$

if v is one of the k terminals

then $k + 1 \rightarrow k$

if $k = K$ then go to step 7.

Step5: construct all $v \rightarrow w_i$ arcs

Step6: construct single $v \rightarrow w_i$

if possible then $w_i \rightarrow w(no)$, go to step 4

else then backtrack until finding an appropriate $w(i)$

if possible then $w(i) \rightarrow v$, go to step 6

else go to step 8.

Step7: $phi + 1 \rightarrow phi$

Step8: if $ns < NS$ go to step 2.

else $(phi / NS) \rightarrow R_K(G)$

Stop.

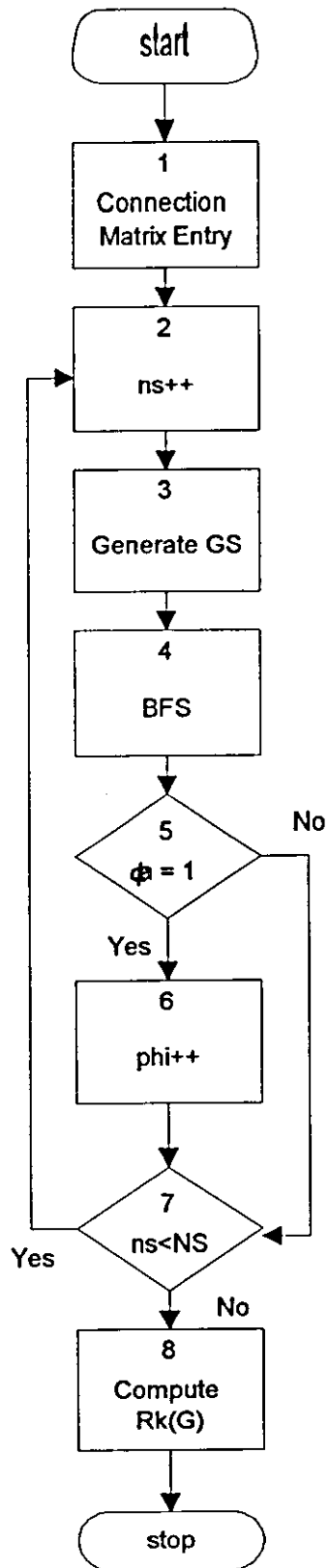
3.4 Algorithm 2, *ALG2*

This algorithm utilizes the breadth-first search together with Crude Monte Carlo simulation so as to estimate the k -terminal reliability. The flow chart of this algorithm is shown in Figure 3.5. It looks similar to that of *ALG1*, however the following discussion will show the differences.

The first three blocks do the same job as that of *ALG1*. The graph is entered in the connection-matrix form then simulation is started. The variable ns works as a counter, and each generated sample is saved in GS . The fourth block does the search process that will determine whether the k terminals are connected or not. It starts at the first node in the k -terminal set and searches for a path, if any, to the second node in the k -terminal set. If a path exists it continues to the third node in the k -terminal set, otherwise the k terminals do not lie in a single component and hence the decision is made to be 'disconnected'. Note here that there is, in a way, a priority in the search pattern and this priority is given for the k terminals of interest, a way that may, in some cases, save time and increase efficiency in the computer program.

The chart continues to operate with its eight blocks similar to *ALG1*. The details are best explained by the use of numerical example. Let us take the same example solved by *ALG1* in Section 3.3, redrawn here in Figure 3.6. We will use the same connection matrix G . In this figure, the search trees are drawn assuming no failures in the network.

The first tree is built by finding a path between terminals one and four. Since it is connected then the search between nodes four and nine is done next, and the decision comes out to be a 'connected' k -terminal nodes.

Figure 3.5: Flow chart for *ALG2*

Now let us take two samples to be simulated by the computer program. The first sample assumes edges 3 and 4 failed, resulting in the sample shown in Figure 3.7(a)

The search tree followed by the computer program is shown in Figure 3.7(b). It starts at vertex v_1 , the first node in the k -terminal set, and looks for the second k -terminal v_4 . There are two adjacent vertices for vertex v_1 , namely v_2 and v_6 . They are numbered and the computer then chooses v_2 to be the next node v . It has v_3, v_7 and v_8 as adjacent nodes. Then the new edge that should be constructed is $(6,5)$. After finishing this level the next v will be v_3 . Then from vertex v_3 , the only adjacent vertices are v_8 and v_9 .

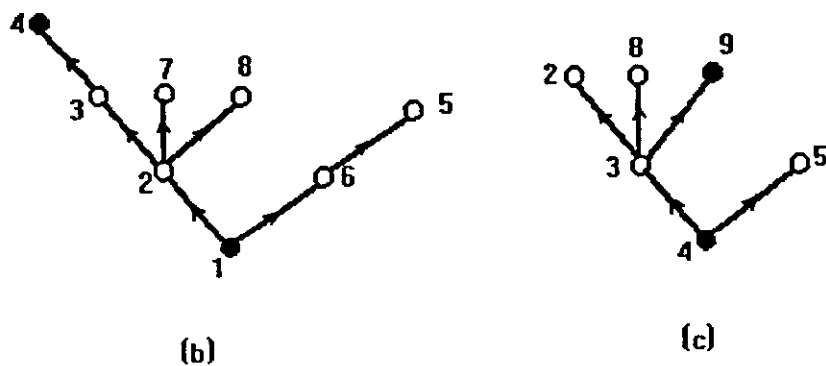
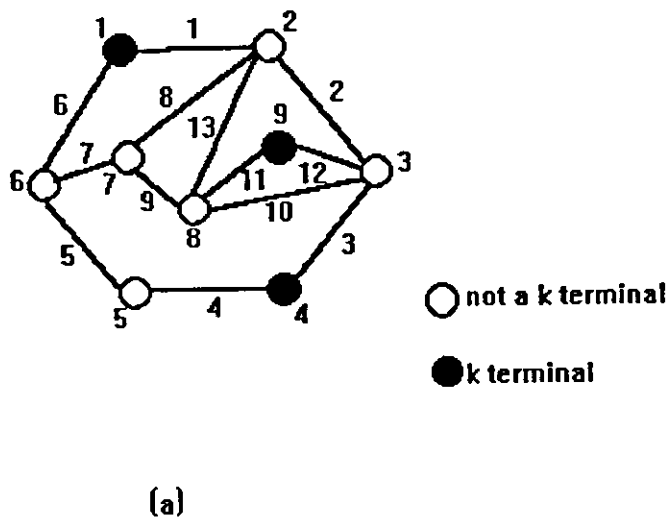


Figure 3.6: *ALG2* solution for a sample network (a) k -terminal reliability problem (b) Search tree between 1 and 4 (c) Search tree between nodes 4 and 9

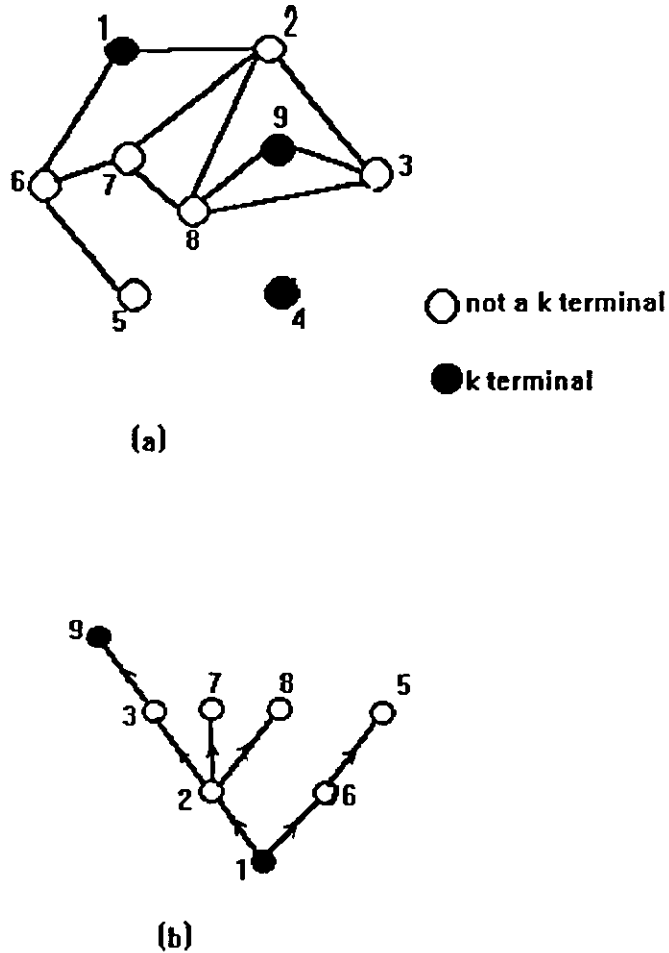
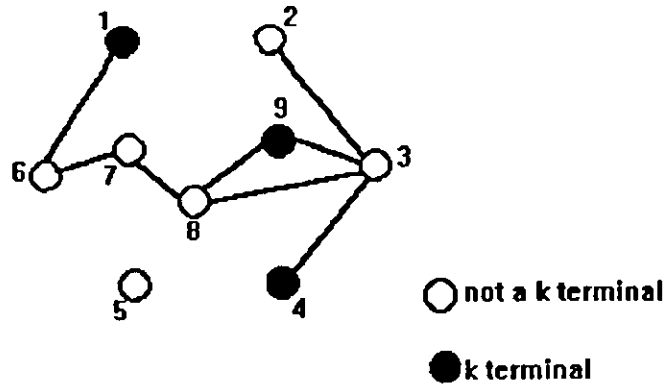


Figure 3.7: *ALG2* solution for a sample network

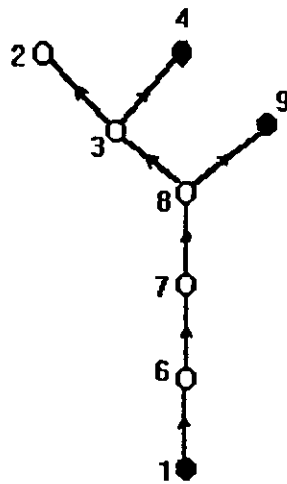
(a) Sample network, *GS* (b) Search tree.

So far the required vertex whose number is four is still not found; while all possible vertices were numbered. This case cannot occur except if that vertex lies in another component leading to the ‘disconnected’ decision. The decision was made without any need to continue to check the connections between other k -terminal nodes, a point that may result in a shorter execution time.

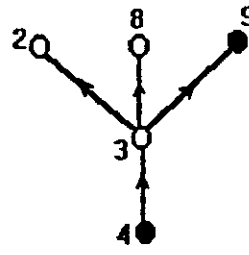
The second sample to discuss assumes that edges 1, 4, 5, 8 and 13 have failed resulting in *GS* shown in Figure 3.8(a)



(a)



(b)



(c)

Figure 3.8: *ALG2* solution for a sample network(a) Sample network, *GS*

(b) Search tree between 1 and 4

(c) Search tree between 4 and 9

At first, the program checks if a path exists between nodes one and four, the first two k -terminal nodes. The search tree is drawn in Figure 3.8(b),

leading to the 'connected' decision. The second step is to check if a path exists between nodes four and nine, the last k -terminal nodes. Hence another search tree is built shown in Figure 3.8(c). Since all the k -terminal nodes are reached then a 'connected' decision is made leading to the assignment of value one to ϕ .

These two samples are just examples, to show how the program proceeds when attacking different situations. The computer program continues in the same manner until the required number of samples is reached giving a sufficient and accurate estimation of the network reliability.

The following pseudo code represents , the *ALG2* algorithm:

| |
|---|
| <p>Input: An undirected network with a known reliability p_i for each edge.</p> <p>Output: The k-terminal reliability $R_K(G)$</p> |
| <p>Step0: initialization</p> <p style="padding-left: 40px;">$0 \rightarrow ns, 0 \rightarrow phi$</p> <p>Step1: enter the connection matrix of the network</p> <p>Step2: $ns + 1 \rightarrow ns, 1 \rightarrow no$</p> <p style="padding-left: 40px;">generate random number p for each edge e_i in G</p> <p style="padding-left: 40px;">if $p \geq p_i$ then assume e_i to be down</p> <p style="padding-left: 40px;">else e_i is up</p> <p>Step3: choose two k-terminal nodes and name them as s and t</p> <p>Step4: if t is adjacent to s then go to step5</p> <p style="padding-left: 40px;">else for all w_i adjacent to s,</p> <p style="padding-left: 80px;">construct $s \rightarrow w_i$</p> <p style="padding-left: 40px;">if no such edge then go to step7</p> <p style="padding-left: 40px;">else choose $w_i \rightarrow s$ then go to step4</p> <p>Step5: $t \rightarrow s$</p> |

603330

choose a new node from the k -terminal nodes and name it t
if all the k terminals are reached then go to step6
else go to step4

Step6: $phi + 1 \rightarrow phi$

Step7: if $ns < NS$ go to step2.

Step8: else $(phi / NS) \rightarrow R_k(G)$

Stop.

Chapter 4

RESULTS

Our two proposed algorithms were programmed in *C* language using *Turbo C++* compiler. A *100 Mhz PC* was used to develop and execute these programs. The two programs are shown in Appendices 3 and 4.

4.1 Algorithm Performance:

Seven networks were first examined to assess the algorithm performance according to reliability measure, number of samples needed and time of execution. For each network we took two different cases: one with a high edge reliability p_i that equals to 0.9, and the other one is taken for low p_i equals 0.4.

The seven networks were given the names : A_1 , A_2 , A_3 , A_4 , A_5 , A_6 , and A_7 . They were created using a methodology of building networks for which the input is the number of nodes N and the number of edges M , while the output is a certain topology. For example, A_1 network is $A_1(6,9)$ takes

the topology shown in Figure 4.1. It distributes the nodes in a polygon form. The first N edges are built to draw the polygon, then in the second cycle, the edges are built to connect vertices 1 with 3, 2 with 4, etc. In the third cycle it connects vertices 1 with 4, 2 with 5, etc., and so on.

