Lower Bounds for Two-Terminal Network Reliability

by

Timothy Benedict Brecht

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, 1985

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

One measure of two-terminal network reliability, termed probabilistic connectedness, is the probability that two specified communication centers can communicate. A standard model of a network is a graph in which nodes represent communications centers and edges represent links between communication centers. Edges are assumed to have statistically independent probabilities of failing and nodes are assumed to be perfectly reliable. Exact calculation of two-terminal reliability for general networks has been shown to be #P-complete. As a result is desirable to compute upper and lower bounds that avoid the exponential computation likely required by exact algorithms.

Two methods are considered for computing lower bounds on two-terminal reliability in polynomial time. One method uses subgraph counting techniques and estimates of subgraph counts to obtain the Kruskal-Katona bounds. The development of the Kruskal-Katona bounds is outlined along with a method for improving the bounds. The other method uses the edge-disjoint paths of a network. Different techniques for finding edge-disjoint paths are discussed, as well as how the choice of paths affects the bound.

The different methods for computing lower bounds are compared and the advantages and disadvantages of each are outlined. Results of the thesis demonstrate that the minimum cost maximum flow technique of finding edge-disjoint paths produces a bound that typically outperforms the other bounds. However, each of the methods investigated has the potential for producing the best bound under certain circumstances. Thus a linear programming technique is suggested to combine all of the bounds to obtain a bound that is at least as good as the best of the bounds and occasionally better.

Acknowledgements

Many people have played important roles in helping me complete my thesis. In particular, my supervisor, Charlie Colbourn, introduced me to the area and provided me with guidance, understanding and support. For these things I thank him. I would also like to express my gratitude to Charlie for reading the numerous drafts of this thesis so carefully and quickly. It really makes a students life much easier knowing that your supervisor will likely have the latest draft of your thesis read over-night.

I would like to thank the members of my committee, Mike Ball and Bill Pulleyblank, for their time and insightful comments regarding this research. I would also like to thank Ehab El Mallah (a fellow graduate student here) who took time from his busy schedule to read my thesis, detect typographical errors and provide valuable comments on the contents and presentation of the material presented here.

I would also like to thank Eric Neufeld, Andre Paradis, and Aparna Ramesh for providing helpful comments and interaction with regards to this thesis. A very special thanks goes to Daryl Harms, for providing me with a foundation for this thesis as well as his programs which saved me arbitrarily many hours of time and headaches.

Most importantly I would like to thank the staff and graduate students at the University of Waterloo; the staff for their constant support and making the impossible, ‘‘doable’’ and the graduate students for providing a rich, stimulating and above all fun environment in which to work and live.

I shall look back on my days at Waterloo and fondly remember the numerous friends that I have made and the good times that we have had together. I thank them for the most important thing of all; helping Barb and I to get together. Barb has been a constant inspiration in my life. She deserves so much of the credit for helping me to complete this thesis. I thank her for coming into my life, for consoling me when I things looked grim, and for loving me. I also thank her for providing me with access to her numerous accounts, without which no one can complete a thesis. Without Barb this thesis would never have been completed.

I would like to thank my parents and family for providing me with more encouragement and love than anyone could ever ask or hope for.

Lastly, I thank the University of Waterloo, Charlie Colbourn and my parents for providing me with financial support. For financial support for conferences I thank Charlie Colbourn and Rick Bunt.

Chapter 1

Introduction

1.1. Introduction

The use of communication networks has spread tremendously during recent years. The existence of large computer communication networks such as the Arpanet, Telenet, Datapac, CSNET and the USENET has been complemented by a proliferation of smaller scale or local area networks. With the widespread use of and dependence upon such networks, it becomes imperative for these networks to be highly reliable.

A major problem lies with determining the reliability of a given network. It is desirable to be able to obtain a quantitative measure of a given network's reliability. One common method for measuring the reliability of a network is to associate a statistical probability of failure with each of the components of the network in order to obtain a statistical measure of the overall reliability of the network. This notion supports an accepted definition of reliability as: ''the probability that a given system or device is operational''. This measure of reliability may be interpreted as a long term average availability. That is, over a specified period of time, what is the probability that the network will remain operational? This also includes a fairly prevalent notion of reliability as the probability that a network is operational at any given moment. To avoid conflicts that arise with various levels of operation within a network's hierarchy, only the topology of the network is considered. This allows a network to be modelled by a graph where the communication centers are represented by the nodes of the graph and communication links are represented by its

edges. Thus, a network is often considered to be operational when each of k specified communication centers is able to communicate with each of the other specified centers in the network.

Two values of k that are of particular interest are $k = n$ (all nodes) and $k = 2$. These reliability problems are referred to as all-terminal reliability and two-terminal reliability, respectively. In the all-terminal problem the network is considered operational if all centers can communicate with every other center in the network. In the two-terminal reliability problem the network is considered operational if the two specified nodes, often called the source, *s*, and the sink or target, *t*, are able to communicate.

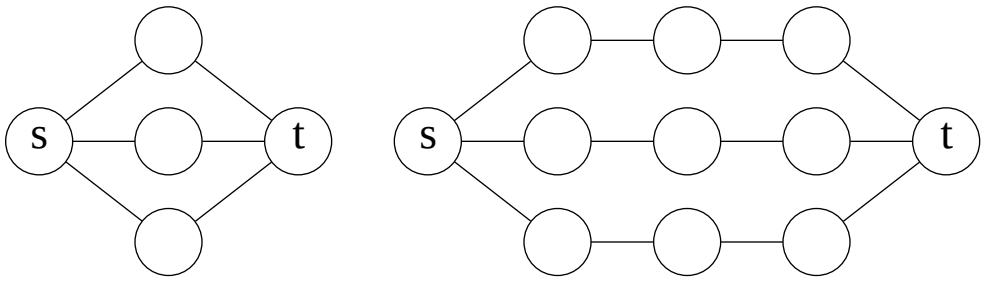
1.2. Measuring Reliability

Frank and Frisch [20] and Wilkov [48] provide surveys of the various definitions of reliability. They identify two distinct classes of reliability measures: deterministic and probabilistic.

The deterministic criteria make use of discrete measures to define the reliability of a network. The assumption made when dealing with deterministic measures is that the network is to be subjected to a destructive force or enemy who has complete knowledge of the topology of the entire network. The purpose of this intelligent enemy is to destroy or disrupt network communication. Thus the main measure of reliability is the least amount of damage the enemy must inflict to render the network inoperative. Deterministic measures can also be viewed as a simple bound on the reliability of the network, since they are often measures of a network's worst-case vulnerability to failure.

For example, in the two-terminal problem, two deterministic measures of reliability are the number of edges and the number of nodes that must be destroyed or removed to disrupt communication between the specified nodes. The minimum number of edges that must be removed in order to disconnect the nodes *s* and *t* is simply the number of edges in a minimum cardinality (s,t)-cut. The minimum number of nodes that must be removed to disconnect *s* and *t* is the (node) connectivity between the vertices *s* and *t*. Both of these measures are computable in polynomial time.

One of the main problems with deterministic measures is that they give rise to some counterintuitive notions of network reliability. For example, consider the graphs shown in Figure 1.1.



(a) (b)

Figure 1.1

According to one deterministic measure that uses (node) connectivity as a measure of the graph's reliability, the graphs of Figure 1.1a and 1.1b are equally reliable since the (s,t)-connectivity of each graph is three. However, intuition leads one to believe that

graph (a) is the more reliable of the two.

The same problem arises when the cardinality of a minimum (s,t)-cutset is used as a measure of reliability. Consider the graphs shown in Figure 1.2.

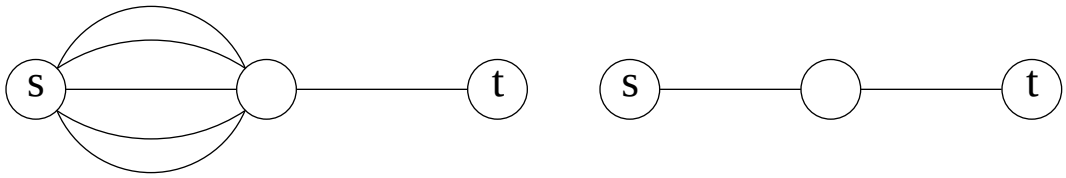


Figure 1.2

Both graphs (a) and (b) have a minimum cardinality (s,t)-cut of size one. Some deterministic measures therefore imply that both are equally reliable. This is again counterintuitive, as one expects graph (a) to be the more reliable of the two. This leads to the notion that a more intuitively acceptable measure of reliability might be a probabilistic measure.

1.3. Probabilistic Reliability

The probabilistic methods for measuring the reliability of a communication network generally assume that the failure of edges and/or nodes are random events. Using predetermined probabilities that the edges and/or nodes are operational, the probability that the network remains operational is computed. A network is considered operational if it is connected. The probability that the network is connected is often called probabilistic con-

nectedness. This probabilistic model is often more appropriate than the deterministic model since it results in a probability that the network is connected at any point in time.

The model used is a probabilistic graph consisting of n nodes representing communication centers and b edges representing links between the communication centers. The probabilistic connectedness of a graph is denoted by R.

Two major assumptions are made in order to make the problem of computing probabilistic connectedness more tractable. The first assumption is one of statistical independence of edge failures. The second assumption is that nodes are perfectly reliable. That is, their probability of failure is zero.

The assumption that edge failures are statistically independent implies that the probability of a link being operational is independent of the states of the other links in the network. The assumption is that link failures are caused by random events and that all links are affected individually. When modelling a real communication network, this assumption may not be valid, since links in one particular area may fail due to natural causes such as a major storm or earthquake. However, this assumption is often made because information about all dependencies of link failures is extremely difficult to obtain. Consequently, such dependencies may not be known.

Without the assumption of statistical independence the problem becomes much more difficult. As a result this assumption is often made, even when modelling networks for which it is known that link failures are not independent. One such example observed by Harms [22] is Lee's model of telephone crossbar switching networks [32].

The assumption that nodes are perfectly reliable may, at first, seem unreasonable. However, there are reasons for adopting it. Node failures necessarily induce edge failures, and moreover introduce failures that are statistically dependent. Hence the assumption of statistical independence requires the assumption of perfectly reliable nodes. If it is necessary to make the assumption that nodes do fail, some of the methods discussed in this thesis could be modified in order to accommodate the assumption.

Much of the development in the area of network reliability has been done under the assumption of perfectly reliable nodes. Moore and Shannon [35] make this assumption in their work on the reliability of telephone switching networks. They model electromechanical relays, where relay contacts sometimes fail due to dust particles or mechanical problems. The connections or circuits between these relays are highly reliable and are therefore assumed to be perfectly reliable. The relay contacts are modelled with the edges of a graph and the connections between the relays are modelled using the nodes of a graph.

All known bounds for all-terminal reliability and for two-terminal reliability make the assumption that nodes are perfectly reliable, typically for reasons of tractability. It is for these reasons, as well as for comparison with existing bounds, that the assumption is used in this thesis.

One might also argue that this assumption can be made for large communication networks since communication centers are often composed of dedicated processors. Such a processor may have a backup or stand-by processor which assumes its responsibilities should it fail.

This definition of network reliability has many applications outside the realm of computer communication networks. For example the same model can be applied to obtain reliability measures of electrical systems, telephone networks, digital networks and electrical power networks, to mention a few.

1.4. Two-Terminal Lower Bounds

Unfortunately there are no known polynomial time algorithms to compute two-terminal reliability exactly. The best known algorithm to compute the reliability exactly requires $O(3^{n-2})$ steps [9]. It is also likely that no polynomial time algorithm exists for general networks, since Valiant [45] has shown this problem to be NP-Hard. In a later paper, Provan and Ball [40] use different means to derive the same result.

Since computing the exact reliability of general networks is computationally infeasible, alternative approaches must be considered. If an exact measure of reliability is required, subclasses of graphs can be examined for which reliability can be computed efficiently (that is in polynomial time). For example, Wald and Colbourn [47] and Satyanarayana and Wood [41] have shown that the reliability of series-parallel networks can be computed efficiently. El Mallah and Colbourn [15] have shown that the *k*-terminal reliability of Δ-Y reducible networks is also efficiently computable. Both series-parallel networks and Δ-Y reducible networks are subnetworks of the larger class of planar networks. Unfortunately, the exact computation of the two-terminal reliability of planar networks has been shown to be #P-complete [39].

For general networks the intractability of the problem necessitates approximation

techniques. One approach taken by Karp and Luby [26] is the use of Monte Carlo methods for estimating reliability. Another approach is to bound the reliability of the network to produce strict upper and lower values to guarantee that the exact reliability is within the resulting range. Since the motivation for bounds is to avoid exponential computation and obtain a method that can be used for large scale networks, the focus of this thesis is on bounds that are computable in polynomial time.

Many methods exist for the efficient computation of bounds for all-terminal reliability. Harms [22] provides an excellent survey as well as an analytical and practical comparison of these bounds. There has been some work conducted on the problem of computing bounds on two-terminal reliability. Unfortunately, for general networks most of these bounds are not computable in polynomial time. There has been no substantial survey or comparison of efficient methods used for computing bounds on two-terminal reliability. Hence the focus of this thesis is on computing polynomial time lower bounds for two-terminal reliability.

The reason that only lower bounds are discussed is straightforward. A lower bound on the reliability of a network is a much more useful and more important measure than is an upper bound. If a lower bound is known it follows that the network is at least as reliable as the value obtained. However, should only an upper bound be known, all that can be conjectured is that the network is less reliable than the obtained value.

1.5. History of Two-Terminal Reliability

A number of methods for computing bounds have been proposed. Many of these

bounds are based on one of state, cut, or path enumeration. Esary and Proschan [16] have developed bounds based on path and cut enumeration. Messinger and Shooman [34] have developed two sets of bounds, one based on cut enumeration and the other based on path enumeration. However, these bounds are not as tight as the Esary-Proschan bounds. Jensen [24] has proposed bounds that are based on state enumeration which can theoretically be made as tight as desired. Although, in practice these bounds are usually not as good as the Esary-Proschan bounds.

Shogun [42] uses the concept of strongly connected components to sequentially compute bounds on the reliability of a given network. These bounds have been shown to be analytically tighter than the Esary-Proschan bounds, making them the tightest known bounds.

All of the above mentioned bounds perform well in practice. However, for general networks they require the enumeration of all paths, cuts or states, the number of which grows exponentially with the size of the graph. Although these algorithms do offer a savings in computation in comparison with exact algorithms, they are all still exponential and are therefore not very useful for large scale communication networks.

1.6. Polynomial Time Bounds

This thesis deals with polynomially computable bounds because they are generally more useful for practical application to large communication networks. There are two previously known polynomial time methods for computing the two-terminal reliability: a bound based on linear programming and a bound based on subgraph counting. The linear

programming bounds were originally developed by Zemel [49] and were later expanded by Assous [2]. The method used to obtain these bounds has some theoretical interest but in practice does not produce very strong bounds. Assous tightens the bounds somewhat by introducing bounds on the joint probability of each pair of edges failing. These bounds are known as the second order linear programming bounds. Unfortunately the second order bounds are not much more accurate than the first order bounds, especially under the assumption of statistical independence of edge failures.

The other known efficient bound is a bound which Van Slyke and Frank [46] show can be developed by making use of a theorem developed independently by Kruskal [30] and Katona [27]. This bound, which is referred to as the Kruskal-Katona bound makes use of subgraph counting techniques and some combinatorial properties to obtain reasonably good bounds. The computational methods involved in computing this bound are described along with the development of methods for improving the bound.

A third class of bounds is developed in this thesis that exploits the extensive pool of graph theoretic research. The idea for the new bound was sparked by the Lomonosov-Polesskii all-terminal lower bound [33] which uses edge-disjoint minimal subgraphs. The two-terminal equivalent makes use of edge-disjoint paths between the specified source and target nodes. A similar technique has been used by Carey and Hendrickson [10] to bound expected flow in a transportation network. Different methods for obtaining the paths and how the choice of paths affects the corresponding bound are discussed. Finally the new set of edge-disjoint path bounds is compared with the subgraph counting bounds.

Chapter 2

Definition of the Model

2.1. Graph Theoretic Definitions

Since graph theoretic definitions are not standardized the definitions used in this thesis are presented in this section. Further definitions can be found in graph theory texts such as [6] or [21].

A network is modelled using a probabilistic graph consisting of nodes representing communication centers and edges representing links between the communication centers. A *graph* $G = (V,E)$ consists of a set of *nodes* or *vertices* V and a set of *edges* E. Undirected edges are used to indicate two way communication links between two nodes. They are represented as unordered pairs (v_i, v_j) where v_i and v_j represent the nodes which are joined by the communication link or edge. An edge is said to be *incident* upon two nodes if the two nodes are joined by the edge. A graph which consists of undirected edges is called an *undirected graph*. The graph in Figure 2.1 is an example of a undirected graph where,

$G = (V,E),$
$V = \{s,t,r,h,w,x,y,z\}$
$E = \{(s,r),(s,x),(s,z),(r,h),(h,w),(h,x),(h,y),(w,t),(x,y),(z,y),(y,t)\}.$

This graph is used to illustrate many of the definitions in this chapter.

The cardinality of set X is denoted $|X|$. The number of nodes in a graph is denoted by $|V|$ or n, while $|E|$ or b denotes the number of edges in a graph. For the graph in Figure 2.1,

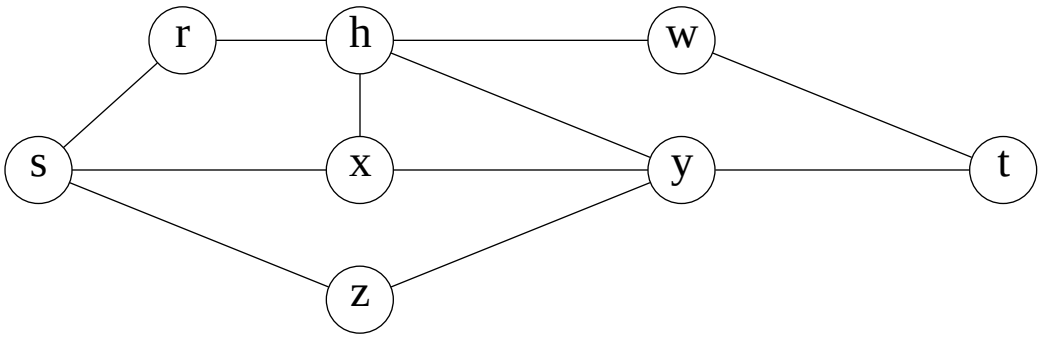


Figure 2.1

$|V| = n = 8$ and $|E| = b = 11$.

A *loop* or *self edge* is an edge that originates and terminates at the same node. *Multiple edges*, *multi-edges* or *parallel edges* occur when two or more edges are incident upon the same pair of nodes. A *simple graph* is one which contains no loops or parallel edges. A graph is *planar* if it can be drawn on the plane so that no edges cross. Unless otherwise stated all graphs are assumed to be undirected simple graphs.

The *all-terminal reliability* problem deals with communication among all nodes in a network. For all nodes to be able to communicate the graph must be connected. A graph is *connected* if there is at least one path between every pair of nodes. The *two-terminal reliability* problem deals with communication between two specified nodes. The specified nodes are referred to as the *source* node *s*, and the *sink* or *target* node *t*. The *probabilistic connectedness* measure of two-terminal reliability is the probability that the specified nodes *s* and *t* are connected. The nodes are *connected* if there exists at least one path between them.

A *walk* is a sequence of nodes $(v_1, v_2, \cdots, v_n)$ in which (v_i, v_{i+1}) is an edge of the graph. An example of a walk from node *s* to node *t* is (s,r,h,x,y,h,w,t). A *path* is a walk in which all edges and all vertices on the walk are unique with the possible exception of the first and last nodes. The walk (s,r,h,y,t) is a path. A *cycle* is a path that begins and ends at the same vertex. An example of a cycle containing node *y* is (y,h,x,s,z,y). An (s,t)-*path,* is a path from node *s* to node *t*. An (s,t)-path of the graph shown in Figure 2.1 is (s,z,y,x,h,w,t). *Edge-disjoint paths* are paths with no edges in common. *Node-disjoint paths* are paths which share no common nodes other than the source and target nodes. For example, (s,r,h,w,t) and (s,x,h,y,t) are edge-disjoint paths while (s,r,h,w,t) and (s,x,y,t) are node-disjoint paths.

Edges frequently have some form of ‘‘weight function’’ associated with them which often represent a physical property of the link. Properties such as the cost of an edge (i,j), denoted *cost*(i,j) or the length of an edge (i,j), denoted *length*(i,j) are often used when modelling communication networks.

A *minimum length* (s,t)-*path* or *shortest* (s,t)-*path* is a path from node *s* to *t* of minimum length. Such a path may also be referred to as a *shortest path* or *minimum length path.* In the case when edges are assumed to have equal lengths a shortest path is simply an (s,t)-path of minimum cardinality. Assuming all edges are of equal length, (s,x,y,t) and (s,z,y,t) are both shortest paths of the graph in Figure 2.1.

An (s,t)-*pathset* or *pathset* is a set of edges that contains at least one path from *s* to *t*. It may also contain other edges. A *pathset of size x* is a pathset that contains *x* edges. Examples of some pathsets of size five in Figure 2.1 are:

$\{(s,x),(x,y),(y,t),(s,z),(z,y)\}$

$\{(s,r),(r,h),(h,y),(y,t),(h,x)\}$

$\{(s,r),(r,h),(h,x),(x,y),(y,t)\}$

A *network cut* or *cutset* is a set of edges whose removal disconnects the graph. A *minimum cardinality cutset* is a cutset that contains the fewest edges. The *edge-connectivity* of a graph is the size of a minimum cardinality cutset. An (s,t)-*cutset* or (s,t)-*cut* is a set of edges whose removal disconnects the nodes *s* and *t*. The set of edges $\{(r,h),(s,x),(s,z),(y,t)\}$ form an (s,t)-cut. A *minimal* (s,t)-*cutset* is a minimal set of edges whose removal disconnects the nodes *s* and *t*. The above (s,t)-cutset $\{(r,h),(s,x),(s,z),(y,t)\}$ is not minimal, since the edge (y,t) can be removed from the set while the remaining edges still form an (s,t)-cutset. The set $\{(r,h),(s,x),(s,z)\}$ does form a minimal (s,t)-cutset. A *minimum cardinality* (s,t)-*cutset* is an (s,t)-cutset that contains the fewest edges. The number of edges in a minimum cardinality (s,t)-cutset is denoted by *c*. The set of edges $\{(w,t),(y,t)\}$ form a minimum cardinality (s,t)-cutset of size two.

A *subgraph* of *G* is a graph whose nodes and edges are contained in *G*. That is, $G' = (V', E')$ is a subgraph of *G* if $V' \subseteq V$ and $E' \subseteq E$. An example of a subgraph of the graph in Figure 2.1 is $G' = (V', E')$, where $V' = \{s,r,h,w,y\}$ and $E' = \{(s,r),(r,h),(h,w)\}$. If the subgraph contains all nodes of the graph the subgraph is a *spanning subgraph*, since it spans all nodes of the graph.

A graph that contains no cycles is called an *acyclic* graph. An acyclic connected graph is called a *tree*. The set of edges $\{(s,r),(s,x),(s,z)\}$ forms a tree. A *spanning tree* is a tree that spans all nodes of the graph. It is a connected spanning subgraph of the graph

and contains $n-1$ edges. The edges $\{(s,r),(s,x),(s,z),(x,h),(h,y),(h,w),(y,t)\}$ form a spanning tree of the graph in Figure 2.1. A spanning tree is a minimal spanning subgraph since removal of any one of its edges disconnects the subgraph. *Edge-disjoint spanning trees* are spanning trees with no edges in common. The graph in Figure 2.1 does not contain a pair of edge-disjoint spanning trees.