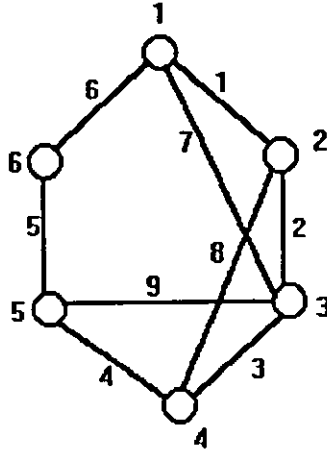


Figure 4.1: A simple, undirected graph, $A_1 = (6,9)$

A program for building networks using this method was written in C language is shown in Appendix 5. Using this method, we generated all the A -networks: $A_1 (6,9)$, $A_2 (10,15)$, $A_3 (15,22)$, $A_4 (20,30)$, $A_5 (30,45)$, $A_6 (50,75)$ and $A_7 (70,105)$.

In the f - t form these networks are written as follows:

$$A_1 = (6,9)$$

$$f = [1,2,3,4,5,6,1,2,3]$$

$$t = [2,3,4,5,6,1,3,4,5]$$

$$A_2 = (10,15)$$

$$f = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5]$$

$$t = [2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 3, 4, 5, 6, 7]$$

$$A_3 = (15,22)$$

$$f = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1, 2, 3, 4, 5, 6, 7]$$

$$t = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1, 3, 4, 5, 6, 7, 8, 9]$$

$$A_4 = (20, 30)$$

$$f = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

$$t = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$$

$$A_5 = (30, 45)$$

$$f = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$$

$$t = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]$$

$$A_6 = (50, 75)$$

$$f = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]$$

$$t = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]$$

$$A_7 = (70, 105)$$

$f = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35]$

$t = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37]$

As seen above, various network sizes were taken that spread from small ones, like A_1 , to medium ones, like A_4 , to large networks, like A_7 .

The number of executions, NS , was also varied so as to investigate its effect on the shape of variations in the reliability measure and execution times. Table 4.1 shows the results for $p_i = 0.9$, while Table 4.2 shows the results for $p_i = 0.4$. Here all the nodes are assumed to be k terminals, hence we find the all-terminal reliability.

| NS | A1 | | | A2 | | | A3 | | | A4 | | |
|-------|----------|----|----|----------|----|----|----------|----|----|----------|----|----|
| | Rk | t1 | t2 | Rk | t1 | t2 | Rk | t1 | t2 | Rk | t1 | t2 |
| 100 | 0.99 | 0 | 0 | 0.89 | 0 | 0 | 0.82 | 0 | 0 | 0.75 | 0 | 0 |
| 1000 | 0.987 | 0 | 0 | 0.922 | 0 | 0 | 0.848 | 1 | 0 | 0.754 | 0 | 1 |
| 2000 | 0.986 | 0 | 0 | 0.935 | 0 | 0 | 0.8415 | 1 | 1 | 0.752 | 1 | 1 |
| 3000 | 0.987 | 0 | 0 | 0.935333 | 1 | 1 | 0.840333 | 1 | 1 | 0.755333 | 2 | 2 |
| 4000 | 0.98775 | 1 | 0 | 0.93525 | 1 | 1 | 0.84025 | 2 | 2 | 0.755 | 2 | 3 |
| 5000 | 0.9862 | 1 | 0 | 0.9334 | 1 | 1 | 0.838 | 2 | 2 | 0.7564 | 3 | 4 |
| 10000 | 0.9853 | 1 | 1 | 0.9335 | 2 | 2 | 0.8397 | 44 | 5 | 0.7608 | 6 | 7 |
| 15000 | 0.9832 | 1 | 2 | 0.934 | 3 | 3 | 0.8396 | 5 | 6 | 0.7598 | 9 | 10 |
| 20000 | 0.9839 | 1 | 2 | 0.93365 | 3 | 4 | 0.8395 | 7 | 8 | 0.7608 | 13 | 13 |
| 30000 | 0.9839 | 3 | 3 | 0.935767 | 5 | 6 | 0.839333 | 11 | 13 | 0.7596 | 18 | 21 |
| 40000 | 0.984125 | 3 | 4 | 0.9372 | 7 | 9 | 0.840275 | 15 | 16 | 0.759725 | 25 | 27 |
| 50000 | 0.98386 | 4 | 5 | 0.93806 | 10 | 11 | 0.83928 | 18 | 21 | 0.75978 | 30 | 34 |

| NS | A5 | | | A6 | | | A7 | | |
|-------|----------|----|----|----------|-----|-----|----------|-----|-----|
| | Rk | t1 | t2 | Rk | t1 | t2 | Rk | t1 | t2 |
| 100 | 0.6 | 1 | 0 | 0.29 | 0 | 1 | 0.13 | 0 | 0 |
| 1000 | 0.591 | 1 | 1 | 0.284 | 3 | 3 | 0.141 | 6 | 6 |
| 2000 | 0.5635 | 3 | 3 | 0.2745 | 7 | 7 | 0.1285 | 13 | 12 |
| 3000 | 0.563333 | 4 | 4 | 0.283333 | 10 | 10 | 0.129 | 20 | 18 |
| 4000 | 0.56475 | 5 | 5 | 0.28 | 14 | 13 | 0.129 | 26 | 24 |
| 5000 | 0.5634 | 7 | 7 | 0.277 | 17 | 17 | 0.131 | 33 | 30 |
| 10000 | 0.5631 | 13 | 14 | 0.2768 | 35 | 33 | 0.1249 | 66 | 60 |
| 15000 | 0.5684 | 20 | 20 | 0.278467 | 52 | 50 | 0.125 | 99 | 90 |
| 20000 | 0.56775 | 26 | 28 | 0.28 | 70 | 67 | 0.1243 | 132 | 121 |
| 30000 | 0.5675 | 39 | 41 | 0.2822 | 104 | 101 | 0.126433 | 198 | 181 |
| 40000 | 0.56675 | 53 | 55 | 0.28525 | 139 | 134 | 0.12755 | 264 | 241 |
| 50000 | 0.56864 | 66 | 68 | 0.28378 | 174 | 168 | 0.12755 | 331 | 302 |

Table 4.1: Reliability and Time vs NS for A-networks, $p_i = 0.9$ (Time is in seconds)

| NS | A1 | | | A2 | | | A3 | | | A4 | | |
|-------|----------|----|----|----------|----|----|----------|----|----|---------|----|----|
| | Rk | t1 | t2 | Rk | t1 | t2 | Rk | t1 | t2 | Rk | t1 | t2 |
| 100 | 0.16 | 0 | 0 | 0.05 | 0 | 0 | 0.01 | 0 | 0 | 0 | 0 | 0 |
| 1000 | 0.18 | 0 | 0 | 0.025 | 0 | 0 | 0.002 | 0 | 1 | 0 | 1 | 0 |
| 2000 | 0.162 | 0 | 0 | 0.029 | 0 | 1 | 0.001 | 1 | 0 | 0 | 1 | 1 |
| 3000 | 0.166667 | 0 | 0 | 0.026 | 1 | 1 | 0.001 | 0 | 1 | 0 | 1 | 1 |
| 4000 | 0.168 | 1 | 1 | 0.026 | 0 | 1 | 0.001 | 1 | 1 | 0 | 1 | 2 |
| 5000 | 0.1674 | 0 | 1 | 0.0248 | 0 | 1 | 0.001 | 2 | 1 | 0 | 2 | 2 |
| 10000 | 0.1746 | 1 | 1 | 0.024 | 2 | 1 | 0.0011 | 3 | 3 | 0 | 4 | 4 |
| 15000 | 0.174467 | 2 | 2 | 0.0256 | 3 | 2 | 0.001267 | 5 | 4 | 0 | 6 | 6 |
| 20000 | 0.1758 | 2 | 2 | 0.02585 | 3 | 4 | 0.00125 | 5 | 5 | 0 | 8 | 8 |
| 30000 | 0.176333 | 3 | 2 | 0.026067 | 5 | 5 | 0.001167 | 8 | 8 | 0 | 12 | 12 |
| 40000 | 0.176 | 4 | 4 | 0.02605 | 6 | 7 | 0.00125 | 11 | 11 | 0.00005 | 16 | 16 |
| 50000 | 0.1751 | 5 | 5 | 0.02622 | 8 | 8 | 0.00136 | 13 | 14 | 0.00004 | 21 | 20 |

| NS | A5 | | | A6 | | | A7 | | |
|-------|----|----|----|----|-----|-----|----|-----|-----|
| | Rk | t1 | t2 | Rk | t1 | t2 | Rk | t1 | t2 |
| 100 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1000 | 0 | 1 | 1 | 0 | 3 | 3 | 0 | 2 | 2 |
| 2000 | 0 | 1 | 1 | 0 | 7 | 7 | 0 | 5 | 4 |
| 3000 | 0 | 2 | 1 | 0 | 10 | 10 | 0 | 8 | 7 |
| 4000 | 0 | 3 | 2 | 0 | 14 | 13 | 0 | 11 | 8 |
| 5000 | 0 | 3 | 3 | 0 | 17 | 17 | 0 | 13 | 11 |
| 10000 | 0 | 7 | 6 | 0 | 35 | 33 | 0 | 25 | 22 |
| 15000 | 0 | 11 | 9 | 0 | 52 | 50 | 0 | 38 | 31 |
| 20000 | 0 | 14 | 13 | 0 | 70 | 67 | 0 | 51 | 43 |
| 30000 | 0 | 21 | 20 | 0 | 104 | 101 | 0 | 76 | 63 |
| 40000 | 0 | 28 | 26 | 0 | 139 | 134 | 0 | 102 | 85 |
| 50000 | 0 | 36 | 33 | 0 | 174 | 168 | 0 | 127 | 106 |

Table 4.2: Reliability and Time vs NS for A-networks, $p_i = 0.4$ (Time is in seconds)

In Table 4.1 and all following tables; t_1 and t_2 correspond to the time needed to find R_K by *ALG1* and *ALG2* respectively. Also, one single column was written for the value of R_K since this value was obtained from both of

the two algorithms. This is an expected result since same simulation method was utilized in both algorithms.

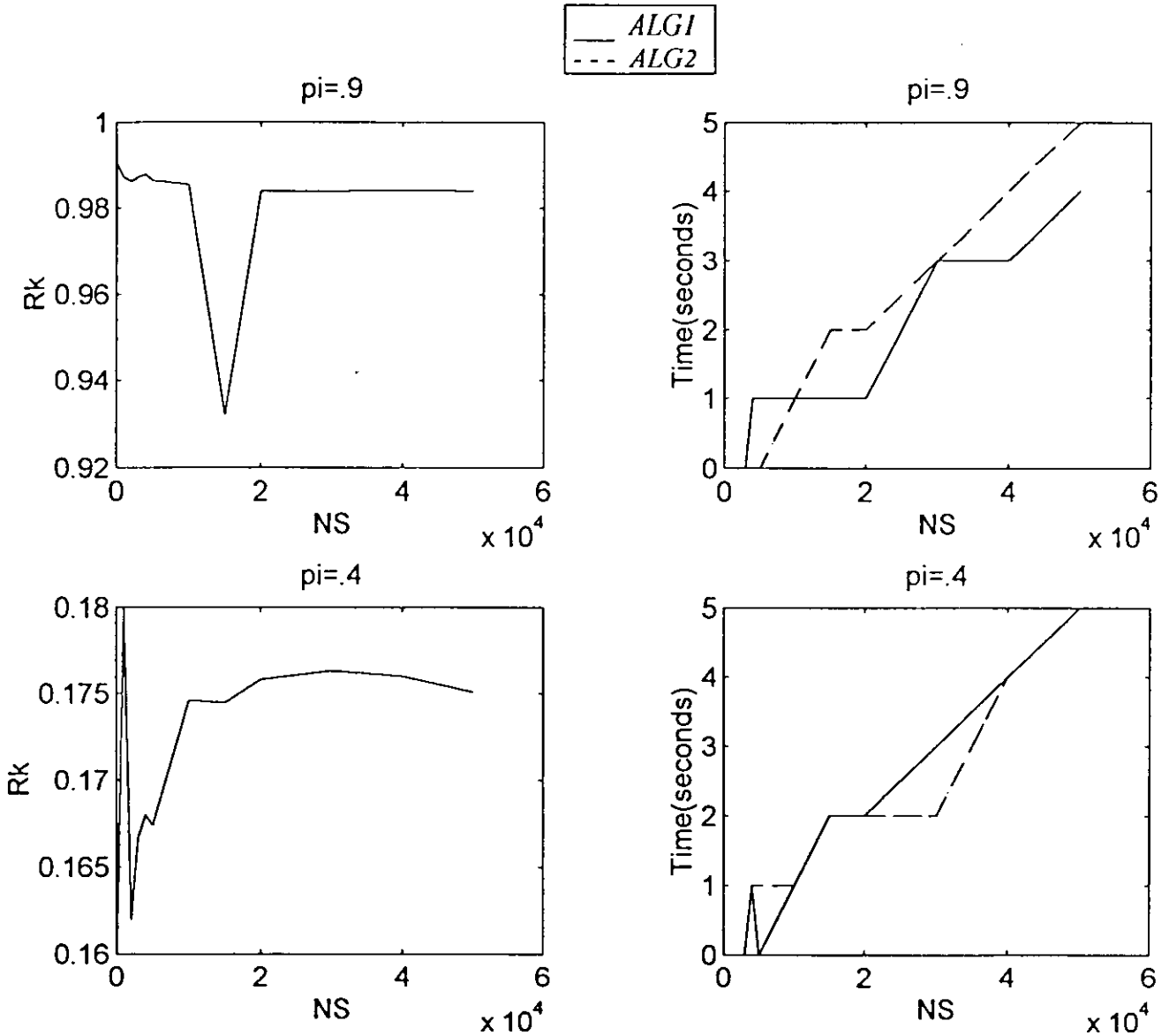


Figure 4.2: Reliability and Time vs NS for A_1 -network, for $p_i = 0.9$ and $p_i = 0.4$

Three cases of the tables above concerning small, medium and large networks are plotted in figures 4.2, 4.3 and 4.4 respectively. For the curves in Figure 4.2 and all following figures, the solid line corresponds to *ALG1* while the broken line corresponds to *ALG2*.

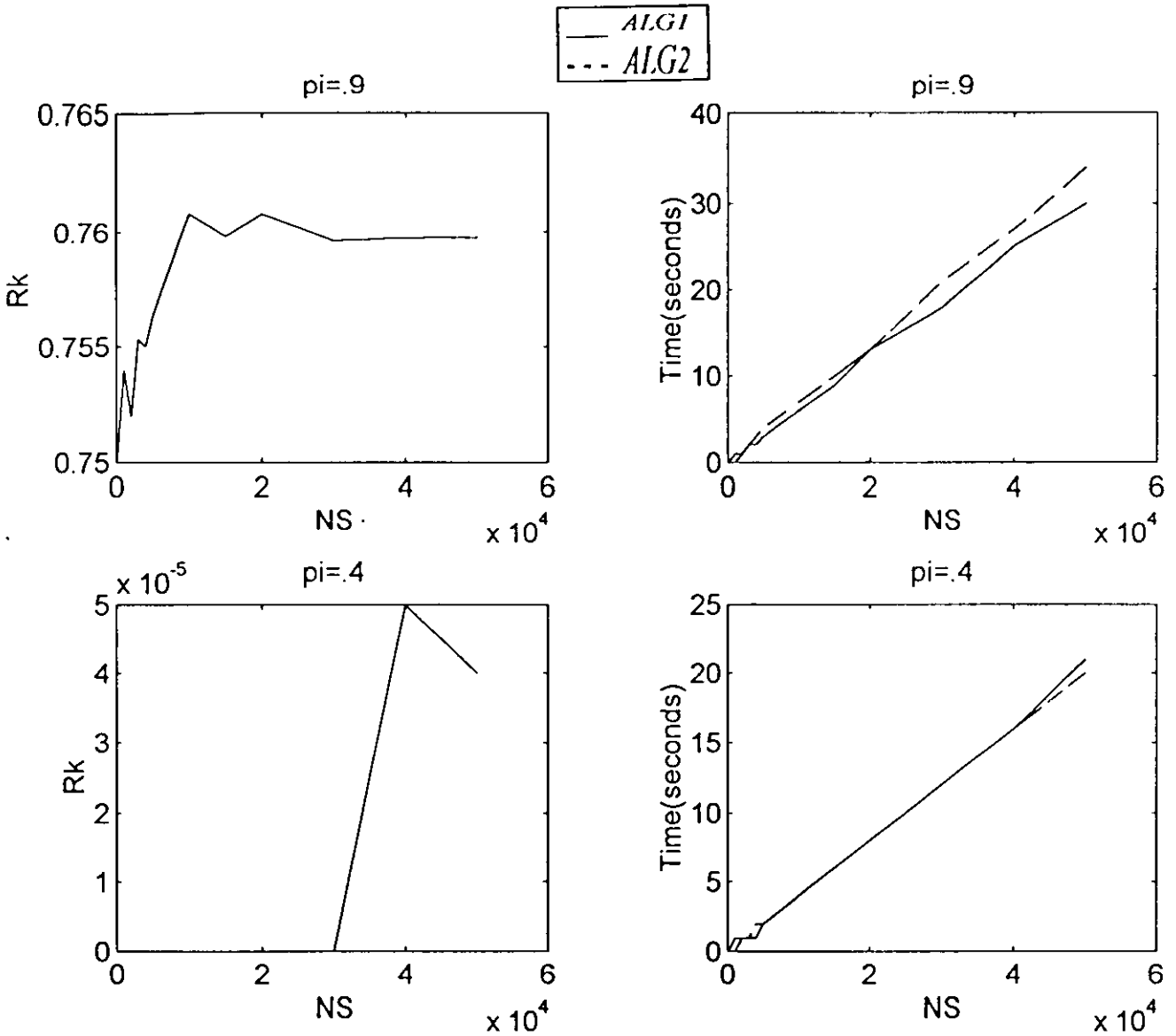


Figure 4.3: Reliability and Time vs NS for A_r -network, for $p_i = 0.9$ and $p_i = 0.4$

In all reliability curves, we can see that all of them may oscillate at first, but after $NS = 25,000$, the reliability measure seems to become constant. So a good reliability estimate may be achieved by taking only 25000 executions. This is correct for both the low- and high-edge reliability networks. Naturally, if the program continues to execute until NS approaches ∞ then the reliability estimated will be the same as the exact one.

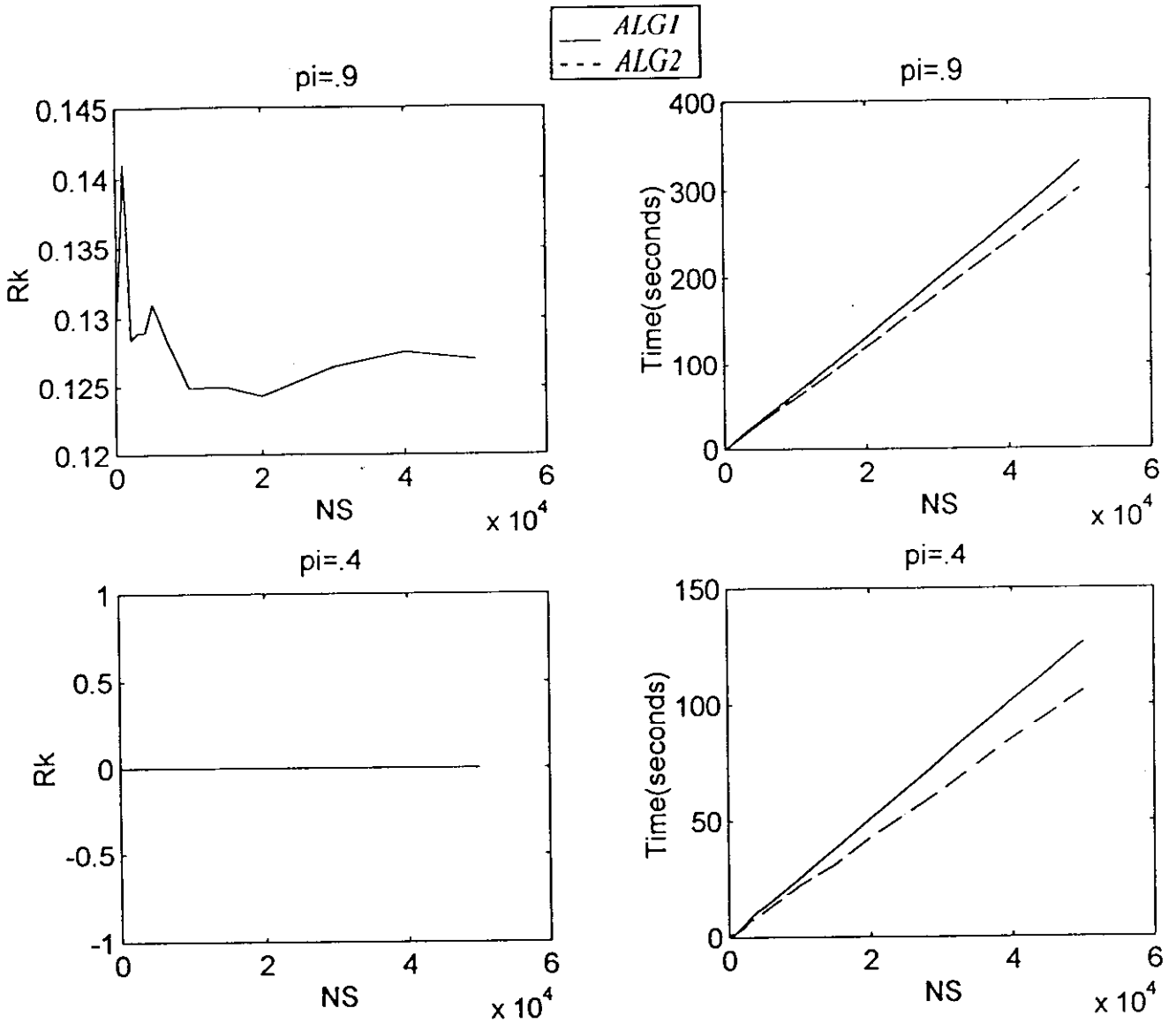


Figure 4.4: Reliability and Time vs NS for A_7 -network, for $p_i = 0.9$ and $p_i = 0.4$

Now let us look at time performance. For high edge-reliability networks, and in large networks, *ALG2* takes shorter times of execution than that of *ALG1*. However, for small and medium networks, *ALG1* shows a faster operation. Table 4.3 shows the average value of the time taken for one execution for various networks, of high edge-reliability:

| network | t1 | t2 |
|---------|-------|-------|
| A1 | 0.083 | 0.100 |
| A4 | 0.608 | 0.683 |
| A7 | 6.610 | 6.030 |

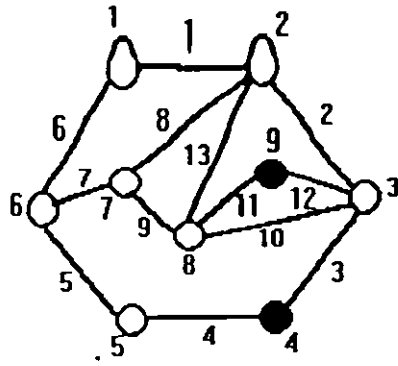
Table 4.3: Time for one execution run for A -network, $p_i = 0.9$ (Time is in milliseconds)

For low edge-reliability networks, $ALG2$ takes less time for all sizes of networks. Table 4.4 shows the average time of execution for various networks:

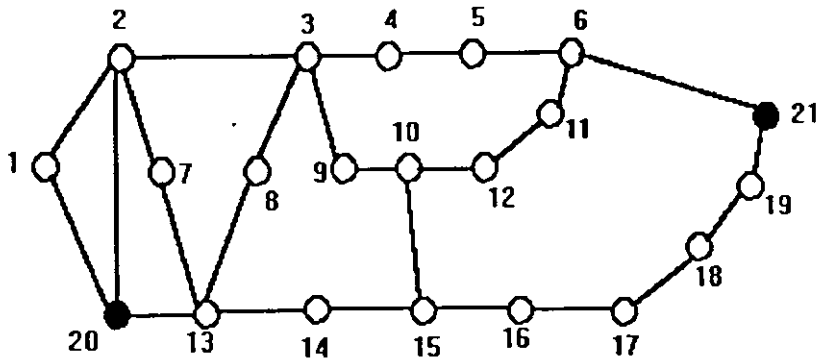
| network | t1 | t2 |
|---------|-------|-------|
| A1 | 0.100 | 0.092 |
| A4 | 0.408 | 0.400 |
| A7 | 2.540 | 2.120 |

Table 4.4: Time for one execution run for A -networks, $p_i = 0.4$ (Time is in milliseconds)

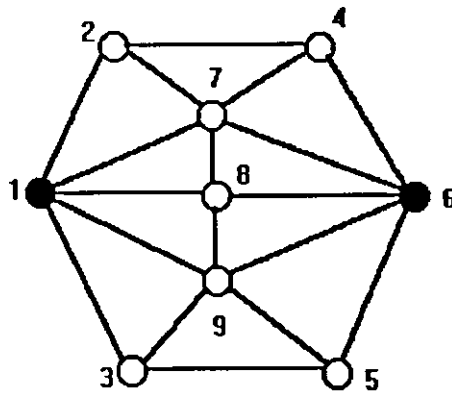
Now the B -networks are studied: B_1 , B_2 , B_3 and B_4 . These networks are found in many published works, as they represent real networks. Figure 4.5 shows these four networks. These networks are concerned with terminal-pair reliability. We examined these networks, assuming that the k -terminal set includes only two terminals as specified in the figure. Results are shown in Table 4.5. Note that we wrote the exact measures in the last row of this table, together with the time needed to calculate it as found by $PRFA$ algorithm (Page and Perry, 1988) that is discussed later in Section 4.2.



(a)

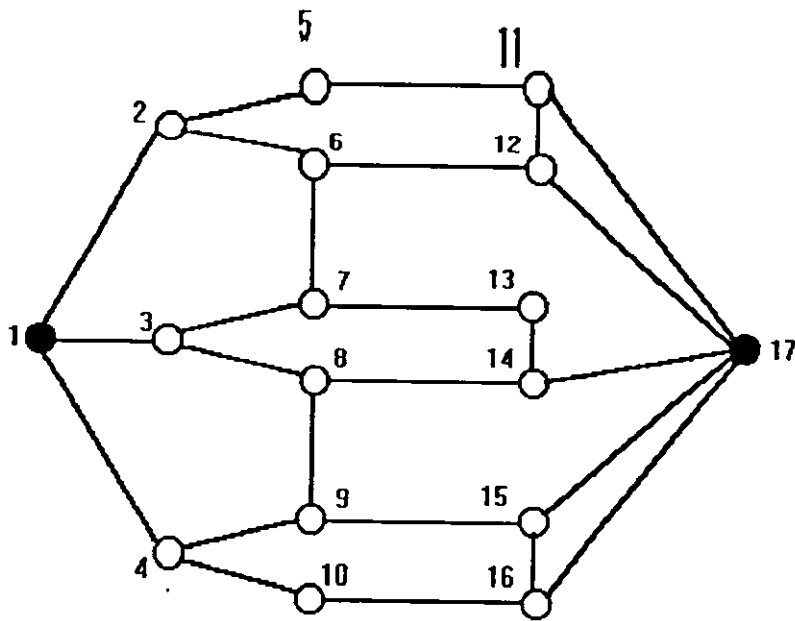


(b)



(c)

Figure 4.5: B -networks (a) $B_1 = (9,13)$ (b) $B_2 = (21,26)$ (c) $B_3 = (9,18)$ (cont'd)



(d)

Figure 4.5: (cont'd) (d) $B_4 = (17, 26)$

| NS | B1 | | | B2 | | | B3 | | | B4 | | |
|-------|----------|----|----|----------|-----|----|----------|-----|----|----------|----|----|
| | Rk | t1 | t2 | Rk | t1 | t2 | Rk | t1 | t2 | Rk | t1 | t2 |
| 100 | 0.97 | 0 | 0 | 0.95 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1000 | 0.973 | 0 | 0 | 0.916 | 1 | 1 | 1 | 0 | 0 | 0.997 | 0 | 0 |
| 2000 | 0.975 | 0 | 1 | 0.922 | 1 | 1 | 1 | 1 | 0 | 0.9975 | 1 | 1 |
| 3000 | 0.975667 | 0 | 1 | 0.917667 | 2 | 1 | 1 | 1 | 0 | 0.997333 | 1 | 1 |
| 4000 | 0.97325 | 0 | 0 | 0.9185 | 2 | 2 | 1 | 0 | 0 | 0.99775 | 1 | 1 |
| 5000 | 0.9712 | 1 | 1 | 0.9172 | 2 | 3 | 1 | 1 | 1 | 0.9978 | 1 | 1 |
| 10000 | 0.9704 | 2 | 1 | 0.9154 | 5 | 4 | 1 | 1 | 1 | 0.9977 | 4 | 4 |
| 15000 | 0.969333 | 3 | 2 | 0.911333 | 8 | 7 | 1 | 3 | 2 | 0.9978 | 5 | 5 |
| 20000 | 0.9702 | 4 | 3 | 0.91155 | 10 | 9 | 0.99995 | 3 | 3 | 0.99815 | 7 | 7 |
| 30000 | 0.968367 | 4 | 4 | 0.913233 | 15 | 13 | 0.999967 | 4 | 5 | 0.998233 | 11 | 11 |
| 40000 | 0.96845 | 6 | 5 | 0.91415 | 20 | 18 | 0.999975 | 6 | 7 | 0.9982 | 14 | 15 |
| 50000 | 0.96864 | 8 | 7 | 0.9137 | 25 | 23 | 0.99998 | 7 | 9 | 0.99816 | 17 | 17 |
| Exact | 0.969112 | 1 | | 0.912914 | 1.9 | | 0.999971 | 4.5 | | 0.998059 | 61 | |

Table 4.5: Reliability and Time vs NS for B-networks, $p_i = 0.9$ (Time is in seconds)

For networks B_1 and B_2 , the results are drawn in Figure 4.6, to show the reliability and the execution time as a function of number of samples done, NS .