The *connectivity* of a graph is the minimum number of nodes that must be removed in order to disconnect the graph or reduce it to a single vertex. The (s,t)-*connectivity* of a graph is the minimum number of nodes that must be removed from the graph to disconnect the nodes *s* and *t*. The (s,t)-connectivity of the graph in Figure 2.1 is two since the removal of the nodes *w* and *y* disconnects the nodes *s* and *t*. A *cutpoint*, is a node whose removal disconnects the graph, or separates it into two or more components. The graph in Figure 2.2 contains two cutpoints *y* and *z*. A graph is *biconnected* if and only if it contains no cutpoint. A *biconnected component* of a graph is a maximal biconnected subgraph. Figure 2.2 contains three biconnected components consisting of the following sets of edges:

$\{(s,w),(s,x),(w,x),(w,y),(x,y)\}$

$\{(y,z)\}$

$\{(z,r),(r,t),(z,t)\}$

The *degree* of a node *v*, is the number of edges that are incident upon *v*. If all nodes in the graph are of the same degree the graph is *regular*. Two nodes are *neighbours* if there exists a common edge between them. Two nodes that are neighbours are *adjacent*. A *complete graph* is a graph in which all pairs of nodes are adjacent. A complete graph on *n* vertices is regular of degree $n-1$ and is denoted K_n. Figure 2.3a depicts K_4, the

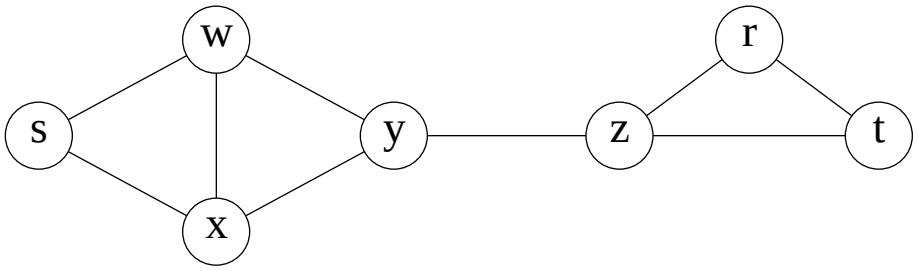
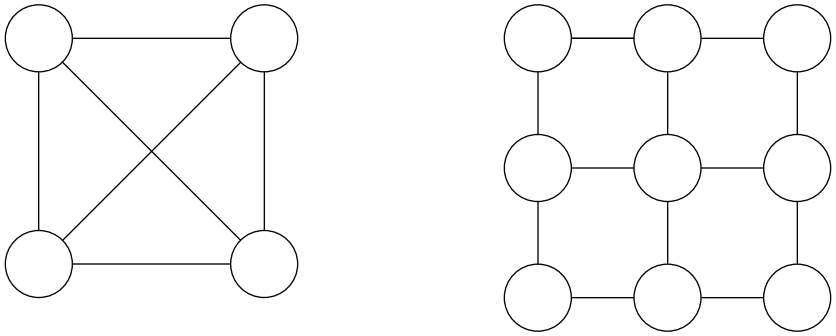


Figure 2.2

complete graph on four vertices.



(a) (b)

Figure 2.3

A *grid graph* of dimensions $m \times n$, is m nodes wide and n nodes high. Figure 2.3b shows an example of a 3×3 grid graph.

A *directed graph*, or *digraph*, is one in which edges are directed. Directed edges are referred to as *arcs*. Arcs represent one way communication links between two nodes with communication taking place in the direction that the arc points. A digraph $G = (V,A)$ consists of a set of nodes or vertices V and a set of arcs A. An arc from v_i to v_j is represented

as an ordered pair $<v_i,v_j>$, v_i is called the *tail* and v_j is called the *head* of the arc. Figure 2.4 is an example of a digraph where,

$G = (V,A)$
$V = \{(s,w,x,y,z,t\}$
$A = \{<s,w>,<s,x>,<w,y>,<y,z>,<x,w>,<x,z>,<y,t>,<z,t>\}.$

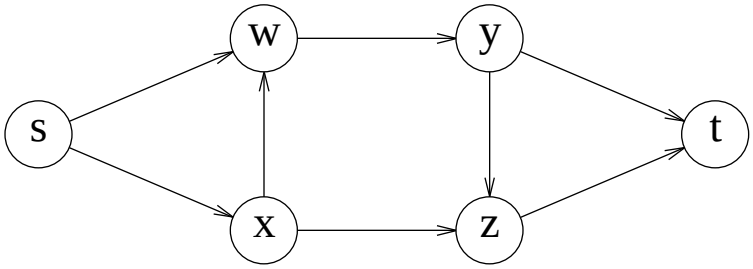


Figure 2.4

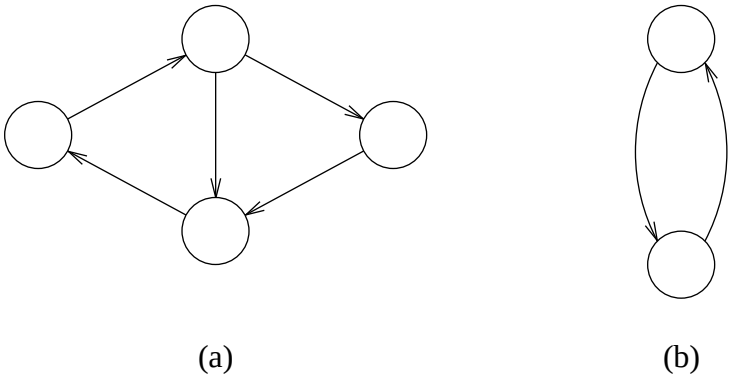
A *directed path* is an ordered sequence of nodes $<v_1,v_2,\cdots,v_n>$ in which $<v_i,v_{i+1}>$ is an arc of the graph. For example $<s,w,y,z,t>$ is a directed path from *s* to *t*.

In a directed graph, a *strongly connected component* is a maximal set of nodes for which there exists a directed path between every ordered pair of nodes in the component, such that the paths pass only through nodes that are also in the component.

Figure 2.5 shows two examples of strongly connected components and Figure 2.6 shows two examples of components that are not strongly connected.

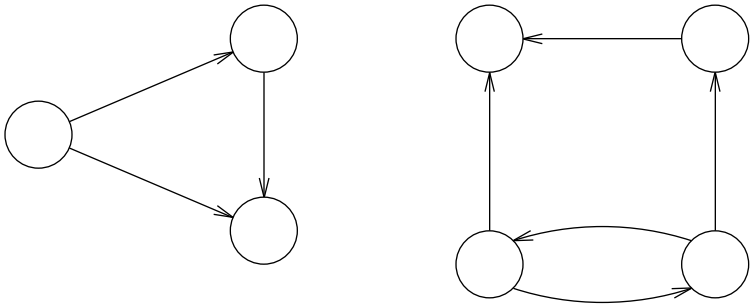
2.2. Operational Components

An edge or link is *operational* or *available* if communication is possible via this link. If the two nodes joined by a link are unable to communicate through the link it is in a *failed state* or simply *failed*. The probability that edge *i* is operational is denoted p_i. The



(a) (b)

Figure 2.5



(a) (b)

Figure 2.6

probability that edge *i* is failed is $q_i = 1 - p_i$.

An *operational subgraph* is a subgraph in which sufficient edges are operational that permit the sites in question to communicate. For example, in the all-terminal case a minimal operational subgraph is an operational spanning tree. For a spanning tree to be operational all of its edges must be operational (since it is a minimal operational

subgraph), and therefore a spanning tree is in a failed state if any of its edges fail. In the two-terminal case a minimal operational subgraph is an operational (s,t)-path. A path is operational if and only if all edges along the path are operational. Therefore, a path is in a failed state if any of the edges along it have failed. If the nodes *s* and *t* are disconnected the graph or network is in a failed state.

2.3. The Reliability Polynomial

Moore and Shannon [35] pioneered much of the work that has been done on the problem of probabilistic connectedness. They modelled electromechanical relays using edges to represent relay contacts and nodes to represent the perfectly reliable connections between the relays.

While investigating the problem of constructing arbitrarily reliable networks from arbitrarily poor components they developed the following important polynomial:

$$h(p) = \sum_{i=0}^{b} A_i \, p^i \, q^{b-i}$$

where $h(p)$ is the probability that the network is operational, p is the probability of a relay being operational, $q = 1 - p$ is the probability that the relay has failed. A_i is the number of ways one can select a subset of i of the b relays of the network such that if these i relays are operational and the remaining relays failed, the network is operational. This can also be written as:

$$h(p) = 1 - \sum_{i=0}^{b} B_i \, p^{b-i} \, q^i$$

where B_i is the number of subsets of i relays that exist, such that the network fails if these relays are failed and all others are operational.

Kel'mans [28] modified these polynomials to apply to the probabilistic connectedness of communications networks, to obtain the well-known *reliability polynomial*:

$$R(p) = \sum_{i=0}^{b} N_i \, p^i \, q^{b-i}$$

which may also be written as:

$$R(p) = 1 - \sum_{i=0}^{b} C_i \, p^{b-i} \, q^i$$

In the reliability polynomial N_i represents the number of ways that i of the b edges of the graph can be operational such that the graph is connected (in the two-terminal connection problem this is the number of pathsets containing i edges). C_i is the number of ways that i of the b edges can fail such that the graph is not connected. (i of the b edges form an (s,t)-cutset). The two equations are related by the identity $N_i + C_{b-i} = \binom{b}{i}$. Since both are positive, observe that $0 \le N_i \le \binom{b}{i}$ and $0 \le C_i \le \binom{b}{i}$.

Chapter 3

The Kruskal-Katona Bounds

3.1. Development

There are generally two methods used in computing bounds for reliability. One method makes use of subgraph counting techniques, and the other method involves finding edge disjoint subgraphs. The focus of this chapter is the subgraph counting method.

Harms [22] surveyed bounds for all-terminal reliability, and located four different bounds which use subgraph counting: the Jacobs bounds, the Bauer-Boesch-Suffel-Tindell bounds, the Kruskal-Katona bounds, and the Ball-Provan bounds. The four sets of bounds form a hierarchy with respect to accuracy, from the Jacobs bounds, which provide the least accurate bounds, to the Ball-Provan bounds, which provide the greatest accuracy. Unfortunately, the Ball-Provan bounds [3] do not apply in the two-terminal case, since their bound is applicable only to ‘‘shellable independence systems’’ (see [3]). Hence the Kruskal-Katona bounds are examined.

A system is *coherent* if the failure of any component can not cause a system that is in a failed state to become operational. In the all-terminal reliability problem a network in which nodes are perfectly reliable is a coherent system since the failure of edges can in no way cause a failed system to become operational. If nodes are not assumed to be perfectly reliable it is possible that a node could become isolated by the failure of an edge, thus rendering the network inoperative. However, should the isolated node then fail, the network could again be considered operational. This is clearly a counterintuitive notion and is

not considered coherent. In the two-terminal reliability problem a network is coherent even without the assumption of perfectly reliable nodes, since once the two specified nodes are disconnected the failure of an edge or node can in no way cause the two nodes to become connected.

Van Slyke and Frank [46] show that a theorem developed by Kruskal [30] and Katona [27] can be applied to obtain an upper and lower bound on the reliability polynomial for any coherent binary system. These bounds, which are known as the Kruskal-Katona bounds, employ the following combinatorial structures whose definition and notation are taken from [4].

3.1.1. K-canonical Representation of Non-Negative Integers

For any non-negative integer *m*, the *k-canonical representation* of *m* is given by $(m_k, m_{k-1}, \ldots, m_l)$ such that:

$$m = \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_l}{l}$$

where $m_k > m_{k-1} > \cdots > m_l \geq l \geq 1$.

The m_i can be computed successively in increasing order of i so that:

$$m_i = \max\left\{ x : \binom{x}{i} \leq m - \sum_{j=i+1}^{k} \binom{m_j}{j} \right\}.$$

For example, the 4-canonical representation of 10 is (5,4,2).

$$\binom{5}{4} + \binom{4}{3} + \binom{2}{2} = 10.$$

The 6-canonical representation of 10 is (7,5,4,3).

$$\binom{7}{6} + \binom{5}{5} + \binom{4}{4} + \binom{3}{3} = 10.$$

This *k*-canonical representation of any non-negative integer is unique [27].