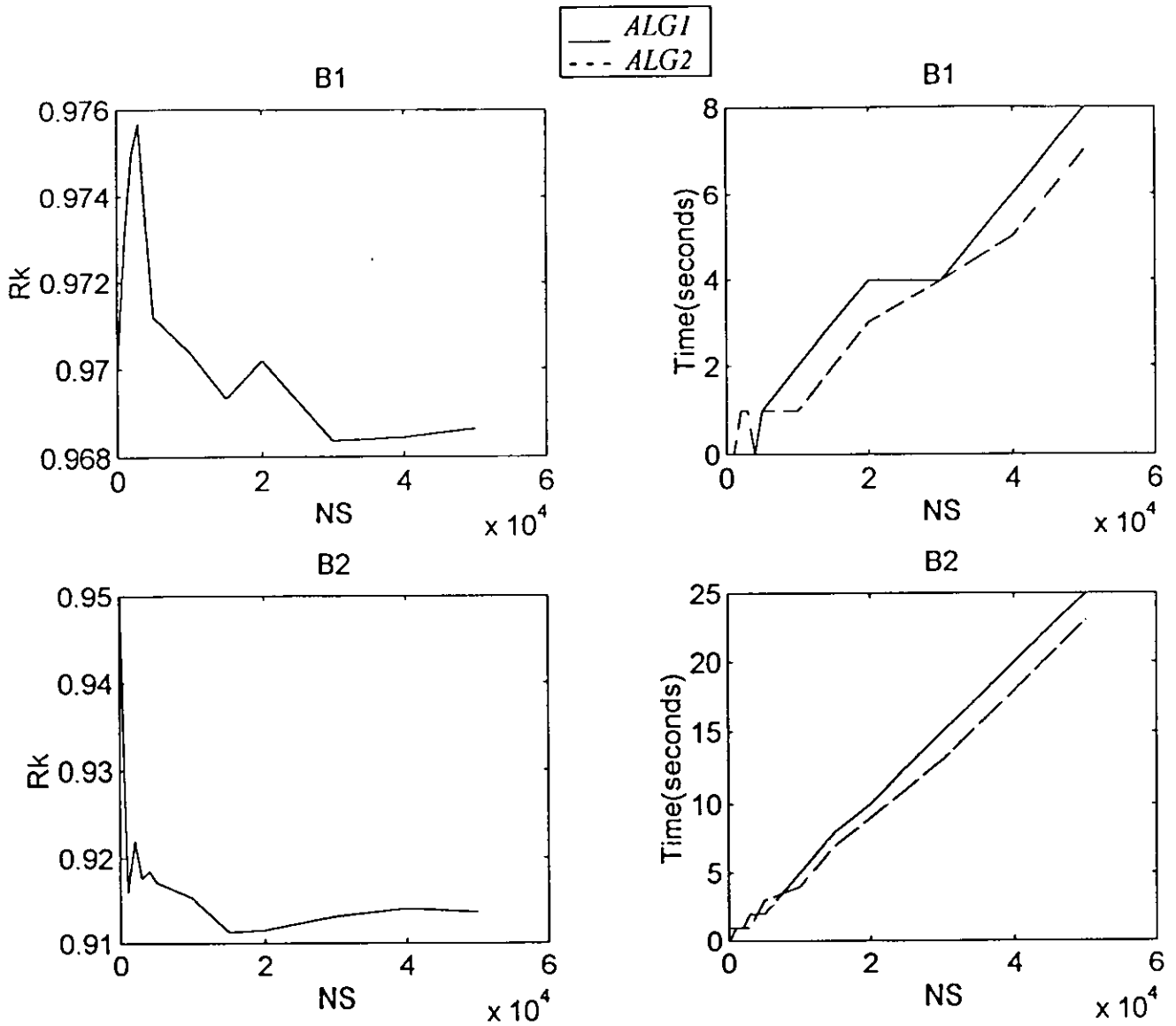


Figure 4.6: Reliability and Time vs NS for B_1 and B_2 networks.

The k -terminal reliability problem, which was stated in Chapter 1, and is redrawn here in Figure 4.7, is solved and the results are shown in Table 4.6. The corresponding curves are shown in Figure 4.8.

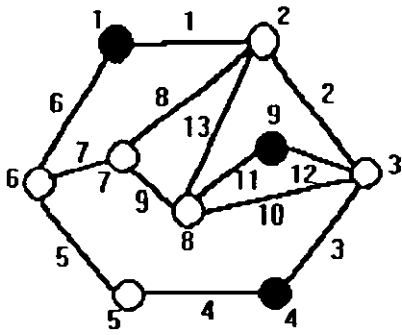


Figure 4.7: k -terminal reliability problem, $K_2 = (9,13)$, $k = 3$

| NS | $p_i = .95$ | | | $p_i = 0.1$ | | |
|-------|-------------|----|----|-------------|----|----|
| | Rk | t1 | t2 | Rk | t1 | t2 |
| 10 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2000 | 0.991 | 0 | 0 | 0.001 | 0 | 1 |
| 4000 | 0.99175 | 1 | 0 | 0.0075 | 1 | 1 |
| 6000 | 0.9915 | 1 | 1 | 0.0005 | 0 | 0 |
| 8000 | 0.991125 | 1 | 1 | 0.000375 | 1 | 1 |
| 10000 | 0.9911 | 1 | 1 | 0.0004 | 2 | 1 |
| 12000 | 0.990833 | 2 | 1 | 0.000417 | 1 | 1 |
| 14000 | 0.990714 | 2 | 2 | 0.000357 | 1 | 2 |
| 16000 | 0.990688 | 2 | 3 | 0.000313 | 2 | 1 |
| 18000 | 0.990778 | 2 | 3 | 0.000278 | 2 | 2 |
| 20000 | 0.99065 | 3 | 3 | 0.00025 | 3 | 2 |
| 22000 | 0.990318 | 4 | 3 | 0.000273 | 2 | 3 |
| 24000 | 0.989958 | 3 | 4 | 0.00025 | 3 | 3 |
| 26000 | 0.989731 | 4 | 4 | 0.000269 | 3 | 3 |
| 28000 | 0.989857 | 5 | 5 | 0.00025 | 3 | 3 |
| 30000 | 0.989633 | 5 | 5 | 0.000233 | 4 | 3 |
| 35000 | 0.989514 | 5 | 6 | 0.000229 | 4 | 4 |
| 40000 | 0.989525 | 6 | 7 | 0.000225 | 5 | 5 |
| 45000 | 0.989556 | 7 | 7 | 0.000222 | 5 | 5 |
| 50000 | 0.9897 | 8 | 8 | 0.00022 | 6 | 5 |
| 60000 | 0.98965 | 9 | 10 | 0.00025 | 7 | 6 |
| 70000 | 0.9895 | 11 | 11 | 0.000214 | 8 | 8 |

Table 4.6: Reliability and Time vs NS for K_2 -network, for both $p_i = 0.95$ and $p_i = 0.1$,

$k = 3$ (Time is in seconds)

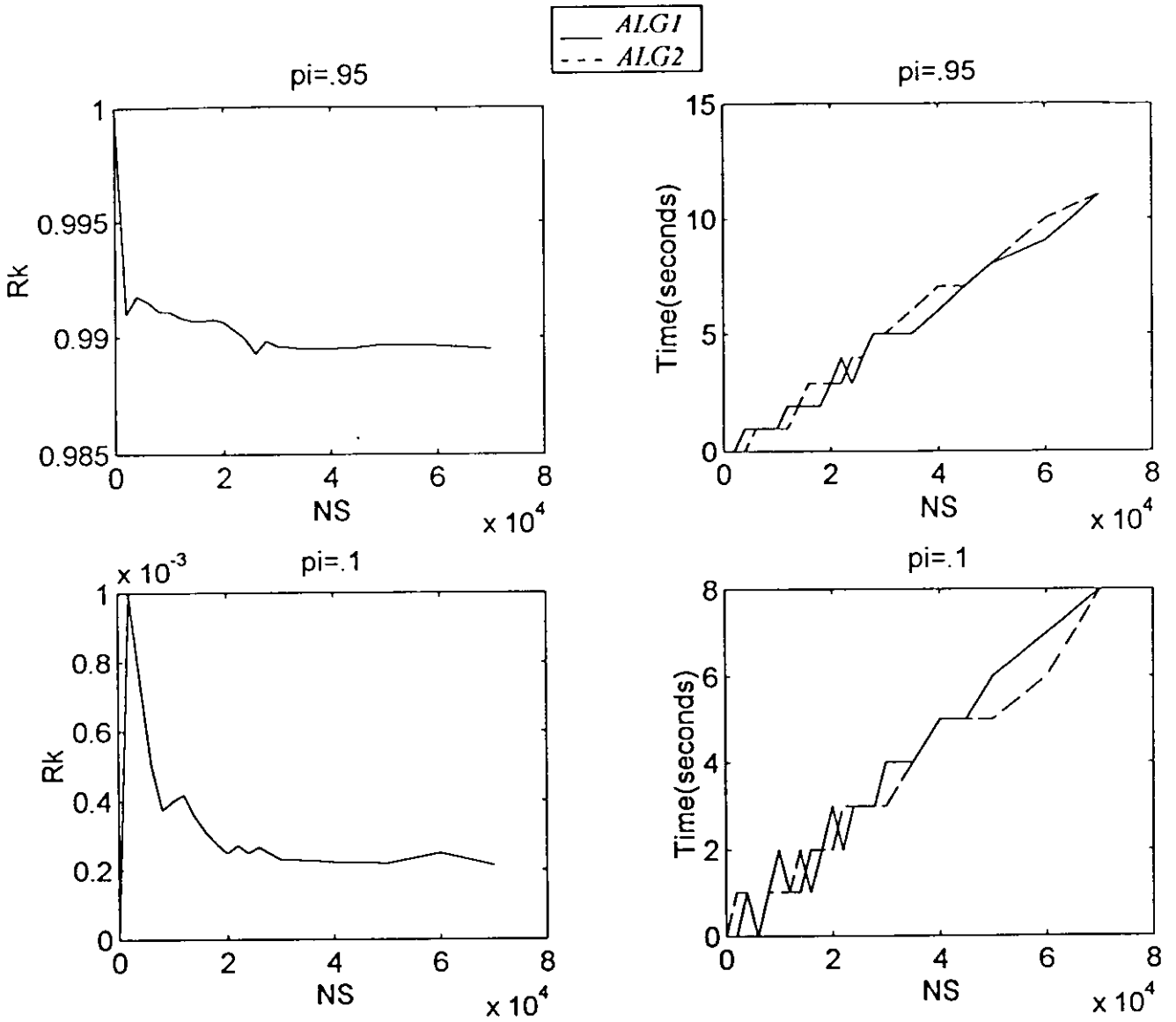


Figure 4.8: Reliability and Time vs NS for K_2 network, , $p_i = 0.95$ and , $p_i = 0.1$

The average time of execution is similar to the characteristics seen previously. It is smaller for *ALG2* compared to *ALG1* except for the case of high edge-reliability networks. Table 4.7 shows the average time of one execution run when solving for the reliability of K_2 network.

| p_i | t_1 | t_2 |
|-------|-------|-------|
| 0.95 | 0.425 | 0.450 |
| 0.1 | 0.325 | 0.300 |

Table 4.7: Time for one execution run for K_2 network (Time is in milliseconds)

Now looking at R_K , for high edge-reliability networks, the last 11 values, in Table 4.6, as shown in Table 4.8 for the extreme cases only. An increase in NS by 48000 affects R_K by only 0.083%. Therefore the reliability may be accurately estimated from the table at $NS = 22000$ without the need to go for additional executions.

| NS | R_k |
|-------|---------|
| 22000 | .990318 |
| 70000 | .989500 |

Table 4.8: Two values of NS and the corresponding R_K for K_2 network
(Time is in seconds)

Now, let us examine the results as a function of p_i . First we fix NS at 30000, then vary p_i , for networks A_4 (medium size all-terminal problem), B_2 (two-terminal problem) and K_2 (k -terminal problem). These results are shown in Table 4.9, and the corresponding curves are shown in figures 4.9, 4.10 and 4.11 respectively.

| pi | K2 | | | A4 | | | B2 | | |
|-------|----------|----|----|----------|----|----|----------|----|----|
| | Rk | t1 | t2 | Rk | t1 | t2 | Rk | t1 | t2 |
| 0.001 | 0 | 4 | 3 | 0 | 10 | 9 | 0 | 9 | 8 |
| 0.1 | 0.000233 | 4 | 3 | 0 | 10 | 9 | 0 | 9 | 8 |
| 0.2 | 0.004 | 3 | 4 | 0 | 10 | 10 | 0.000267 | 10 | 9 |
| 0.3 | 0.022867 | 4 | 3 | 0 | 11 | 11 | 0.001733 | 10 | 9 |
| 0.4 | 0.0837 | 4 | 4 | 0 | 12 | 11 | 0.010033 | 12 | 9 |
| 0.5 | 0.208767 | 4 | 5 | 0.002533 | 14 | 14 | 0.042533 | 13 | 10 |
| 0.6 | 0.403633 | 4 | 4 | 0.0257 | 16 | 16 | 0.138433 | 14 | 11 |
| 0.7 | 0.627967 | 4 | 4 | 0.12033 | 19 | 19 | 0.339833 | 15 | 12 |
| 0.8 | 0.825033 | 5 | 5 | 0.383267 | 19 | 20 | 0.644667 | 15 | 13 |
| 0.9 | 0.956 | 5 | 4 | 0.7596 | 19 | 20 | 0.913233 | 15 | 13 |
| 0.999 | 1 | 4 | 5 | 0.999967 | 15 | 17 | 1 | 17 | 14 |

Table 4.9: Reliability and Time vs p_i for K_2 , A_4 and B_4 networks
(Time is in seconds)

The average time of execution is smaller for low edge-reliability networks. This is an expected result since a 'connected' decision cannot be taken until all the k -terminal nodes are scanned. Results tabulated above also agree with relationships between the time of execution and NS for low and high edge-reliability networks, as mentioned earlier concerning both $ALG1$ and $ALG2$.