3.1.2. The (i,k)th Lower Pseudopower

Let $(m_k, m_{k-1}, \ldots, m_l)$ be the *k*-canonical representation of an integer *m*. The *(i,k)th lower pseudopower* of $(m_k, m_{k-1}, \ldots, m_l)$ for $k \geq l \geq 1$ is:

$$\binom{m_k}{i} + \binom{m_{k-1}}{i-1} + \cdots + \binom{m_l}{i-k+l}$$

and is denoted by $(m_k, m_{k-1}, \ldots, m_l)^{i/k}$.

The (i,k)th lower pseudopower of *m* is denoted by $m^{i/k}$ and is used to refer to the lower pseudopower of the *k*-canonical representation of *m*. Kruskal and Katona specify that:

$$\binom{m}{r} = 0 \quad \text{if} \quad \begin{cases} m < 0 \\ r < 0 \\ r > m \\ m = r = 0 \end{cases}$$

An example of some (i,k)th lower pseudopowers are:

$$10^{3/4} = \binom{5}{3} + \binom{4}{2} + \binom{2}{1} = 18 \qquad 10^{4/6} = \binom{7}{4} + \binom{5}{3} + \binom{4}{2} + \binom{3}{1} = 54$$

$$10^{5/4} = \binom{5}{5} + \binom{4}{4} = 2 \qquad 10^{5/6} = \binom{7}{5} + \binom{5}{4} + \binom{4}{3} + \binom{3}{2} = 33$$

Let F_i denote the number of ways that *i* edges can fail such that the network remains operational. Note that F_i is equivalent to N_{b-i} of the reliability polynomial. Therefore the reliability polynomial can also be written as:

$$R = \sum_{i=0}^{b} F_i \, p^{b-i} \, q^i$$

Note that the following restrictions must hold for the F_i coefficients of the reliability polynomial:

$$F_0 = 1$$
$$0 \le F_{i+1} \le F_i^{\,i+1/i} \qquad i = 0,1,\ldots,b-(n-1).$$

Kruskal and Katona's theorem states that when $m = F_k$:

$a)$ $m^{i/k} \ge F_i$, when $i \ge k$.

$b)$ $m^{i/k} \le F_i$, when $i \le k$.

In the all-terminal case, these inequalities are used to obtain bounds in the following manner. Once a value for F_c, (the number of ways that c edges can fail such that the graph remains connected, where c is the size of a minimum cardinality cutset) is computed, equation (a) can be used to find overapproximations for the F_i values where $i \ge c$. Let n_c represent the number of ways that c edges can fail such that the graph is disconnected. That is, n_c is the number of minimum cardinality cutsets. Therefore the number of ways that c edges can fail such that the graph is connected is simply $F_c = \binom{b}{c} - n_c$. The number of minimum cardinality pathsets, $F_{b-(n-1)}$ (in this case spanning trees) and equation (b) can be used to find an underapproximation for the F_i values when $i \le b-(n-1)$. The number of minimum cardinality cutsets for undirected graphs can be calculated in polynomial time using an algorithm by Ball and Provan [4]. This algorithm makes use of results of Even and Tarjan [17] and Bixby [5]. The number of spanning trees can be efficiently counted using a result due to Kirchhoff [29]. A more recent paper by Brooks, Smith, Stone and Tutte [7] also presents this theory in a computational form. Knowing these values gives the following bounds for all-terminal reliability:

$$R \le \sum_{i=0}^{c-1} \binom{b}{i} p^{b-i} q^i + F_c\, p^{b-c} q^c + \sum_{i=c+1}^{d-1} {F_c}^{i/c}\, p^{b-i} q^i + F_d\, p^{b-d} q^d$$

$$R \ge \sum_{i=0}^{c-1} \binom{b}{i} p^{b-i} q^i + F_c\, p^{b-c} q^c + \sum_{i=c+1}^{d} {F_d}^{i/d}\, p^{b-i} q^i$$

where $d = b-(n-1)$.

3.1.3. The Two-Terminal Bounds

In the two-terminal reliability problem, F_i represents the number of sets of i edges that can fail in such a way that s and t can still communicate (that is i edges fail but there remains at least one path between s and t). The minimum cardinality pathsets in the two-terminal case are therefore (s,t)-paths of minimum cardinality, or simply, shortest (s,t)-paths. Suppose that the cardinality of a shortest (s,t)-path is l. If d now equals $b - l$ and d edges fail, then l edges remain operational. Therefore, if all edges along any shortest (s,t)-path are operational, the network remains operational. However, if $i > d$ edges fail, s and t can not possibly communicate since fewer than l edges are operational. Hence, $F_i = 0$ for $i > d$.

In the two-terminal case, c is the cardinality of a minimum cardinality (s,t)-cut. If fewer than c edges fail, s and t can surely communicate, since not enough edges have failed to form an (s,t)-cut. Hence $F_i = \binom{b}{i}$ for $i < c$.

The number of shortest (s,t)-paths, F_d, can be computed exactly in polynomial time using an algorithm developed by Ball and Provan [4]. However, calculating the number of minimum cardinality (s,t)-cuts, n_c, has been shown to be #P-complete for general graphs [40]. Therefore the computation of $F_c = \binom{b}{c} - n_c$ is also #P-complete. Nevertheless, the

value of n_c can be underestimated to obtain an overestimate for F_c. Since every graph must have at least one minimum cardinality (s,t)-cut $n_c \geq 1$ is an underestimate that can be used to obtain the overapproximation of $F_c \leq \bar{F}_c = \binom{b}{c} - 1$. The resulting bounding reliability polynomials are:

$$R \leq \sum_{i=0}^{c-1} \binom{b}{i} p^{b-i} q^i + \sum_{i=c}^{d-1} \bar{F}_c{}^{i/c} p^{b-i} q^i + F_d p^{b-d} q^d$$

$$R \geq \sum_{i=0}^{c-1} \binom{b}{i} p^{b-i} q^i + \sum_{i=c}^{d} F_d{}^{i/d} p^{b-i} q^i$$

where $d = b - l$.

These bounds are referred to as the basic Kruskal-Katona bounds or simply the KK0 bounds. Computation of the upper bound can be done in the same fashion as the lower bound with no additional effort, since only those values required to compute the lower bound are required to compute the upper bound. The upper bound has been implemented only for comparison with the lower bound and details of the upper bound are not discussed henceforth.

3.2. Motivation

The main reason that the Kruskal-Katona bound is of interest is that it performs reasonably well in the tests for computing all-terminal bounds conducted by Harms [22]. It is also the most accurate subgraph counting technique found in the literature that can be applied to the two-terminal problem. Another major reason for studying this method is the potential of improving the bound. It is anticipated that the bound can be improved by computing exact values for F_i for $c \leq i \leq d-1$. Improvements of this type are discussed in

greater detail later in this chapter.

3.3. Implementation Overview

Many of the implementation details of the Kruskal-Katona method are taken from Harms [22] and Ball and Provan [4].

Evaluation of the term $\sum_{i=0}^{c-1} \binom{b}{i} p^{b-i} q^{i}$ requires the calculation of a value for c, the size of a minimum cardinality (s,t)-cut. This can be done in polynomial time using the well known Max-Flow Min-Cut Algorithm of Ford and Fulkerson [18].

Evaluation of the term $\sum_{i=c}^{d} {F_d}^{i/d} p^{b-i} q^{i}$ for $d = b - l$, requires the computation of values for l, F_d and the (i,d)th lower pseudopower of F_d for $c \le i \le d$.

Computing a value for l can be done in polynomial time using Dijkstra's shortest path algorithm [12]. Ball and Provan's algorithm *pathcnt* [4] can be used to compute the number of (s,t)-paths of length l. In the same paper, they also describe how to compute the k-canonical representation of a non-negative integer m and how to compute $m^{i/k}$. They also prove that both of these operations can be performed in polynomial time.

Since all values required to compute each term of the polynomial can be computed in polynomial time the Kruskal-Katona bounds on the two-terminal reliability of a given network can be computed in polynomial time.

3.4. Implementation Details

In general the implemented algorithms are not concerned with efficiency, as long as

they are computable in polynomial time. That is, the actual algorithms used are not necessarily the most efficient ones available, nor are they necessarily implemented in the most efficient manner possible. Specific implementations are often chosen for understandability or ease of programming.

The algorithms presented here are presented for directed graphs since it is usually easier to transform an algorithm on directed graphs to apply to undirected graphs rather than visa-versa. However, the methods presented in this thesis are all discussed in reference to undirected graphs, while conversion of these methods to apply to directed graphs would be fairly straightforward.

3.4.1. Maxflow Algorithm

The Ford and Fulkerson *maxflow labelling algorithm* presented here is taken from Lawler [31]. Let c_{ij} be the capacity on an arc $<i,j>$. Let x_{ij} be the amount of flow through the arc $<i,j>$. If $c_{ij} = 1$ for all arcs $<i,j>$ and $c_{ij} = \infty$ if there is no arc $<i,j>$, the algorithm produces a minimum cardinality (s,t)-cut.

- label$(s) = (-,\infty)$

/* labelling and scanning */

- repeat until all labelled nodes have been scanned

    - repeat until *t* is labelled

        - find a labelled but unscanned node *i*

        - scan node *i* as follows
        - for each arc$<i,j>$
            if $x_{ij} < c_{ij}$ and *j* is unlabelled

$\delta_j = \min\{c_{ij} - x_{ij}, \delta_i\}$
label$(j) = (i^+, \delta_j)$

- for each arc$< j, i >$
  if $x_{ij} > 0$ and j is unlabelled
  $\delta_j = \min\{x_{ij}, \delta_j\}$
  label$(j) = (i^-, \delta_j)$

- end repeat

/* augmentation */

- Starting at node t, use the labels to construct an augmenting path. The label on node t points to the second last node in the path and so on. The flow is augmented by increasing or decreasing the arc flows by δ_i. The label i^+ indicates the flow is to be increased and i^- indicates that the flow is to be decreased.

- unlabel all of the nodes

- end repeat

- a maximal flow has been found
- a cutset of minimum capacity is obtained by placing all labelled nodes in a set S and all unlabelled nodes in a set T. Any edge from a node in set S to a node in set T is in a minimum capacity cutset.

3.4.2. Shortest (s,t)-path Algorithm

If the costs of each edge are assumed to be equal, a breadth-first search technique can be used to find shortest (s,t)-paths (see [1]). The more general algorithm presented here was developed by Dijkstra [12] for networks with positive arc lengths. It is shown here in a form presented by Lawler [31].

Let u_i represent the length of a path from the origin (u_1) to the node i. If the node i is labelled permanently (that is, $i \in P$) then u_i represents the ''true'' shortest distance from the origin. If the node i is labelled temporarily (that is, $i \in T$) then u_i represents a tentative

labelling of that nodes distance from the origin. The vector *path* is used to create a sequence of nodes corresponding to a shortest path. The length of an arc $<i, j>$ is denoted a_{ij}. Note that $a_{ij} = \infty$, if there does not exist an arc $<i, j>$.