4.2 ALG1 and ALG2 Vs Previous Studies :

In this section, results for both $ALG1$ and $ALG2$ are compared to the MAP algorithm (Ayoub and Shahbaz, 1996) and the $PRFA$ algorithm (Page and Perry, 1988). The former one is used to solve the all-terminal problem in the A -networks. It finds the probability of failure $F(G)$, and consequently $R(G)$:

$$R(G) = 1 - F(G)$$

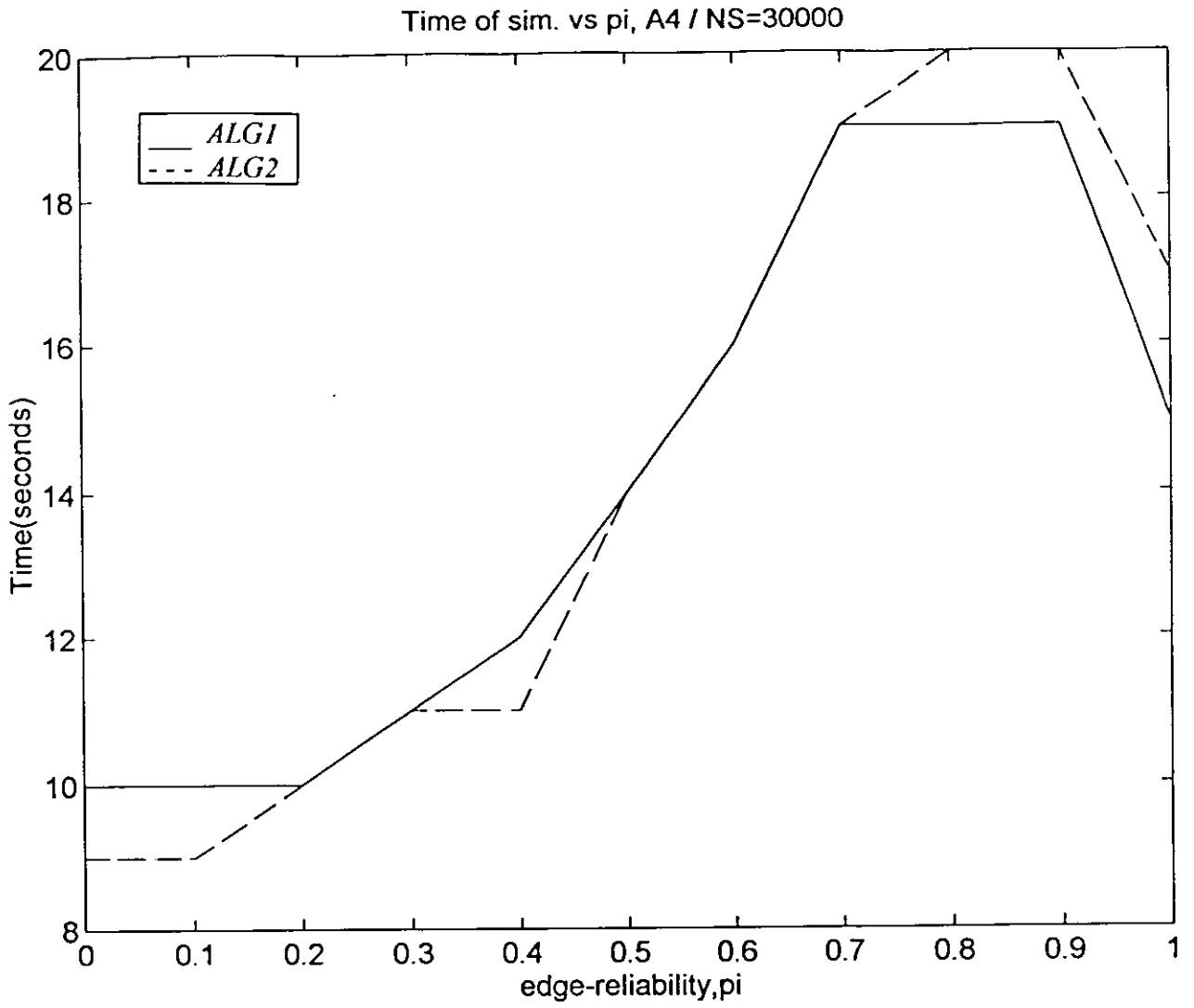


Figure 4.9 : Time vs p_i for A_4 network.

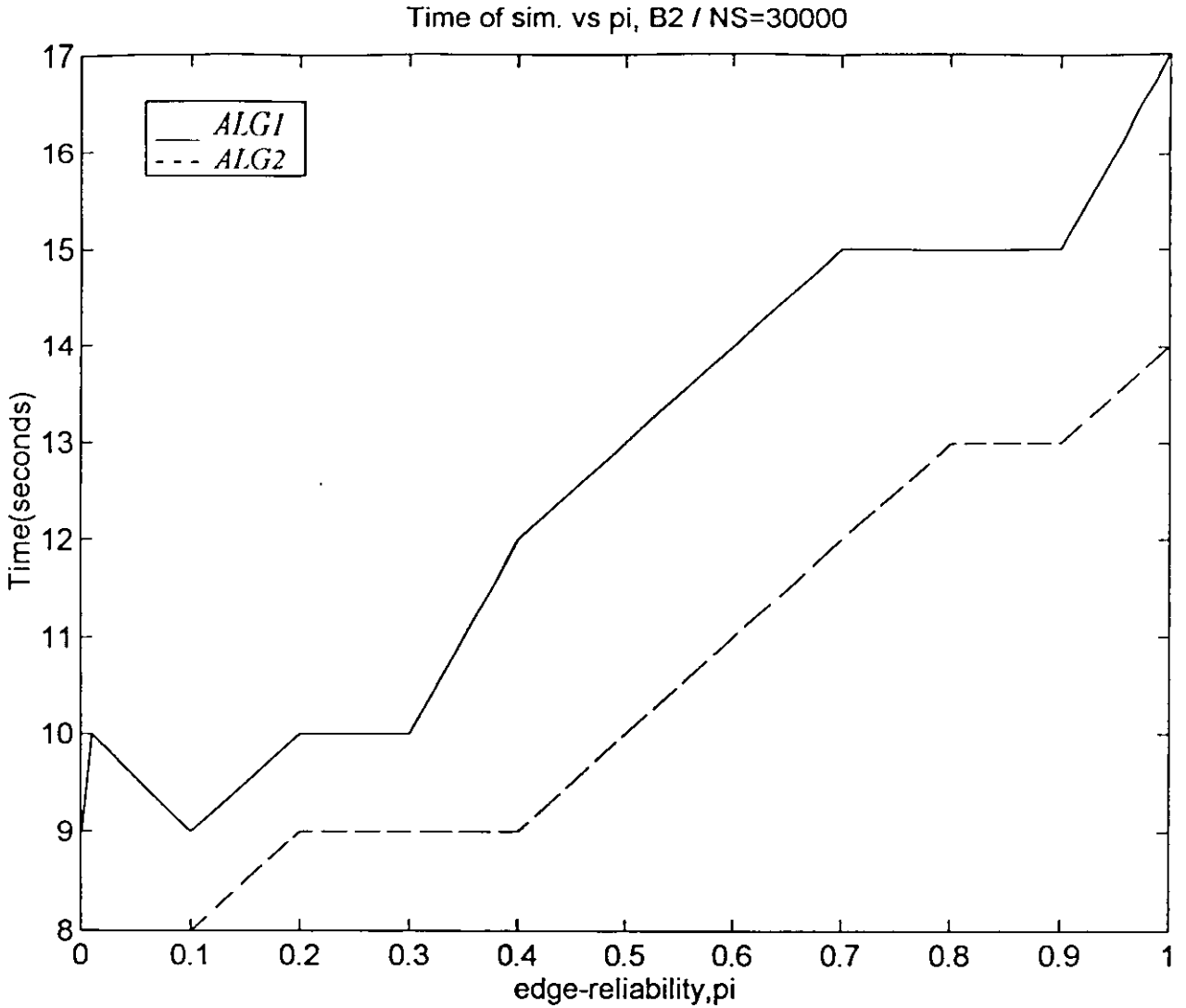


Figure 4.10: Time vs p_i for B_2 network.

Table 4.10 shows a comparison between *ALG1* and *ALG2* with *MAP* according to the value of R_K and the execution time needed. These results were taken with a fixed $NS=30,000$ for a p_i equals to 0.9. In this table; t_{map} corresponds to the time needed to find $F(G)$ by *MAP* algorithm as found by the authors of (Ayoub and Shahbaz,1996).

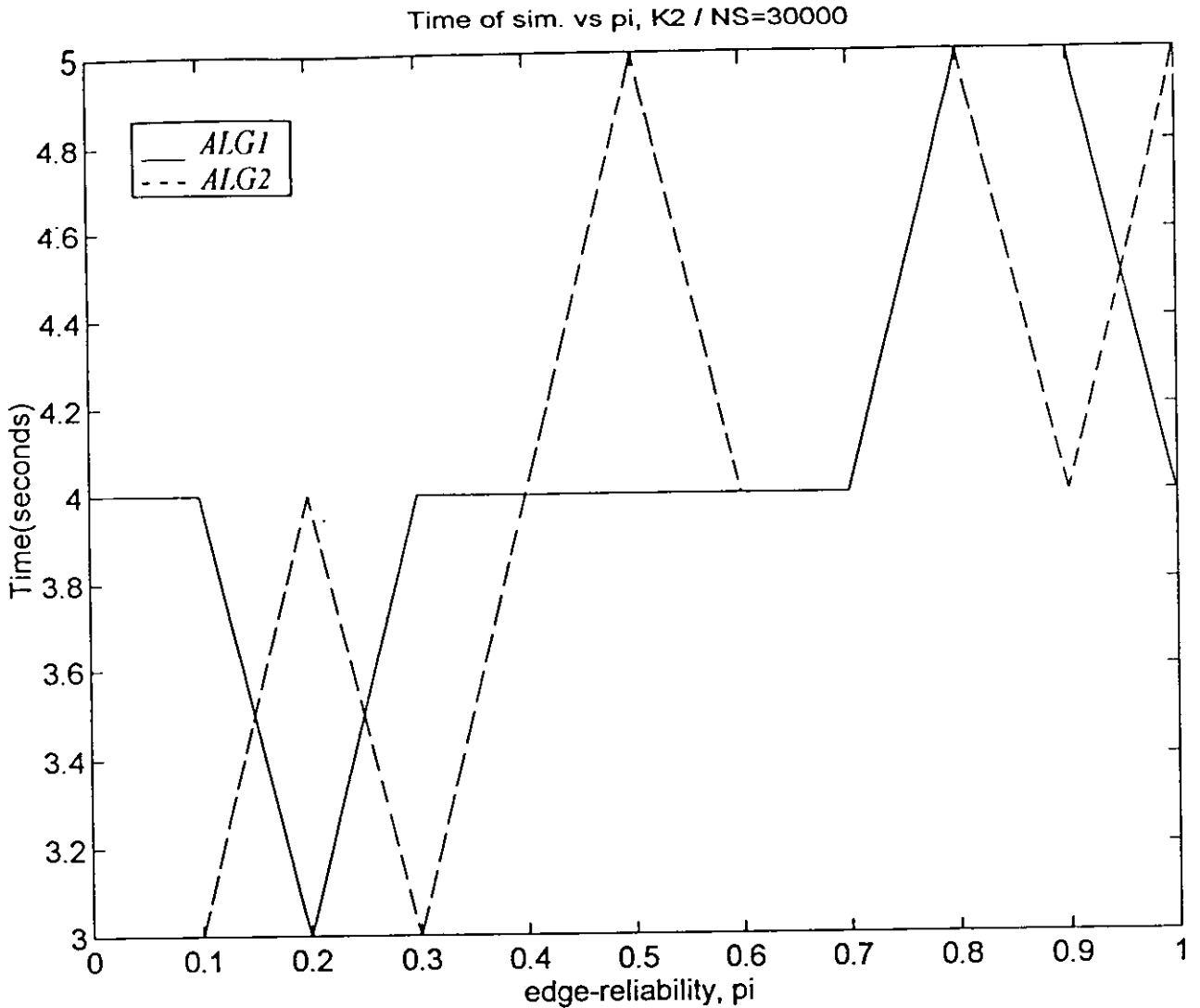


Figure 4.11: Time vs p_i for K_2 network.

While the value of the reliability approximately the same, Figure 4.12 shows the big savings in time due to the use of our algorithms compared with that of the *MAP*. This is expected since our algorithms do not search the whole network in each sample. In addition, Figure 4.13 shows the relationship between the time needed and the number of nodes in the *A*-networks for the low edge-reliability case ($p_i = 0.4$). The shown curves when interpolated may have a slope which is 50% less than that of the high edge-reliability case. This indicates the power of the two algorithms in solving

low edge-reliability networks, in contrast to all existing exact methods which are weak in dealing with this special case of networks.

| Network | Rk | R(G) map | t1 | t2 | tmap |
|---------|----------|----------|-----|-----|------|
| A1 | 0.9839 | 0.984 | 3 | 3 | 65 |
| A2 | 0.935767 | 0.984 | 5 | 6 | 122 |
| A3 | 0.839333 | 0.8399 | 11 | 13 | 196 |
| A4 | 0.7596 | 0.7585 | 18 | 21 | 290 |
| A5 | 0.5675 | 0.5695 | 39 | 41 | 667 |
| A6 | 0.2822 | 0.2806 | 104 | 101 | 737 |
| A7 | 0.126433 | 0.179 | 198 | 181 | 1035 |

Table 4.10: Reliability and Time vs N for A -networks, using $ALG1$, $ALG2$ and MAP
(Time is in seconds)

The $PRFA$ algorithm is used to solve the exact value of the k -terminal reliability problem for the B -networks, for the special case of $k=2$. Table 4.11 shows, for various values of NS , the values of R_k , while fixing p_i at 0.9, and the difference between the exact and the approximate reliability measures. the achieved accuracy can be easily seen in the reliability measure.

Although the time needed by the $PRFA$ algorithm may appear less than that of our algorithms, we should not forget that $PRFA$ has limitations on its use. One of them is that it treats networks that do not have a large number of vertices of degree greater than two (Page and Perry, 1988). In contrast, our algorithms are general for any network topology. If we compare it with the time needed to solve network B_4 , we can see that our algorithms greatly reduce the execution time. The time taken to solve B_4 was 61 seconds in $PRFA$ and the exact value was 0.998059. Our two algorithms do 30000 samples in only 11 seconds to obtain an estimate value

equals to 0.998233 with a percentage error of 0.0174 %. This is a very accurate result with a 80% saving in execution time. One can notice the jump in the execution time between B_1 and B_4 . Although B_4 is smaller it takes larger time due to the change of the topology of the network.

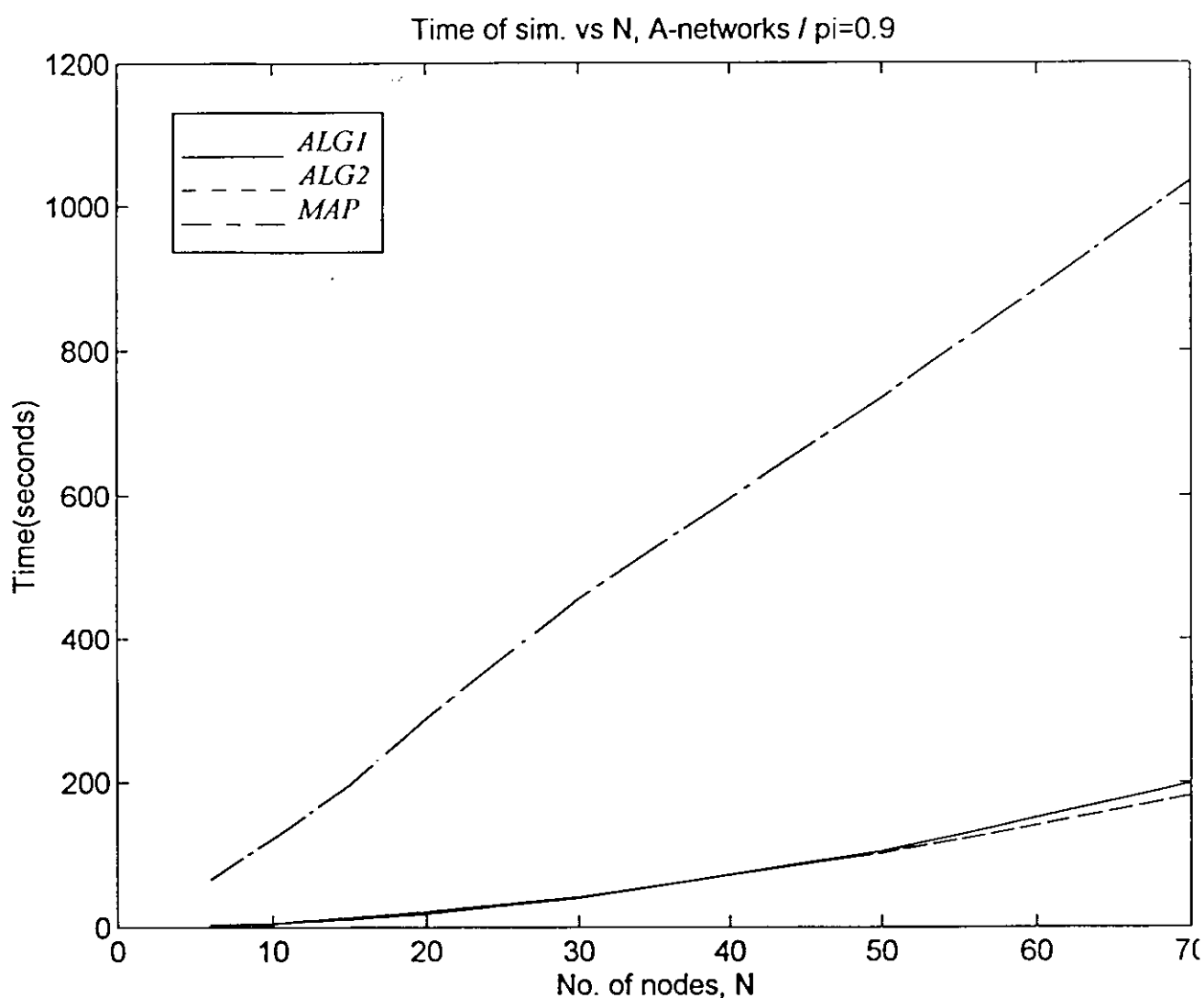


Figure 4.12: Time vs N for A -networks, $p_i = 0.9$

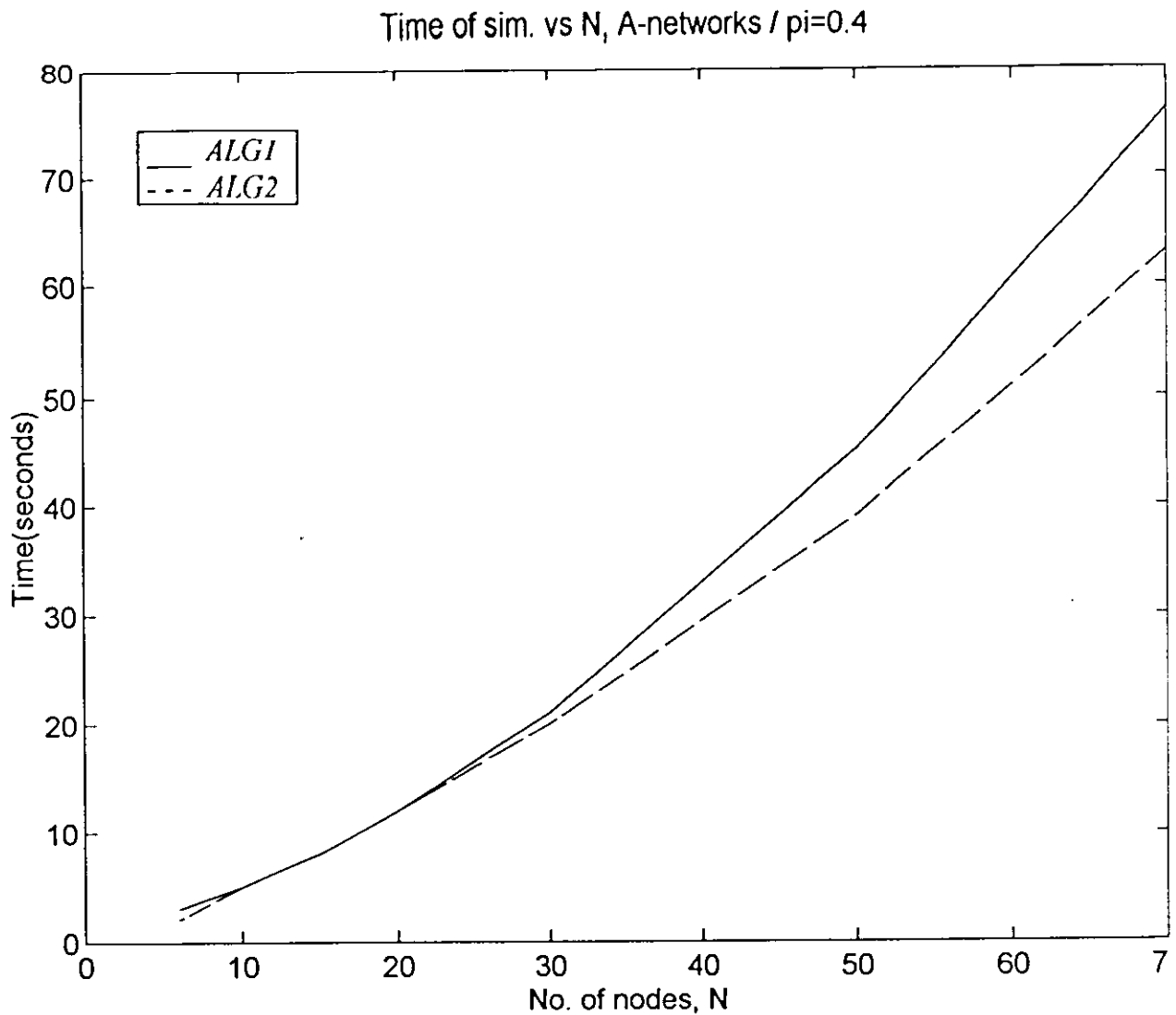


Figure 4.13: Time vs N for A -networks, $p_i = 0.4$.

| NS | B1 | | | | B2 | | | |
|-------|----------|----------|----|----|----------|----------|-----|----|
| | Rk | Rk-Rk | t1 | t2 | Rk | Rk-Rk | t1 | t2 |
| 20000 | 0.9702 | 0.00109 | 4 | 3 | 0.91155 | 0.00136 | 10 | 9 |
| 30000 | 0.968367 | 0.000745 | 4 | 4 | 0.913233 | 0.000319 | 15 | 13 |
| 40000 | 0.96845 | 0.000662 | 6 | 5 | 0.91415 | 0.00124 | 20 | 18 |
| 50000 | 0.9684 | 0.000712 | 8 | 7 | 0.9137 | 0.000786 | 25 | 23 |
| Exact | 0.969112 | | 1 | | 0.912914 | | 1.9 | |

Table 4.11: Reliability and Time vs NS for B_1 and B_2 networks, using $ALG1$, $ALG2$ and $PRFA$ (Time is in seconds)

$ALG2$ seems to be faster than $ALG1$ here. This is due to the fact that it does a single search between any two terminals following the shortest-path

between them. *ALG1* on the other hand, does not give any priority to the k -terminal nodes.

4.3 All k -Terminal Reliability:

In this section we are dealing with a network, for which we want to find the network reliabilities that result from all possible k sets. The network we deal with here is the network we mentioned in Chapter 1, redrawn here in Figure 4.14. We assumed that $p_i = 0.9$. Results are tabulated in Table 4.12.

For example, the first row in Table 4.12 assumes that the k -terminal nodes are nodes one and two. There are two values of NS : 20000 and 30000. For the case of $NS=20000$, the value of R_K was found to be 0.9972, and the time needed by *ALG1* was 2 seconds, while one second was needed by *ALG2*, and so on. It is obvious that the number of k -terminal nodes may be 2, 3, 4, 5, 6 and 7. In this manner, the whole table was produced.

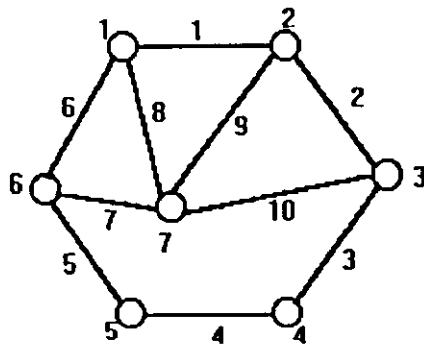


Figure 4.14: $K_t = (7,10)$

| k-set | 20000 | | | 30000 | | |
|-------|---------|----|----|----------|----|----|
| | Rk | t1 | t2 | Rk | t1 | t2 |
| 1,2 | 0.9972 | 2 | 1 | 0.997233 | 2 | 2 |
| 1,3 | 0.99535 | 1 | 2 | 0.995267 | 3 | 3 |
| 1,4 | 0.97715 | 2 | 2 | 0.976933 | 3 | 3 |
| 1,5 | 0.97585 | 2 | 1 | 0.975067 | 3 | 3 |
| 1,6 | 0.99585 | 2 | 2 | 0.9958 | 2 | 3 |
| 1,7 | 0.9983 | 2 | 2 | 0.998233 | 3 | 2 |
| 2,3 | 0.9959 | 2 | 1 | 0.995967 | 3 | 3 |
| 2,4 | 0.97745 | 2 | 2 | 0.977433 | 3 | 3 |
| 2,5 | 0.97555 | 2 | 2 | 0.975167 | 3 | 3 |
| 2,6 | 0.99545 | 2 | 1 | 0.9958 | 3 | 3 |
| 2,7 | 0.9985 | 2 | 2 | 0.998533 | 3 | 2 |
| 3,4 | 0.97945 | 2 | 1 | 0.9794 | 3 | 3 |
| 3,5 | 0.9752 | 2 | 2 | 0.974967 | 3 | 3 |
| 3,6 | 0.9938 | 2 | 1 | 0.9942 | 2 | 3 |
| 3,7 | 0.99695 | 2 | 2 | 0.9969 | 3 | 2 |
| 4,5 | 0.9779 | 2 | 2 | 0.9771 | 3 | 3 |
| 4,6 | 0.97785 | 2 | 2 | 0.9778 | 3 | 3 |
| 4,7 | 0.97865 | 2 | 2 | 0.9785 | 3 | 2 |
| 5,6 | 0.97795 | 2 | 2 | 0.977467 | 3 | 3 |
| 5,7 | 0.97695 | 2 | 2 | 0.976367 | 3 | 3 |
| 6,7 | 0.99685 | 2 | 2 | 0.997033 | 4 | 2 |
| 1,2,3 | 0.99425 | 2 | 2 | 0.994267 | 3 | 2 |
| 1,2,4 | 0.97595 | 1 | 2 | 0.975867 | 3 | 3 |
| 1,2,5 | 0.9744 | 2 | 3 | 0.973833 | 3 | 3 |
| 1,2,6 | 0.9943 | 2 | 2 | 0.994467 | 3 | 3 |
| 1,2,7 | 0.99705 | 2 | 2 | 0.997033 | 3 | 3 |
| 1,3,4 | 0.97605 | 2 | 2 | 0.975867 | 3 | 3 |
| 1,3,5 | 0.97335 | 2 | 2 | 0.972767 | 2 | 4 |
| 1,3,6 | 0.99255 | 2 | 2 | 0.992667 | 3 | 4 |
| 1,3,7 | 0.99535 | 3 | 2 | 0.995233 | 3 | 3 |
| 1,4,5 | 0.966 | 2 | 2 | 0.965067 | 3 | 3 |
| 1,4,6 | 0.9755 | 2 | 2 | 0.975367 | 4 | 3 |
| 1,4,7 | 0.9771 | 2 | 2 | 0.976867 | 4 | 4 |
| 1,5,6 | 0.97495 | 2 | 2 | 0.974333 | 3 | 3 |
| 1,5,7 | 0.9756 | 2 | 2 | 0.974867 | 3 | 3 |
| 1,6,7 | 0.9955 | 3 | 2 | 0.995533 | 3 | 3 |