/* initialize distances and path vector */

- $u_1 = 0$
- for $(j = 2,3,...,n)$
  $u_j = a_{ij}$
  $path(j) = 1$ /* path will begin at source node */

- $P = 1$
- $T = 2,3,...,n$
- while $(T \neq \varnothing)$

  /* designate permanent labels */

  - find $k \in T$ where $u_k = \min_{j \in T} u_j$
  - $T = T - k$
  - $P = P + k$

  /* revise tentative labels */

  - for all $j \in T$
    - if $(u_k + a_{kj} < u_j)$
      $path(j) = k$

    - $u_j = \min\{u_j, u_k + a_{kj}\}$
  - end of for loop

- end while

/* get the shortest path from the path vector */
/* nodes can be obtained in reverse order and */
/* a list of nodes in the path made as follows */

- set *current_node* to *t*

- while (*current_node* $\neq s$)
  - insert *current_node* at the front of the path

- $current\_node = path(current\_node)$
- end while
- insert s at the front of the path

3.4.3. Shortest (s,t)-path Counting Algorithm

The algorithm used to count the number of minimum cardinality (s,t)-paths was developed by Ball and Provan [4]. The algorithm makes use of the following notation. For any arc a, let $tail(a)$ be the vertex from which arc a points and let $inarc(v) = \{a \mid a = <w,v> \text{ is } an\ arc\ \}$. *The variable* $npaths(i)$ *represents the number of minimum cardinality paths from s to i.*

- perform a breadth-first search on the digraph in order to produce:
  - a breadth-first labelling, using $\{1,2,...,n\}$ as node labels.
  - for each node x, the distance from s to x, $d(s,x)$.
- delete all arcs $<x,y>$ for which $d(s,x) \geq d(s,y)$.

/* the remaining digraph is a ‘‘breadth-first acyclic digraph’’ */

- for all $v \in V - s$
  $npaths(v) = 0$

- $npaths(s) = 1$

- for $(i = 1,2,...,n-1)$
  $$npaths(i) = \sum_{a \in inarc(i)} npaths(tail(a))$$

- return $(npaths(t))$ /* number of (s,t)-paths */

3.4.4. K-canonical Representation Algorithm

The algorithm *k-canon* described below is an adaptation of Ball and Provan's *k/dcalc* algorithm [4]. The algorithm computes the *k*-canonical representation of the positive integer *num* and stores the result in the vector *M*. The parameter *x* is basically a starting point for the algorithm. In order to find the *d*-canonical representation of F_d, the algorithm is called with $k = d$, $x = b+1$ and $num = F_d$. In order to find the *c*-canonical representation of $\overline{F}_c$, the algorithm is called with $k = c$, $x = b+c-2$ and $num = \overline{F}_c$.

$k-canon\,(k,x,num,M)$

- $i = k$
- while $num > 0$
  - if $\binom{x}{i} - \binom{x-k}{i-k} > num$
    - $x = x - 1$
  - else
    - $M(i) = x$
    - $num = num - \left(\binom{x}{i} - \binom{x-k}{i-k}\right)$
    - $x = x - 1$
    - $i = i - 1$
  - end else
- end while

Harms [22] points out a small problem discovered during the implementation of the algorithm. During the first pass through the algorithm the computation is, $\binom{x}{c} - \binom{x}{0}$. If $\binom{x}{0} = 1$ the result obtained is the *c*-canonical representation of $num+1$. Harms [22] also mentions that this can be fixed quite simply by initializing *num* to be one less than the

value desired.

Ball and Provan [4] point out that the successive recalculation of $\binom{x}{i} - \binom{x-k}{i-k}$ can be performed using four arithmetic steps by making use of the following identities:

$$\binom{j-1}{i} = \frac{j-m}{j}\binom{j}{i} \qquad \text{for} \quad j > m \geq 0$$

$$\binom{j-1}{i-1} = \frac{m}{j}\binom{j}{i} \qquad \text{for} \quad j \geq m > 0$$

3.4.5. Lower Pseudopower Algorithm

The lower pseudopower of a non-negative integer *m* can be computed using the method described earlier. The implementation details are taken directly from the definitions except that the identities used in *k-canon* are also used to determine successive lower pseudopowers (that is determining $z^{i+1/k}$ from $z^{i/k}$) by keeping track of the previous value for each of the $\binom{m_i}{i}$ calculations.

3.4.6. Other Implementation Considerations

All algorithms have been implemented in the language ‘‘C’’, on a VAX 11/780. Reliability values were computed using double precision arithmetic (sixty-four bits). Values are usually reported using enough digits of precision to demonstrate significant differences in the bounds being examined.

3.5. Improvements on the Kruskal-Katona Bounds

One of the main reasons for investigating the Kruskal-Katona bounds is that they can be improved upon quite substantially in the two-terminal case.

The reliability polynomial expands as:

$$R = F_0 p^{b-0} q^0 + F_1 p^{b-1} q^1 + \cdots + F_{c-1} p^{b-(c-1)} q^{c-1} + \mathbf{F}_c p^{b-c} q^c + \mathbf{F}_{c+1} p^{b-(c+1)} q^{c+1} + \cdots + F_{d-1} p^{b-(d-1)} q^{d-1} + F_d p^{b-d} q^d$$

where the terms which are not shown in bold type are the terms for which exact, efficient, algorithms are known. The bounds may be improved by computing exact values for some of the terms shown in bold type rather than estimating them.

Since computing n_c, the number of minimum (s,t)-cuts is #P-complete [40], computing F_c is also #P-complete. It is therefore expected that no easy improvement can be made on the bound from the F_c term upward. Thus, the possibility of finding values for F_{d-1}, F_{d-2},... is examined. Note that F_{d-1} is the number of pathsets which contain $l+1$ edges. If the number of pathsets of size $(l+1)$ can be counted in polynomial time, F_{d-1} is known exactly and the bounds might be improved.

3.5.1. Counting Pathsets of Size (l+1)

Using a method similar to that used by Ball and Provan [4] to find the number of (s,t)-paths containing l edges, it is possible to compute the number of (s,t)-paths containing $(l+1)$ edges.

3.5.1.1. Algorithm Countwalks

This algorithm iteratively computes the number of walks of the current length, from the source node to the node in question. Let $nwalks(vertex, length)$ denote the number of walks of the specified length from the source node to the specified vertex.

- for $(i = 1,2,...,n)$
  $nwalks(i, 0) = 0$

- $nwalks(s, 0) = 1$

- for $(i = 1,2,...,l+k)$
  - for each vertex v with neighbours $N(v)$
    $$nwalks(v, i+1) = \sum_{w \in N(v)} nwalks(w, i)$$

/* $nwalks(t, l)$ = number of walks of length l from s to t */
/* $nwalks(t, l+k)$ = number of walks of length $l+k$ from s to t */

Since walks of length l and $(l+1)$ are necessarily paths, the algorithm *countwalks* can be used to compute the number of (s,t)-paths consisting of l and $(l+1)$ edges, where l is the number of edges in a shortest (s,t)-path. Let *numpaths(l)* denote the number of paths of length l. The number of pathsets consisting of $(l+1)$ edges, F_{d-1}, can be computed using *numpaths(l)* and *numpaths(l+1)* in the following manner:

$$F_{d-1} = numpaths(l) \times (b-l) + numpaths(l+1).$$

This yields the following new bounding polynomials:

$$R \le \sum_{i=0}^{c-1} \binom{b}{i} p^{b-i} q^i + \sum_{i=c}^{d-2} \bar{F}_c^{\,i/c} p^{b-i} q^i + F_{d-1} p^{b-(d-1)} q^{d-1} + F_d p^{b-d} q^d$$

$$R \ge \sum_{i=0}^{c-1} \binom{b}{i} p^{b-i} q^i + \sum_{i=c}^{d-1} F_{d-1}^{\,i/d-1} p^{b-i} q^i + F_d p^{b-d} q^d$$

These bounds are referred to as KK1, since they revise the basic Kruskal-Katona bounds by making use of pathsets of size $(l+1)$.

3.5.2. Counting Pathsets of Size (l+2)

The problem of counting pathsets of size $(l+2)$ is not as simple as counting pathsets of size $(l+1)$. Complications arise because walks of length $(l+k)$ are not necessarily paths of length $(l+k)$ for $k \ge 2$. In the case $k = 2$, an overcounting occurs in the algorithm because each edge can be traversed in both directions. For example, using the graph shown in Figure 3.1, the algorithm counts the following three walks of length $l+2 = 4$.

1) (s,x,y,x,t)
2) (s,x,s,x,t)
3) (s,x,t,x,t)

An algorithm has been designed and implemented to count the number of paths of length $l+2$, by first using the algorithm *countwalks* to compute the number of walks and then subtracting values for the two types of overcounting that can occur. These two types of overcounting are:

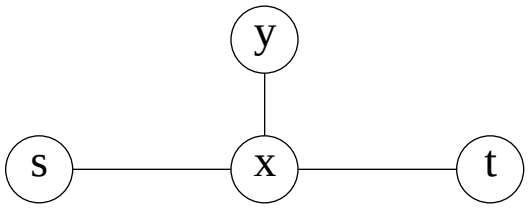


Figure 3.1

(i) overcounting as a result of an edge which is attached in some way to an (s,t)-path of length l. This is demonstrated in Figure 3.1 by the path (s,x,y,x,t). Let $shortpaths(i,j)$ denote the number of shortest (i,j)-paths and let $length(i,j)$ be the length of a shortest (i,j)-path. The value can be counted using the following algorithm.

- for each vertex $x \in V$
  - if $length(s,x) + length(x,t) = length(s,t)$
    - if ($x = s$ or $x = t$)
      - count = count + $shortpaths(s,x) \times (degree(x) - 1) \times shortpaths(y,t)$
    - else count = count + $shortpaths(s,x) \times (degree(x) - 2) \times shortpaths(y,t)$
- end for
- return (count)
- end

(ii) overcounting an edge which is actually part of an (s,t)-path of length l. This is demonstrated by the paths (s,x,s,x,t) and (s,x,t,x,t) in Figure 3.1. The overcounting will be $numpaths(l) \times l$ since each path has l edges and these l edges will be overcounted once for each shortest (s,t)-path.

Let $numwalks(l+2)$ represents the number of walks of length $(l+2)$ and $overcount(i)$ and $overcount(ii)$ denote the number of times configurations of type (i) and (ii) are over-counted. A count of the number of paths of length $(l+2)$ can be obtained using the following:

$$numpaths(l+2) = numwalks(l+2) - overcount(i) - overcount(ii)$$

Once the number of paths of length $(l+2)$ is known, the number of pathsets of size $(l+2)$ is computed. Again an overcounting is obtained, and the number of configurations that have been overcounted are then subtracted from this value.

$$\text{overcount} = numpaths(l) \times \binom{b-l}{2} + numpaths(l+1) \times (b-(l+1)) + numpaths(l+2)$$

This equation results in two types of overcounting: configurations of type A, as shown in Figure 3.2, and those of type B, as shown in Figure 3.3.

Configuration A:

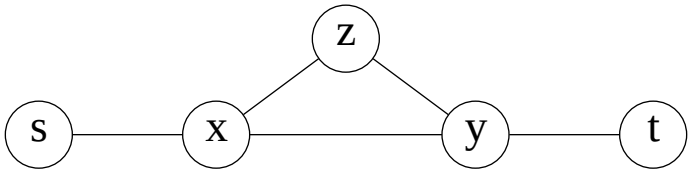


Figure 3.2

Configurations of type A are counted twice as pathsets of size $(l+2)$. The edges (x,z) and (z,y) along with the path (s,x,y,t), form a pathset of size $(l+2)$. However, the same edges are also counted when considering the path (s,x,z,y,t) along with the edge (x,y).

Configurations of type B are also counted twice. The path (s,x,v,y,t) and the edges (x,u), (y,u) form a pathset of size $(l+2)$ and the path (s,x,u,y,t) along with the edges (x,v),

Configuration B

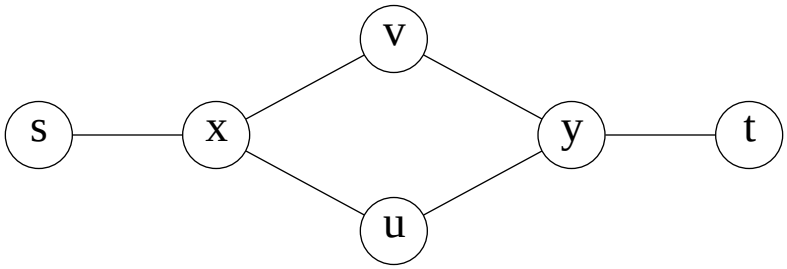


Figure 3.3

(y,v), form the same pathset of size $(l+2)$. Therefore configurations of type B must also be counted and subtracted from the overcounting.