| | | | | | | |
|---------|---------|---|---|----------|---|---|
| 2,3,4 | 0.97645 | 2 | 2 | 0.976467 | 3 | 3 |
| 2,3,5 | 0.9734 | 2 | 2 | 0.973133 | 3 | 3 |
| 2,3,6 | 0.9926 | 2 | 2 | 0.993 | 3 | 4 |
| 2,3,7 | 0.9957 | 3 | 2 | 0.995733 | 3 | 2 |
| 2,4,5 | 0.96595 | 2 | 2 | 0.965333 | 3 | 3 |
| 2,4,6 | 0.97545 | 2 | 2 | 0.975633 | 3 | 3 |
| 2,4,7 | 0.97735 | 2 | 2 | 0.9773 | 3 | 3 |
| 2,5,6 | 0.97455 | 2 | 2 | 0.974367 | 2 | 4 |
| 2,5,7 | 0.9755 | 2 | 2 | 0.975067 | 3 | 3 |
| 2,6,7 | 0.9954 | 2 | 2 | 0.9957 | 3 | 3 |
| 3,4,5 | 0.96675 | 2 | 1 | 0.9662 | 4 | 3 |
| 3,4,6 | 0.97565 | 2 | 2 | 0.975833 | 3 | 3 |
| 3,4,7 | 0.9776 | 2 | 2 | 0.977467 | 3 | 3 |
| 3,5,6 | 0.9736 | 2 | 3 | 0.9735 | 3 | 3 |
| 3,5,7 | 0.9746 | 2 | 2 | 0.974167 | 4 | 3 |
| 3,6,7 | 0.9938 | 2 | 2 | 0.994067 | 3 | 3 |
| 4,5,6 | 0.96735 | 2 | 2 | 0.9667 | 3 | 3 |
| 4,5,7 | 0.9672 | 2 | 2 | 0.966433 | 3 | 3 |
| 4,6,7 | 0.9767 | 2 | 2 | 0.976733 | 4 | 3 |
| 5,6,7 | 0.97595 | 2 | 2 | 0.975567 | 3 | 2 |
| 1,2,3,4 | 0.97495 | 2 | 2 | 0.9749 | 3 | 3 |
| 1,2,3,5 | 0.97225 | 2 | 2 | 0.9718 | 3 | 3 |
| 1,2,3,6 | 0.99145 | 2 | 2 | 0.991667 | 3 | 4 |
| 1,2,3,7 | 0.99425 | 2 | 2 | 0.994233 | 3 | 3 |
| 1,2,4,5 | 0.9648 | 2 | 2 | 0.964 | 3 | 4 |
| 1,2,4,6 | 0.9743 | 2 | 2 | 0.9743 | 3 | 4 |
| 1,2,4,7 | 0.97595 | 2 | 3 | 0.975833 | 3 | 4 |
| 1,2,5,6 | 0.9735 | 2 | 3 | 0.9731 | 3 | 4 |
| 1,2,5,7 | 0.97435 | 2 | 2 | 0.973767 | 3 | 3 |
| 1,2,6,7 | 0.99425 | 2 | 2 | 0.9944 | 4 | 3 |
| 1,3,4,5 | 0.9649 | 2 | 2 | 0.964 | 3 | 3 |
| 1,3,4,6 | 0.9744 | 2 | 3 | 0.9743 | 3 | 3 |
| 1,3,4,7 | 0.97605 | 2 | 2 | 0.975833 | 3 | 4 |
| 1,3,5,6 | 0.97245 | 2 | 2 | 0.972033 | 4 | 3 |
| 1,3,5,7 | 0.97335 | 2 | 3 | 0.972733 | 3 | 4 |
| 1,3,6,7 | 0.99255 | 3 | 3 | 0.992633 | 3 | 3 |
| 1,4,5,6 | 0.9651 | 2 | 2 | 0.964333 | 3 | 4 |
| 1,4,5,7 | 0.96595 | 2 | 3 | 0.965 | 3 | 4 |
| 1,4,6,7 | 0.97545 | 2 | 3 | 0.9753 | 4 | 4 |
| 1,5,6,7 | 0.9747 | 2 | 3 | 0.974133 | 3 | 4 |

| | | | | | | |
|-------------|---------|---|---|----------|---|---|
| 2,3,4,5 | 0.96495 | 2 | 2 | 0.964367 | 3 | 3 |
| 2,3,4,6 | 0.97445 | 2 | 2 | 0.974667 | 3 | 4 |
| 2,3,4,7 | 0.97635 | 2 | 2 | 0.976333 | 3 | 3 |
| 2,3,5,6 | 0.9724 | 2 | 2 | 0.972333 | 3 | 3 |
| 2,3,5,7 | 0.9734 | 2 | 2 | 0.973067 | 3 | 3 |
| 2,3,6,7 | 0.9926 | 2 | 2 | 0.992933 | 3 | 4 |
| 2,4,5,6 | 0.96495 | 2 | 2 | 0.964533 | 3 | 4 |
| 2,4,5,7 | 0.96595 | 2 | 3 | 0.965267 | 4 | 3 |
| 2,4,6,7 | 0.97545 | 2 | 2 | 0.975567 | 3 | 3 |
| 2,5,6,7 | 0.9745 | 2 | 3 | 0.974267 | 3 | 4 |
| 3,4,5,6 | 0.96515 | 2 | 2 | 0.964733 | 3 | 3 |
| 3,4,5,7 | 0.96615 | 2 | 3 | 0.9654 | 4 | 3 |
| 3,4,6,7 | 0.97565 | 3 | 2 | 0.9757 | 3 | 3 |
| 3,5,6,7 | 0.9736 | 2 | 2 | 0.973367 | 4 | 4 |
| 4,5,6,7 | 0.9662 | 3 | 2 | 0.965633 | 3 | 3 |
| 1,2,3,4,5 | 0.9638 | 2 | 3 | 0.963033 | 3 | 3 |
| 1,2,3,4,6 | 0.9733 | 2 | 2 | 0.973333 | 3 | 4 |
| 1,2,3,4,7 | 0.97495 | 2 | 2 | 0.974867 | 3 | 3 |
| 1,2,3,5,6 | 0.97135 | 2 | 2 | 0.971067 | 3 | 3 |
| 1,2,3,5,7 | 0.97225 | 2 | 3 | 0.971767 | 3 | 3 |
| 1,2,3,6,7 | 0.99145 | 2 | 3 | 0.991633 | 3 | 4 |
| 1,2,4,5,6 | 0.9639 | 2 | 3 | 0.963267 | 3 | 4 |
| 1,2,4,5,7 | 0.9648 | 2 | 3 | 0.963967 | 3 | 3 |
| 1,2,4,6,7 | 0.9743 | 2 | 3 | 0.974267 | 3 | 3 |
| 1,2,5,6,7 | 0.97345 | 2 | 3 | 0.973033 | 3 | 4 |
| 1,3,4,5,6 | 0.964 | 2 | 2 | 0.963267 | 3 | 3 |
| 1,3,4,5,7 | 0.9649 | 2 | 2 | 0.963967 | 3 | 4 |
| 1,3,4,6,7 | 0.9744 | 2 | 3 | 0.974267 | 3 | 4 |
| 1,3,5,6,7 | 0.97245 | 3 | 2 | 0.972 | 4 | 4 |
| 1,4,5,6,7 | 0.96505 | 2 | 2 | 0.964267 | 3 | 4 |
| 2,3,4,5,6 | 0.96395 | 2 | 2 | 0.963567 | 3 | 3 |
| 2,3,4,5,7 | 0.96495 | 2 | 2 | 0.9643 | 3 | 3 |
| 2,3,4,6,7 | 0.97445 | 2 | 2 | 0.9746 | 3 | 3 |
| 2,3,5,6,7 | 0.9724 | 2 | 2 | 0.972267 | 3 | 3 |
| 2,4,5,6,7 | 0.96495 | 3 | 2 | 0.964467 | 3 | 4 |
| 3,4,5,6,7 | 0.96515 | 2 | 2 | 0.9646 | 4 | 3 |
| 1,2,3,4,5,6 | 0.9629 | 2 | 3 | 0.9623 | 4 | 4 |
| 1,2,3,4,5,7 | 0.9638 | 2 | 2 | 0.963 | 3 | 3 |
| 1,2,3,4,6,7 | 0.9733 | 2 | 2 | 0.9733 | 3 | 4 |
| 1,2,3,5,6,7 | 0.97135 | 2 | 2 | 0.971033 | 4 | 3 |

| | | | | | | |
|---------------|---------|---|---|----------|---|---|
| 1,2,4,5,6,7 | 0.9639 | 2 | 2 | 0.963233 | 3 | 3 |
| 1,3,4,5,6,7 | 0.964 | 2 | 2 | 0.963233 | 4 | 3 |
| 2,3,4,5,6,7 | 0.96395 | 3 | 2 | 0.9635 | 3 | 4 |
| 1,2,3,4,5,6,7 | 0.9629 | 3 | 3 | 0.962267 | 3 | 3 |

Table 4.12: All k -terminal set for K_7 network (Time is in seconds)

The above table shows the extreme values of the k -terminal reliability of K_7 network. For example, for the case of $k=2$, the maximum reliability occurs when considering nodes 2 and 7 as the k -terminal nodes, while minimum reliability occurs when considering nodes 3 and 5 as the k -terminal nodes. This may appear to be different from our previous discussions, since in our solution we were given the k -terminal nodes of special interest. However, let us look at the problem from another point of view; where we have an already built network and want to determine the optimum distribution of some nodes that are of special interest. In such a case, the table shown above is of special importance; it directs you to the suitable locations for the k -terminal nodes so as to maximize the reliability.

If we want to add a new terminal to the k -terminal nodes, what are the expected extreme reliabilities. From the above table, it is obvious that by increasing the number of terminals in the k -set then the reliability will become smaller, but how much will it be lowered? This table answers such a question. Hence it would provide a helpful tool whenever a designer wants to design or do variations on existing network.

Chapter 5

CONCLUSIONS

In this thesis, two algorithms *ALG1* and *ALG2* were stated to solve the k -terminal reliability problem using Monte Carlo simulation, based on two of the main graph search methods. No published papers have used these methods in this context. *ALG1* utilizes the depth-first search, while *ALG2* utilizes the breadth-first search techniques.

After doing sufficient number of simulations, an accurate estimate of the reliability measure is found. At most cases, the sufficient number NS is around 25,000, this is enough to give an estimate that is within a maximum of $\pm 5\%$ of the exact measure. This accuracy may be increased further simply by increasing NS .

We suggest a general solution to the k -terminal reliability problem without imposing any restrictions. The assumptions, stated in Section 3.1, are used by most of the reliability evaluation researchers. Our algorithms solve small, medium and large networks, and obtain the reliability for cases with unequal edge-reliability networks, and do not depend on network topology. Hence a general solution is obtained for generic networks.

This thesis treats the k -terminal reliability problem, whose k differs from the two- and the all-terminals problems and solves them. Previously published literature is very scarce when addressing the general k -terminal problem.

ALG1 is suggested to be used in small high edge-reliability networks, while *ALG2* shows a faster operation for other types of networks. For the case of terminal-pair problem, *ALG2* is the best choice due to faster operation irrespective of the size or edge-reliability of the network.

References :

Aho, Hopcraft and Ulman, 1976, " *The Design and Analysis of computer Algorithms* ", 3rd edition, Addison-Wesley Publishing Company, USA.

Al-Raei, 1990, " *Terminal-Pair Reliability of Communication Networks* ", M. Sc. Thesis, University of Jordan, Amman, Jordan.

Ayoub and Shahbaz, 1996, " *An Efficient Algorithm for Simulation Analysis of Network Reliability* ", International Conference on Telecommunications, ICT'96, Istanbul - Turkey, 1-17, (822-825).

Ayoub and Sutri, 1997, " *Network Reliability Based on Lost Call Traffic* ", Fourth Communication Networks Symposium, Manchester, England 7-8.

Baily and Kulkarni, 1986, " *A Recursive Algorithm for computing exact reliability measures* ", IEEE Trans. on Reliability, Vol. R-35, No.1, USA,(36-40).

Barton, 1970, " *A Primer on Simulation and Gaming* ", 1st edition, Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA.

- Belovich, 1995, " *A Design Technique for Reliable Networks Under a Non-Uniform Traffic Distribution* ", IEEE Trans. on Reliability, Vol. 44, No.3, USA,(377-387).
- Brassard and Bartley, 1988, " *Algorithmics Theory and Practice* ", 1st edition, Prentice-Hall International, USA.
- Bulka and Dugan, 1994, " *Network s-t Reliability Bounds Using a 2-Dimensional Reliability Polynomial* ", IEEE Trans. on Reliability, Vol. 3, No. 1, USA,(39-45).
- Clark and Holton, 1991, " *A First Look at Graph Theory* ", World Scientific.
- Cravis, 1981, " *Communications Network Analysis* ", 1st edition, Arthur D. Little, USA.
- Dengiz, Altiparmak and Smith, 1997, " *Efficient Optimization of All-Terminal Reliable Networks Using an Evolutionary Approach* ", IEEE Trans. on Reliability, Vol. 46, No. 1, USA, (18-26).
- Deo, 1990, " *Graph Theory with Applications to Engineering and Computer Science* ", 6th edition, Prentice-Hall of India, Newdelhi,.
- Elperin, Gertsbakh and Lomonosov, 1991, " *Estimation of Network Reliability Using Graph Evolution Models* ", IEEE Trans. on Reliability, Vol. 40, No. 5, USA, (572-581).