Configurations of type A are counted by locating nodes x and y such that $length(s,x) + length(x,y) + length(y,t)=l$. Since the length of the shortest path between x and y must be one and the number of paths that are one longer than the shortest path can be counted using the algorithm *countwalks*, the number of paths from x to y of length two are counted using this same algorithm. This gives a count of the number of triangles whose base (the edge (x,y)) is an edge in a minimum (s,t)-path. Each of these triangles is counted once for every minimum path that contains the edge (x,y). Therefore, the count of paths which contain a triangle with the base (x,y) is:

$$count\_A = minpaths(s,x) \times next\_to\_minpaths(x,y) \times minpaths(y,t)$$

where $minpaths(i,j)$ denotes the number of minimum (i,j)-paths and $next\_to\_minpaths(i,j)$ denotes the number of paths from i to j which contain one more edge than the minimum (i,j)-path. This value is counted for each edge (x,y) along all minimum (s,t)-paths to give the number of configurations of type A. For example Figure

3.4 contains twelve configurations of type A.

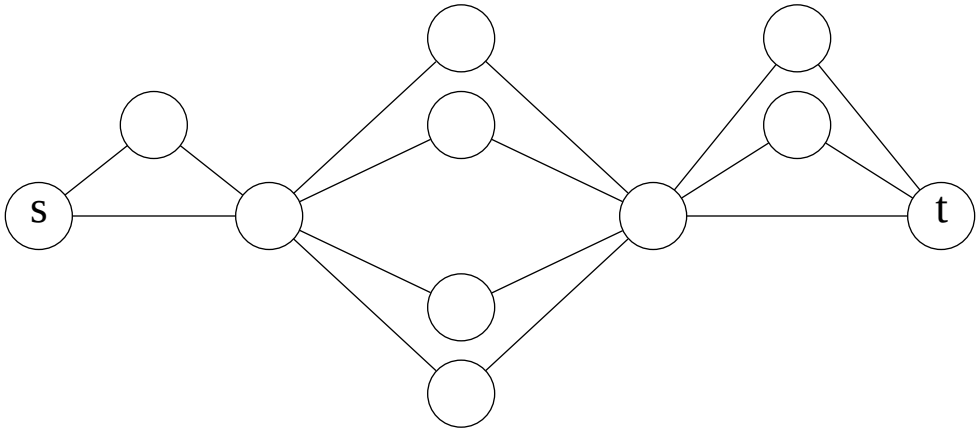


Figure 3.4

The number of configurations of type B can be counted in a similar fashion. The differences are that $length(s,x) + length(x,y) + length(x,t) = l$ and $length(x,y) = 2$, before the nodes x and y are considered to be part of a configuration of type B. Also the number of paths between the nodes x and y are chosen in groups of two, giving a count of the number of minimum (s,t)-paths that contain configurations of type B as:

$$count\_B = minpaths(s,x) \times \left\{ \begin{matrix} minpaths(x,y) \\ 2 \end{matrix} \right\} \times minpaths(y,t)$$

This is done for all pairs of nodes x, y resulting in a count of the number of configurations of type B. For example, Figure 3.4 contains six such configurations.

Therefore a final value for the actual number of pathsets of size $(l+2)$ is:

$$\begin{aligned} F_{d-2} \;=\; & numpaths(l) \times \left\{ \begin{matrix} b-l \\ 2 \end{matrix} \right\} + numpaths(l+1) \times (b-(l+1)) \\ & + numpaths(l+2) - count\_A - count\_B \end{aligned}$$

The new bounds which make use of the number of pathsets of size $(l+2)$ are:

$$R \le \sum_{i=0}^{c-1} \binom{b}{i} p^{b-i} q^i + \sum_{i=c}^{d-3} \bar{F}_c^{\,i/c} p^{b-i} q^i + F_{d-2} p^{b-(d-2)} q^{d-2} + F_{d-1} p^{b-(d-1)} q^{d-1} + F_d p^{b-d} q^d$$

$$R \ge \sum_{i=0}^{c-1} \binom{b}{i} p^{b-i} q^i + \sum_{i=c}^{d-2} F_{d-2}^{\,i/d-2} p^{b-i} q^i + F_{d-1} p^{b-(d-1)} q^{d-1} + F_d p^{b-d} q^d$$

These bounds are referred to as the KK2 bounds, since they revise the KK1 bounds by using the number of pathsets of size $(l+2)$.

3.5.3. Counting Pathsets of Size (l+k)

This technique of computing the number of (s,t)-paths of length $(l+k)$ to obtain an overcount of the number of pathsets of size $(l+k)$ and then subtracting the number of different types of configurations that have been overcounted, could, in theory, be applied to compute the number of pathsets of size $(l+k)$ for any fixed value of k. The number of pathsets of size $(l+k)$ correspond to values for the terms F_{d-k}, so theoretically the bounds can be made arbitrarily accurate. Unfortunately, one of the main problems is that for larger values of k, determining the number of actual (s,t)-paths of length $(l+k)$, as well as being able to identify the many different types of configurations that are overcounted becomes a difficult task. The cause of this difficulty is that the number of configurations that are overcounted grow quite rapidly with k. One approach is to generate the types of configurations that are overcounted and identify how many times the overcounting occurs.

This could be done using ‘‘brute-force’’ techniques, since it would only have to be done once for any value of *k*. Therefore, even an exponential method of enumerating these configurations could be acceptable.

3.5.4. Improving the Bounds for Planar Networks

Ball and Provan [4] point out that by using the (s,t)-dual of a planar graph the number of minimum cardinality (s,t)-cuts, n_c, and therefore the value F_c can be computed efficiently. Shortest (s,t)-paths in an (s,t)-dual represent minimum cardinality (s,t)-cuts in the original network. In general, paths of length $(l+k)$ in the (s,t)-dual represent cutsets of size $(c+k)$ in the original network. Therefore, for planar networks the same methods used to count pathset of size $(l+k)$ can be applied to count cutsets of size $(c+k)$. This should dramatically improve the bounds for planar networks since they can be improved substantially from both ends of the reliability polynomial.

3.6. Comparison of Methods and Computational Results

The computation of the number of pathsets of size $(l+1)$ and $(l+2)$ has been described to demonstrate the practicality of these methods for improving the basic Kruskal-Katona bounds. A comparison of some results obtained using each of the methods KK0, KK1, and KK2 is presented to illustrate the improvements gained by using these improved methods.

Table 3.1 illustrates the coefficients computed for the 10 node ladder depicted in the appendix. They were computed using the KK0, KK1, and KK2 methods and a technique for computing exact reliability of series-parallel networks developed in [47]. The

Coefficients for 10 node ladder					
F_i	KK0 lower	KK1 lower	KK2 lower	exact	KK2 upper
F_0	1	1	1	1	1
F_1	17	17	17	17	17
F_2	77	88	90	134	135
F_3	285	340	362	643	665
F_4	714	879	989	2073	2275
F_5	1286	1616	1945	4671	5733
F_6	1708	2177	2829	7403	11011
F_7	1688	2177	3073	8078	16445
F_8	1231	1612	2484	5756	19305
F_9	645	870	1469	2458	17875
F_{10}	230	331	613	613	613
F_{11}	50	84	84	84	84
F_{12}	5	5	5	5	5

Table 3.1

coefficients produced for the upper bound by the KK2 method are included for interest and comparison's sake, since they produce an absolute upper bound on the reliability of the network. Table 3.1 demonstrates how knowing the exact value for an extra coefficient can dramatically affect the bounds. Table 3.2 depicts the reliability values obtained by using the coefficients of Table 3.1. The increase in accuracy of the coefficients is reflected by the tightening of the bounds. This example is not meant to demonstrate typical behaviour of the bounds but rather to demonstrate the potential power that exists for using this method to improve the bounds.

Reliability Results for 10 Node Ladder				
p	KK0	KK1	KK2	Exact
0.10	0.0000409510	0.0000562040	0.0000687793	0.0000742847
0.20	0.0010757116	0.0015250012	0.0022165203	0.0029734859
0.30	0.0067385833	0.0092529739	0.0141760289	0.0241842722
0.40	0.0236085387	0.0312412637	0.0462748820	0.0955346843
0.50	0.0605545044	0.0777969360	0.1065139771	0.2436523438
0.60	0.1285654555	0.1607078562	0.2007626807	0.4586913633
0.70	0.2428156278	0.2927375241	0.3353783955	0.6890327298
0.80	0.4301397466	0.4901456259	0.5227844138	0.8721058722
0.90	0.7279011487	0.7684233106	0.7815387689	0.9732904939
0.91	0.7631727458	0.7994351962	0.8105402996	0.9788740513
0.92	0.7986710793	0.8303271364	0.8394946226	0.9837181988
0.93	0.8339442950	0.8607237497	0.8680534036	0.9878540988
0.94	0.8684065014	0.8901464966	0.8957672824	0.9913151900
0.95	0.9013089334	0.9179917230	0.9220639998	0.9941369051
0.96	0.9317059579	0.9435047703	0.9462225923	0.9963563916
0.97	0.9584151291	0.9657495444	0.9673430499	0.9980122394
0.98	0.9799703949	0.9835728568	0.9843107484	0.9991442158
0.99	0.9945674374	0.9955627602	0.9957548793	0.9997930117

Table 3.2

Results for example flow graph 1			
p	KK0	KK1	KK2
0.10	0.0010000000	0.0012439000	0.0012746422
0.20	0.0080000000	0.0111232000	0.0116067613
0.30	0.0270000000	0.0394173000	0.0411333050
0.40	0.0640000001	0.0941056000	0.0972685666
0.50	0.1250000491	0.1796875346	0.1835327493
0.60	0.2160088640	0.2968766472	0.3001808084
0.70	0.3435463935	0.4435062396	0.4454494096
0.80	0.5248603153	0.6226064723	0.6232596781
0.90	0.8223618631	0.8668610716	0.8669223894
0.91	0.8566941176	0.8930703977	0.8931114770
0.92	0.8895211886	0.9179065502	0.9179325107
0.93	0.9197074617	0.9405666327	0.9405818560
0.94	0.9460669318	0.9602201001	0.9602281853
0.95	0.9675241929	0.9761248654	0.9761286109
0.96	0.9833401889	0.9877889781	0.9877903989
0.97	0.9933802170	0.9951612370	0.9951616289
0.98	0.9983539372	0.9987998061	0.9987998663
0.99	0.9998702229	0.9999055904	0.9999055926

Table 3.3

Results for K_5			
p	KK0	KK1	KK2
0.10	0.1000026982	0.1243905913	0.1291971799
0.20	0.2002510848	0.2781340672	0.2974278656
0.30	0.3030036258	0.4386022617	0.4688246091
0.40	0.4150208512	0.5912270848	0.6212859904
0.50	0.5449218750	0.7275390625	0.7490234375
0.60	0.6927148032	0.8416601088	0.8528077824
0.70	0.8388493498	0.9273068497	0.9311909131
0.80	0.9476395008	0.9788526592	0.9795670016
0.90	0.9947027862	0.9980410473	0.9980712279
0.91	0.9963569675	0.9986630906	0.9986814343
0.92	0.9976165447	0.9991318573	0.9991423331
0.93	0.9985361360	0.9994706888	0.9994762174
0.94	0.9991722835	0.9997028351	0.9997054666
0.95	0.9995819480	0.9998509509	0.9998520387
0.96	0.9998207048	0.9999365068	0.9999368734
0.97	0.9999406123	0.9999791076	0.9999791971
0.98	0.9999877223	0.9999957084	0.9999957206
0.99	0.9999991971	0.9999997211	0.9999997215

Table 3.4

A study by Frank and Chou [19] used the assumption of equivalent and statistically independent link failures to study the Arpanet. They discovered a typical value for p=0.98. As a result the bounds here are calculated with a greater concentration of values of p between 0.90 and 0.99. Note also that practical networks such as the Arpanet are typically very sparse. The size of a minimum cardinality (s,t)-cut is usually only two or three. For this reason the majority of the test cases presented here are relatively sparse. The results shown in the tables which follow were computed using the network stated in the title of the table. Diagrams of these networks can be found in the appendix.