Even, 1979, "*Graph Algorithms* ", 1st edition, Computer Science, USA.

Golumbic, 1980 , "*Algorithmic Graph Theory and Perfect Graphs* ", Academic Press.

Harary, 1972, "*Graph Theory* ", 3rd edition, Addison-Wesley Publishing Company, USA.

Hartless and Leemis, 1996, "*Computational Algebra Applications in Reliability Theory* ", IEEE Trans. on Reliability, Vol. 45, No. 3, USA, (393-399).

Page and Perry, 1988, "*A practical Implementation of the Factoring Theorem of Network Reliability* ", IEEE Trans. on Reliability, Vol. 37, No. 3, USA, (259-267).

Page and Perry, 1994, "*Reliability Polynomials and Link Importance in Networks* ", IEEE Trans. on Reliability, Vol. 43, No. 1, USA, (51-58).

Resende, 1986, "*A Program for Reliability Evaluation of Unidirectional Networks Via Polygon-to-Chain Reductions*", IEEE Trans. on Reliability, Vol. R-35, No. 1, USA, (24-29).

Tarjan, 1972, "*Depth-First Search and Linear Graph Algorithms*" , Siam Computers, Vol. 1, No. 2, , (146-160).

Wood, 1986, "*Factoring Algorithms for Computing k-Terminal Network Reliability*", IEEE Trans. on Reliability, Vol. R-35, No. 3, USA, (269-278).

Yin and Silio, 1994, "*k-Terminal Reliability in Ring Networks*", IEEE Trans. on Reliability, Vol. 43, No. 3, USA, (389-401).

Yoo and Deo, 1988, "*A Comparison of Algorithms for Terminal-Pair Reliability*", IEEE Trans. on Reliability, Vol. 37, No. 2, USA, (210-215).

Appendix 1:

f-t To Connection-matrix Transformer

```

//      M= # of edges
//      N= # of nodes
//      f:1*M ,t:1*M
//      g:N*N
// prepared by: Wael H. Saafin

#include<stdio.h>
void main(void)
{
int i,j,M,N,f[100],t[100],g[100][100];
M=6;
N=4;
printf("f[i] t[i]");
for (i=0;i<M;i++)
scanf("%d %d",&f[i],&t[i]);
for (i=0;i<M;i++)
{g[i][i]=0;
g[f[i]-1][t[i]-1]=1;
g[t[i]-1][f[i]-1]=1;
}

for (i=0;i<N;i++)
{printf("\n");
for (j=0;j<N;j++)

```

Appendix 2

Incidence- To Connection-Matrix Transformer

```

// prepared by: Wael H. Saafin
#include<stdio.h>
void main(void)
{
int i,j,k,N,M;
int A[7][10],G[7][7],g[2];
N=7; // number of nodes
M=10; // number of edges
for(i=0;i<N;i++)
for(j=0;j<N;j++)G[i][j]=0;
for(i=0;i<N;i++)
{for(j=0;j<M;j++) A[i][j]=0;}

A[0][0]=1;
A[0][5]=1;
A[0][7]=1;
A[1][0]=1;
A[1][1]=1;
A[1][8]=1;
A[2][1]=1;
A[2][2]=1;
A[2][9]=1;
A[3][2]=1;

```

```

A[3][3]=1;
A[4][3]=1;
A[4][4]=1;
A[5][4]=1;
A[5][5]=1;
A[5][6]=1;
A[6][6]=1;
A[6][7]=1;
A[6][8]=1;
A[6][9]=1;
for(i=0;i<M;i++)
{ k=0;
  for(j=0;j<N;j++)
  {
    if(A[j][i]==1){g[k]=j;
                    k++;
                  }
    if (k==2){G[g[0]][g[1]]=1;
              G[g[1]][g[0]]=1;
              goto NEXT;
            }
  }
  NEXT:;}
for(i=0;i<N;i++)
{printf("\n");
  for(j=0;j<N;j++) printf(" %d ",G[i][j]);
}
}

```

Appendix 3

ALG1

```

// GRAPH A1    6/9
// prepared by: Wael H. Saafin

#include<stdio.h>
#include<time.h>
#include<dos.h>
#include<stdlib.h>
#include<graphics.h>
void far clearviewport(void);
void main(void)
{
long int ns,NS,phi,phi1;
int i,j,no,v,k1,p1,N,K,M;
int g[6][6],gs[6][6],w[6];
int f[9]={0,1,2,3,4,5,0,1,2};
int t[9]={1,2,3,4,5,0,2,3,4};
int k[6]={0,1,2,3,4,5};
float r[6][6],p,Fk,ns1;
long float Pk;
time_t tt1,tt2,tt3;
long float t1,t2;
// % g is the original network
// % gs is the sampled network
tt1=time(NULL);

```



```

N=6;

M=9; // # of edges

K=6;

ns=0;

phi=0; //

NS=300; // % # of samples

STEP1: // in this step the connection matrix is entered together with
        // the probability matrix element by element
    for (i=0;i<N;i++)
    { for (j=0;j<N;j++){ g[i][j]=0;
                        gs[i][j]=0;
                        r[i][j]=.4;}}

    for(i=0;i<M;i++)
    {if(r[i]<t[i])g[r[i]][t[i]]=1;
     else g[t[i]][r[i]]=1;
    }

    tt2=time(NULL);

STEP2: ns++; // Here a simulated version is calculated
    for (i=0;i<N;i++)
    { for (j=i+1;j<N;j++)
      {if(g[i][j]==0)gs[i][j]=g[i][j];
      else{ p1=random(1000);
            p=p1/1000.;
            if (p>=r[i][j])gs[i][j]=0;
            else gs[i][j]=g[i][j];
          }
      }
    }
}

```

STEP3: //Begin the inspection process

i=0;

k1=0;

w[0]=k[0];

no=0;

STEP4: v= w[no];

no++;

```
for (j=0;j<K;j++) {if(v==k[j]){ k1++;
                                goto step41;}
```

```
}
```

step41: if(k1==K)goto STEP9;

STEP5: for (i=no-2;i>=0;i--)

```
{ if(v>w[i]) {if (gs[w[i]][v]==1) gs[w[i]][v]=0;}
  if(v<=w[i]) {if (gs[v][w[i]]==1) gs[v][w[i]]=0;}
}
```

STEP6: for(i=0;i<N;i++)

```
{ if (v>i) {if (gs[i][v]==1) {w[no]=i;
                                gs[i][v]=0;
                                goto STEP4;
                                }
```

```
}
```

```
if(v<=i) {if (gs[v][i]==1) {w[no]=i;
                                gs[v][i]=0;
                                goto STEP4;
                                }
```

```
}
```

```
}
```

STEP7: // no adjacent edges

```

for (j=n0-2;j>=0;j--)
{ for (i=0;i<N;i++)
  { if (gs[w[j]][i]==1||gs[i][w[j]]==1) { v=w[j];
                                          goto STEP6;
                                          }
  }
}
STEP8:// There is at least one cut in the graph
goto STEP10;
STEP9: phi++;
STEP10:printf("\n phi= %d",phi);
        if(ns<NS) goto STEP2;
STEP11:phi1=phi;
        ns1=NS;
        Pk=phi1/ns1;
        Fk=1-Pk;
tt3=time(NULL);
t1=tt2-tt1;
t2=tt3-tt2;
printf("\n The propability of connection %f \n And of failure %f \n
",Pk,Fk);
printf("The time taken for entering the network was %f",t1);
printf(" \n and for finding the Rel. was %f",t2);

}

```

Appendix 4

ALG2

```

// A1 6/9
// prepared by: Wael H. Saafin
#define N 6 // # of nodes
#define M 9 // # of links
#define inf 9999 .
#include<dos.h>
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
#include<values.h>
int s,t,Bool;
int gs[N][N];
void BFS(void);
void main(void)
{
long int ns,NS,phi;
int p1,l,j,no,i,kmax;
int g[N][N],gs[N][N],w[N];
float r[N][N],p,phi1,Pk,Fk;
int d,dd,e,Bool;
int dist[N],pred[N],pp[M];
time_t tt1,tt2,tt3;
float t1,t2;
int f0[M]={0,1,2,3,4,5,0,1,2};

```

```

int t0[M]={1,2,3,4,5,0,2,3,4};
int k1[6]={0,1,2,3,4,5};
tt1=time(NULL);
ns=0;
phi=0;
NS=50000;
kmax=6; // all the terminals are k-nodes
STEP1: // in this step the connection matrix is entered together with
        // the probability matrix element by element
for (i=0;i<N;i++)
{ for (j=0;j<N;j++){ g[i][j]=0;
                    gs[i][j]=inf;
                    r[i][j]=.4;}}
for(i=0;i<M;i++)
{if(f0[i]<t0[i])g[f0[i]][t0[i]]=1;
 else g[t0[i]][f0[i]]=1;
}
tt2=time(NULL);
STEP2: ns++; // Here a simulated version is calculated
no=0;
for (i=0;i<N;i++)
{ for (j=i+1;j<N;j++)
    {if(g[i][j]==0)gs[i][j]=inf;
    else{ p1=random(1000);
         p=p1/1000.;
         if (p>=r[i][j])gs[i][j]=inf;
         else{ no++;
              gs[i][j]=no;}
    }
}

```

```

    }
  }
}
STEP3: for (l=0;l<kmax-1;l++)
  {s=k1[l];
  t=k1[l+1];
STEP11: for (i=0;i<N;i++)
  { dist[i]=inf;
  pred[i]=-1;
  }
  dist[s]=0;
  d=-1;
STEP12: if(pred[t]!=-1)goto STEP14;
  d++;
  Bool=0;
  for(i=0;i<N;i++)
    {if(dist[i]==d) { dd=d+1;
      for(j=0;j<N;j++)
        { if(gs[i][j]==inf && gs[j][i]==inf)goto NEXT;
          if(dist[j]<=dd)goto NEXT;
          dist[j]=dd;
          pred[j]=i;
          Bool=1;
          NEXT::
        }
      }
  }
}
STEP13: if(Bool==0)goto STOP; //no path exists

```

```

goto STEP12;
STEP14: //for(i=0;i<M;i++) pp[i]=0;
    j=t;
STEP15:if(j==s)goto RETURN;
    i=pred[j];
    //e=gs[i][j];
    //pp[e]=1;
    j=i;
    goto STEP15;
RETURN::
}
STOP:  if(Bool!=1) goto STEP5; //no path exists
STEP4:phi++; // connected
STEP5:
    if(ns<NS)goto STEP2;
STEP6:phi1=phi;
    Pk=phi1/NS;
    Fk=1-Pk;
    tt3=time(NULL);
    t1=tt2-tt1;
    t2=tt3-tt2;
    printf("\n The reliability is %f\n prob of failure %f",Pk,Fk);
    printf("\n The time taken for entering the network was %f ",t1);
    printf("\n and for finding the rel. was %f ",t2);
}

```

Appendix 5

Network Generator

```

// note that the numbering here starts from 1 not zero
// prepared by: Wael H. Saafin
// m is the # of edges
// n is the # of nodes
#include<dos.h>
#include<time.h>
#include<stdio.h>
//#include<b.c>
void main(void)
{
float tt;
time_t tt1,tt2;
int f[200],t[200];
int h,m,n,i,j,k,mics;
tt1= time(NULL); //tt1 *timer);
printf("\n\n enter the # of edges (m) then the # of nodes (n) \n");
scanf("%d %d",&m,&n);
h=m/n;
for (i=0;i<=h;i++)
{for (j=1;j<=n;j++)
{ k=j+i*n;
f[k]=j;
if(i+j<n)t[k]=i+j+1;
else t[k]=i+j+1-n;
}
}
}

```



```
    }  
  }  
  printf(" f[i]      t[i]\n-----      -----\n",m,n);  
  for (i=1;i<=m;i++)  
  printf("%d, ",f[i]);  
  printf("\n-----\n");  
  for(i=1;i<=m;i++)  
  printf("%d, ",t[i]);  
  tt2= time(NULL); //tt2 *timer);  
  tt=tt2-tt1;  
  printf("\n %f",tt);  
}
```

ملخص

اعتمادية شبكات الاتصالات لعدد (ك) من الأطراف

اعداد

وانل حسن عبدالله السعافين

المشرف

د. بسام كحالة

المشرف المشارك

د. د. جميل أيوب

من الحقول الهامة التي تعالج شبكات الاتصالات موضوع اعتمادية هذه الشبكات. هذه الرسالة تدرس اعتمادية الشبكة لعدد (ك) من الأطراف, وهي الاعتمادية التي تبحث في احتمالية بقاء مجموعة معينة من الأطراف متصلة مع بعضها البعض عبر قنوات اتصال.

الطرق التي تعالج هذه المشكلة هي اما حقيقية أو تقريبية. الحل الحقيقي عادة ما يكون غير عملي لأنه لا ينطبق الا على الشبكات الصغيرة, حيث أن هذه الطرق تتطلب وقتا كبيرا لحساب الاعتمادية.

في هذه الرسالة, نقتراح خوارزميتان تستخدمان الطرق التقريبية في الحل. هاتان الخوارزميتان تقدمان حلا للشبكات عامة في وقت معقول و عملي وفي نفس الوقت تقدم حلا دقيقا قريبا من الحقيقي. هاتان الخوارزميتان تستخدمان طريقة مونتني كارلو للتمثيل (Monte Carlo Simulation) والتي تقوم بتنفيذ عدد من المحاولات لحساب الاعتمادية. في كل محاولة, بعض الوصلات تعتبر لاغية و باحتمالية تعتمد على مدى الاعتمادية الخاصة بكل وصلة على حدة. بعد افتراض سقوط بعض هذه الوصلات تستخدم طريقة في البحث للتأكد من بقاء الاطراف المعنية متصلة مع بعضها أم لا. في الخوارزمية الاولى والمسماة (ALG1) نستخدم طريقة (depth-first

(search) للبحث بينما في الخوارزمية الثانية و المسماة (ALG2) فان
طريقة (breadth-first search) للبحث هي المستخدمة للتحقق من
اتصال الاطراف المعنية مع بعضها البعض .
اذن فنحن نقترح حلا عاما للشبكات عامة دون وضع قيود على شكل أو
حجم هذه الشبكات .