Arpanet 1979 s = ISI22 t = CCA	
p	KK0 = KK1 = KK2
0.10	0.0000010000
0.20	0.0000640000
0.30	0.0007290000
0.40	0.0040960000
0.50	0.0156250000
0.60	0.0466560000
0.70	0.1176490008
0.80	0.2621475296
0.90	0.5346365516
0.91	0.5734980575
0.92	0.6159781768
0.93	0.6628758466
0.94	0.7150016592
0.95	0.7727654047
0.96	0.8352877904
0.97	0.8988771008
0.98	0.9552067419
0.99	0.9914138687

Table 3.5

Table 3.3 demonstrates the typical improvement gained by using the different subgraph counting methods. The graph used is the example flow graph 1 shown in the appendix. In this case the KK1 bound dramatically improves on the KK0 bounds while the KK2 bound only slightly improves on the KK1 bound.

Table 3.4 shows the results obtained for the complete graph on five vertices. Again there is a substantial improvement in the KK1 bound over the KK0 bound while the improvement in the KK2 bound over the KK1 bound is not as dramatic.

Results for 7×7grid			
p	KK0	KK1	KK2
0.10	0.0000000001	0.0000000001	0.0000000002
0.20	0.0000000979	0.0000001188	0.0000002145
0.30	0.0000058649	0.0000071449	0.0000119043
0.40	0.0001047754	0.0001300234	0.0001972150
0.50	0.0009765029	0.0012207031	0.0016782284
0.60	0.0060466014	0.0074978058	0.0093706539
0.70	0.0282475235	0.0341795051	0.0391953831
0.80	0.1073741999	0.1245540673	0.1331096404
0.90	0.3488357172	0.3802008500	0.3871709320
0.91	0.3897708376	0.4216281521	0.4279343938
0.92	0.4351706648	0.4670626819	0.4726214307
0.93	0.4856635613	0.5170013044	0.5217435175
0.94	0.5421209522	0.5721458541	0.5760234641
0.95	0.6057775408	0.6335095520	0.6365030496
0.96	0.6782813259	0.7024601573	0.7045875537
0.97	0.7612905280	0.7803552887	0.7816826342
0.98	0.8543882458	0.8666635386	0.8673171943
0.99	0.9477608984	0.9523719676	0.9525528440

Table 3.6

Table 3.5 contains the results of computations done using the 1979 representation of the Arpanet taken from [4] (see the appendix for the configuration used here). The two sites chosen are ISI22 and CCA since they are on the west and east coast respectively and have an (s,t)-cut of size three. It is interesting to note that for the Arpanet no improvement is gained by knowing the exact value of the extra coefficients. The reason in this case being that the two coefficients F_{d-1} and F_{d-2} are estimated correctly by the KK0 method.

The results for the 7×7 grid graph are shown in Table 3.6. Again an improvement is gained using each of the methods KK1 and KK2.

In general the subgraph counting bounds perform well when the size of the minimum cardinality (s,t)-cut is relatively large in comparison with the number of edges. This is because the first term of the bounded polynomial which can be computed exactly predominates. It also means that fewer terms of the reliability polynomial are estimated. As well, the performance is quite good when the number of shortest (s,t)-paths is high.

The method for improving the Kruskal-Katona bounds has been shown to be of practical use since the improvements are quite substantial in most cases. The amount of improvement that each method produces depends upon how well the original Kruskal-Katona method estimates the unknown coefficients.

Chapter 4

Edge-Disjoint Path Bounds

4.1. Development

Harms [22] reports on some bounds for all-terminal reliability developed in the Russian literature [38], which use an approach quite different from that used to obtain bounds by using subgraph counting techniques. A lower bound on the reliability of a graph can be obtained by observing that the probability of a graph failing is no larger than the product of the failure probabilities of each member of any set of edge-disjoint operational subgraphs. This follows from the assumption of statistical independence of edge failures.

Polesskii [37] shows that the maximum number of edge-disjoint spanning trees in a graph is at least $\left\lfloor \frac{c}{2} \right\rfloor$, where c is the size of a minimum cardinality cutset. This also follows from a stronger result developed independently by Nash-Williams [36] and Tutte [44]. Polesskii [38] uses the fact that each of the edge-disjoint spanning trees must fail before the graph fails, to obtain the lower bound on the all-terminal reliability:

$$R \geq 1 - (1 - p^{n-1})^{\left\lfloor \frac{c}{2} \right\rfloor}$$

The only value that must be computed to obtain this bound is c; a polynomial time algorithm was given for this in Chapter 3. This bound can thus be computed quite easily, in polynomial time.

Polesskii's bound can be improved in some instances by enumerating edge-disjoint spanning trees of a graph. This can be accomplished using a result on matroids, developed

by Edmonds [13]. If the number of edge-disjoint spanning trees is computed, it can be used in the Polesskii bound in place of the previous value of $\left\lfloor \frac{c}{2} \right\rfloor$.

4.1.1. The Two-Terminal Bound

Polesskii's idea of using edge-disjoint spanning subgraphs can be easily applied to the two-terminal problem. In the two-terminal case a minimal operational subgraph is simply a path; hence edge-disjoint paths are employed. Ford and Fulkerson's Max-Flow Min-Cut Theorem [18] guarantees that there will be at least *c* edge-disjoint paths between *s* and *t*, which gives the following lower bound on two-terminal reliability:

$$R \geq 1 - \left(\prod_{i=1}^{c} P_i \right)$$

where P_i is the probability that the i^{th} path fails. P_i can be expressed as:

$$P_i = \left(1 - \prod_{j=1}^{L_i} p_j \right)$$

where p_j is the probability that the j^{th} edge in the path is operational and L_i denotes the length of the i^{th} path.

For example a lower bound on the reliability of the graph in Figure 4.1 can be determined by using the two obvious edge-disjoint paths. This gives the following lower bound on the reliability:

$$R \geq 1 - [\,(1 - p_1\, p_2\, p_3)\,(1 - p_4\, p_5)]$$

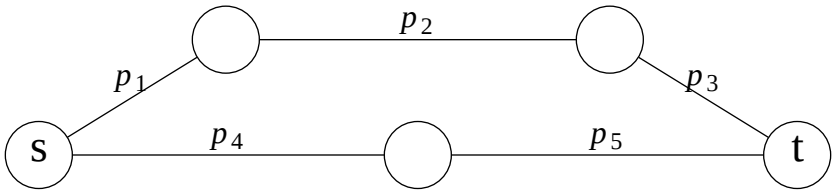


Figure 4.1

4.2. Motivation

The edge-disjoint method of obtaining bounds has one distinct advantage over the family of bounds which uses subgraph counting techniques. The subgraph counting techniques depend on the edges having equal failure probabilities, while the edge-disjoint method uses edge-disjoint operational subgraphs, so the failure probability need not be the same for each edge. The edge-disjoint method is therefore very useful in computing a lower bound on two-terminal reliability when the failure probability of each edge is known and is different.

As stated earlier, the number of edge-disjoint spanning trees in the all-terminal case is typically less than c, the size of a minimum cardinality cutset, and could be as low as $\left\lfloor \frac{c}{2} \right\rfloor$. In the two-terminal case however, the number of edge-disjoint paths between s and t is equal to c, the size of a minimum cardinality (s,t)-cut. Harms [22] shows that for some classes of graphs, the Lomonosov-Polesskii [33] edge-disjoint method produces the best all-terminal lower bounds. Thus, edge-disjoint path methods were developed in the hope that they might produce strong two-terminal lower bounds.

The edge-disjoint path bound makes use of elementary graph theoretic results, making the bound quite easy to understand. It is also simple to compute, as all that is required is one or more edge-disjoint paths between *s* and *t*. In fact, a crude lower bound on two-terminal reliability can be obtained by using any (s,t)-path.

The first order linear programming bounds proposed by Zemel [49] and Assous [2] essentially make use of one (s,t)-path. The second order linear programming bounds [2] are useful for graphs or networks for which edges do not fail independently since they are able to compute a bound on the reliability by using the joint failure probability of every pair of edges. However, they are not much more sophisticated than the first order linear programming bounds and are therefore not discussed in further detail.

Another important reason for investigating the edge-disjoint path bound is that it makes use of different information than the Kruskal-Katona bounds. This makes it interesting to compare the two bounds since, although they use different techniques and make use of different information, they are both computable in polynomial time.

4.3. Implementation Overview

The implementation of the edge-disjoint path bounds is relatively simple in comparison with the Kruskal-Katona bounds. All that is required to compute an edge-disjoint path bound is any algorithm that generates edge-disjoint paths between *s* and *t*. The failure probability of each edge on a path is used to compute the probability that the path fails. The path fails when any single edge of that path fails. The bound on the probability that the two specified nodes *s* and *t* can communicate is one minus the product of the failure

probabilities of each of the chosen edge-disjoint (s,t)-path.

Any algorithm which generates edge-disjoint paths can be used to compute an edge-disjoint path bound. For example a greedy algorithm can be used to obtain a number of edge-disjoint paths. It first finds the most reliable path and marks the edges along that path as ‘‘used’’ so they will not be chosen in the formation of other paths, thus ensuring that the paths are edge-disjoint. The algorithm is greedy in that it finds the most reliable (s,t)-path possible, then using the remaining edges it finds the next most reliable path and so on until no more paths can be found. It is a simple method, and because of its greedy nature it does not guarantee that the maximum number of edge-disjoint paths will be found.

If edges are assumed to fail with varying failure probabilities the most reliable path can be found using an algorithm much like a shortest path algorithm by assuming that the cost of each edge is $(1 - p_i)$, where p_i is the probability that edge i is operational. Instead of summing the costs of each edge they are multiplied, since the failure probability of each edge is multiplied to obtain the probability that the path fails. The object is to find the most reliable path, which means that the probability of the path failing is to be minimized. Alternatively, the cost of each edge could be assigned the value $log\,(1 - p_i)$, thus allowing the use of the standard shortest path algorithm using addition rather than multiplication.

Edges are often assumed to have equal failure probability, especially since failure probabilities for each individual edge are not usually available. Since the subgraph counting bounds are valid only under the assumption that all edges have equal failure probabilities, much of the testing and discussion is done with the assumption that the edges fail with equal probabilities. Under this assumption the most reliable path in a graph is simply the

shortest path.

To ensure that the maximum number of edge-disjoint paths is found a maximum flow algorithm is used. This algorithm finds the maximum flow between the specified nodes by augmenting flows along edges that have already been used if such an augmentation leads to an increase in the (s,t)-flow. If each edge is assumed to have a capacity for carrying one unit of flow this algorithm determines which edges are used in making up the maximum number of edge-disjoint paths. The paths themselves can then be easily obtained and used to produce a lower bound.

Another algorithm that can be used to compute edge-disjoint paths is a minimum-cost flow algorithm. This algorithm finds the maximum flow through the graph subject to the constraint that the overall cost of the edges used is to be minimized. If edges fail with equal probabilities the cost of each edge and its capacity are assumed to be one. The algorithm determines which edges comprise the maximum number of edge-disjoint paths while using the least total number of edges possible. Jewell [25] and Busacker and Gowan [8] have shown that the augmentation by δ, of a minimum-cost flow of value v along a minimum-cost augmenting path yields a minimum-cost flow of value $\delta + v$. The minimum-cost maximum flow algorithm therefore uses a shortest path computation to find minimum-cost augmenting paths. The flow is iteratively incremented until a final flow of c is reached. If failure probabilities are different for each edge then the most reliable path is obtained rather than the shortest.

As mentioned earlier, any method for producing edge-disjoint paths can be used to obtain a lower bound. Unfortunately, there is no known method for determining the most

reliable set of edge-disjoint paths. As a result, heuristic approaches are considered. Besides the mincost, maxflow and greedy methods investigated here other possibilities exist. Of particular interest is the Edmonds and Karp method for finding maximum flows [14]. Their method is similar to the Ford and Fulkerson *maxflow labelling algorithm* described in Chapter 3 except that during the labelling process nodes are scanned on a ‘‘first-labelled first-scanned’’ basis. That is, before scanning a labelled node *u*, scan those nodes that were labelled before *u*. Thus a maximum flow is obtained by augmenting the flow along an augmenting path containing the fewest arcs, until no more augmenting paths exist.

Of the methods mentioned the Ford and Fulkerson *maxflow labelling algorithm*, the mincost algorithm and the greedy algorithms have been implemented. The implementation of the maxflow algorithm is essentially the same as shown in Chapter 3. The only difference is that the paths are actually produced.

4.4. Implementation Details

Because edge failures are independent the edge-disjoint bounds can make use of one very simple observation to improve the bounds obtained by the ‘‘strict’’ mincost and maxflow algorithms. If the graph has one or more cutpoints it can be divided into biconnected components. The two-terminal reliability of the entire graph is equal to the product of the reliability of each component containing a cutpoint that is a node along a shortest (s,t)-path. For example, the reliability of the graph in Figure 4.2 can be computed, by first computing a bound on the probability that the nodes *s* and *x* can communicate (the probability that at least one of the paths in the first component is operational). Call this bound *C*.

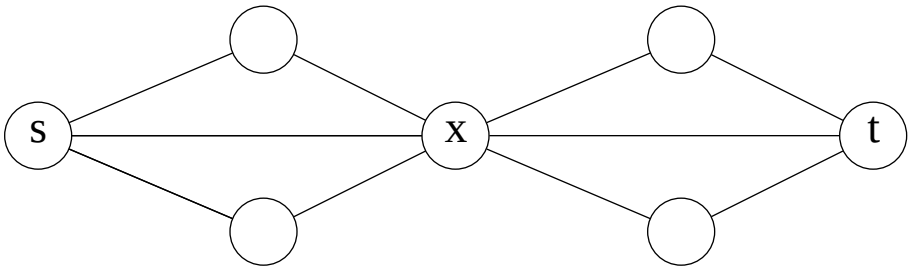


Figure 4.2

Then compute a bound on the probability that nodes *x* and *t* can communicate (the probability that a path in the second component is operational), and call this bound *D*. A bound on the probability that the two nodes *s* and *t* can communicate is $C \times D$.

An algorithm for finding biconnected components of a graph can be found in [43].

4.4.1. Mincost Algorithm

The minimum-cost flow algorithm presented here determines a minimum-cost flow for a given network flow value *v*. Most algorithms for solving minimal cost flows are based on linear programming optimality conditions. Such approaches are discussed by Ford and Fulkerson [18].

The algorithm presented here is the Edmonds and Karp minimum-cost maximum flow algorithm [14] taken from Lawler [31]. If the cost of each edge is one and $v = c$, the algorithm determines which edges make up a maximum flow using the minimum total number of edges. This is desirable, since the more edges a path consists of, the more susceptible that path is to failure.

The following new costs are used to compute an augmenting path using a shortest path computation. For a given flow $x = (x_{ij})$ and arc costs a_{ij}, let

$$\bar{a}_{ij} = \begin{cases} a_{ij}, & \text{if } x_{ij} < c_{ij},\ x_{ji} = 0. \\ \min(a_{ij}, -a_{ji}), & \text{if } x_{ij} < c_{ij},\ x_{ji} > 0. \\ -a_{ji}, & \text{if } x_{ij} = c_{ij},\ x_{ji} > 0. \\ +\infty, & \text{if } x_{ij} = c_{ij},\ x_{ji} = 0. \end{cases}$$

Note also that $a_{ij} = +\infty$ if (i, j) is not an arc of the network.

- repeat until no more flow-augmenting paths exist

    - apply a shortest path algorithm with respect to (nonnegative) arc lengths $\bar{a}_{ij}^{(k)}$ to find the shortest (s,t)-path, where $\bar{a}_{ij}^{(k)} = \bar{a}_{ij} + \pi_i^{(k)} - \pi_j^{(k)}$

    - augment the flow by δ, where $v' + \delta \leq v$, along a minimum cost (s,t)-path as found using the shortest path computation

    - let $u_j^{(k)}$ denote the length of a shortest path from s to j with respect to arc lengths $\bar{a}_{ij}^{(k)}$, compute $\pi_j^{(k+1)} = \pi_j^{(k)} + u_j^{(k)}$ where $\pi_j^{(0)} = 0$ and $u_j^{(k)} = \pi_j^{(k)} = +\infty$ if node j is inaccessible from s

- end of for loop

Edmonds and Karp [14] have shown that for networks with arc costs that are initially positive, each cost $\bar{a}_{ij}^{(k)}$ is nonnegative. Thus, Dijkstra's algorithm for finding shortest paths in networks with positive arc costs, presented in Chapter 3, can be used to perform the shortest path computations.

4.4.2. How the Algorithms are Used

The maxflow and mincost algorithms determine the edges of the graph that are used in forming the edge-disjoint paths. The actual paths used in computing the bound on the reliability of the graph must be determined from these edges. This has been done by first choosing the shortest path (or the most reliable), then choosing the second shortest (second most reliable) and so on in a greedy fashion until all the paths have been determined. This method will yield the greatest reliability value over the fixed number

of edges if there are exactly two edge-disjoint paths.

Proposition:

Let π_1 and π_2 be edge-disjoint (s,t)-paths and let π_3 and π_4 be two other edge-disjoint (s,t)-paths, such that $E(\pi_1) \cup E(\pi_2) = E(\pi_3) \cup E(\pi_4)$, where $E(\pi_i)$ denotes the set of edges that make up the path π_i. If π_1 is a shortest (s,t)-path, then the lower bound obtained using the paths π_1 and π_2 will be no less than the lower bound obtained using the path π_3 and π_4.

Proof:

Let $a = |E(\pi_2)|, \quad b = |E(\pi_1)|$.

Let $e = |E(\pi_3)|, \quad f = |E(\pi_4)|$.

Since the sum of the lengths of the paths of each set is equal:

$$a + b = e + f.$$

Assume $0 < p < 1$ and $a, e, f > b$; otherwise the bounds are equal.

This is shown diagrammatically in Figure 4.3.

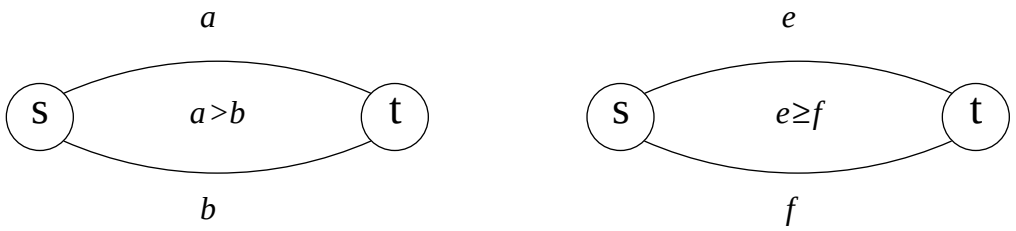


Figure 4.3

Assume to the contrary that the paths π_3 and π_4 produce the best bound. Therefore, using the methods described to compute lower bounds given a set of edge-disjoint paths, yields:

$$\begin{aligned} 1-[(1-p^a)(1-p^b)] &< 1-[(1-p^e)(1-p^f)] \\ 1-[1-p^a-p^b+p^{a+b}] &< 1-[1-p^e-p^f+p^{e+f}] \\ p^a+p^b-p^{a+b} &< p^e+p^f-p^{e+f} \end{aligned}$$

since $a+b=e+f$

$$\begin{aligned} p^a+p^b &< p^e+p^f \\ p^a-p^e &< p^f-p^b \\ p^e(p^{a-e}-1) &< p^b(p^{f-b}-1) \\ p^{e-b}(p^{a-e}-1) &< p^{f-b}-1 \end{aligned}$$ I

Since the length of a shortest path is b, $f-b > 0$ and $p^{f-b}-1$ is negative. Hence,

$$\frac{p^{e-b}(p^{a-e}-1)}{(p^{f-b}-1)} > 1$$

Recall *that* $a+b=e+f$, therefore $a-e=f-b$. Thus *leaving*:

$$p^{e-b} > 1$$

However, since the length of a shortest path is b, $e-b > 0$.

$$\textit{Therefore} \quad p^{e-b} < 1 \quad \text{for } 0 < p < 1.$$

This yields a contradiction. Hence, the original assumption must be false.

Consequently:

$$1-[(1-p^a)(1-p^b)] > 1-[(1-p^e)(1-p^f)].$$

Therefore, the set of paths containing the shortest path produces the best bound.

Unfortunately this method of choosing paths in a greedy fashion does not guarantee that the greatest reliability value is obtained when dealing with more than two paths. For example, consider the paths in Figure 4.4.

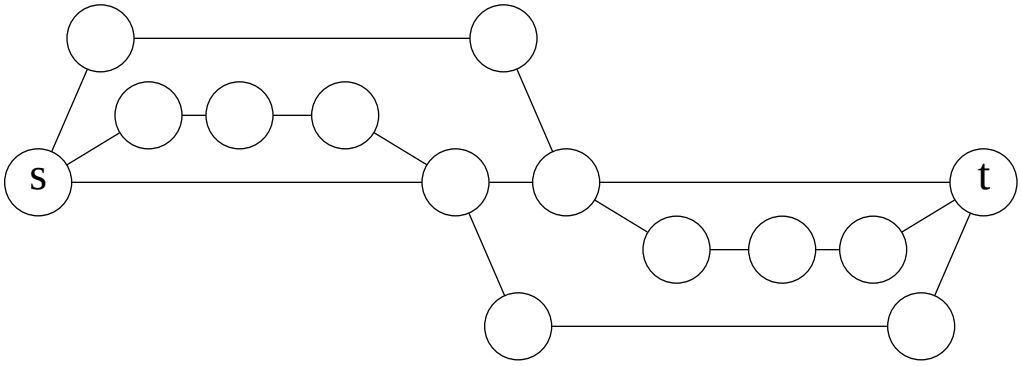


Figure 4.4

The set of three paths that result by greedy selection is a set of paths *P*, containing paths of lengths 3,7 and 7. Another possible set of three paths is *P′*, containing paths of lengths 4,4 and 9. Each set of paths contains 17 edges and neither produces a uniformly better bound than the other. Table 4.1 demonstrates the bounds obtained using these two sets of paths.

Although the greedy method does not guarantee the best reliability value possible it is easy to compute. It also works quite well in practice, since practical networks usually contain a wider range of path lengths.

In an attempt to achieve an as accurate bound as possible, the lower bound is actually computed over all possible numbers of edge-disjoint paths. That is, the bound is computed first using one path, then two, until the maximum number of paths has been

Bounds using different sets of paths		
p	$P = 3,7,7$	$P' = 4,4,9$
0.10	0.0010001998 *	0.0001999910
0.20	0.0080253954 *	0.0031979506
0.30	0.0274255470 *	0.0161537580
0.40	0.0670645754 *	0.0507935370
0.50	0.1386184692 *	0.1228103638
0.60	0.2592796259 *	0.2500387020
0.70	0.4467577310 *	0.4458542098
0.80	0.6952197552	* 0.6982125908
0.90	0.9262409756	* 0.9275520912
0.91	0.9424540526	* 0.9435063045
0.92	0.9567336876	* 0.9575444792
0.93	0.9689628394	* 0.9695568288
0.94	0.9790656551	* 0.9794734759
0.95	0.9870211027	* 0.9872776793
0.96	0.9928791981	* 0.9930216286
0.97	0.9967802107	* 0.9968451911
0.98	0.9989772802	* 0.9989980499
0.99	0.9998629268	* 0.9998657208

Table 4.1

Note: * denotes the better of the two bounds

found. The largest bound over all number of paths is then used. This is important, since in certain types of graphs a better bound results, for some failure probabilities, by using a small number of short paths rather than a larger number of longer paths. For example a lower bound for the graph in Figure 4.5 using one path is:

$$R \geq 1 - (1 - p^3) = p^3 \qquad \text{I}$$

When two paths are used the bound becomes:

$$R \geq 1 - (1 - p^8)^2 = 2p^8 + p^{16}$$

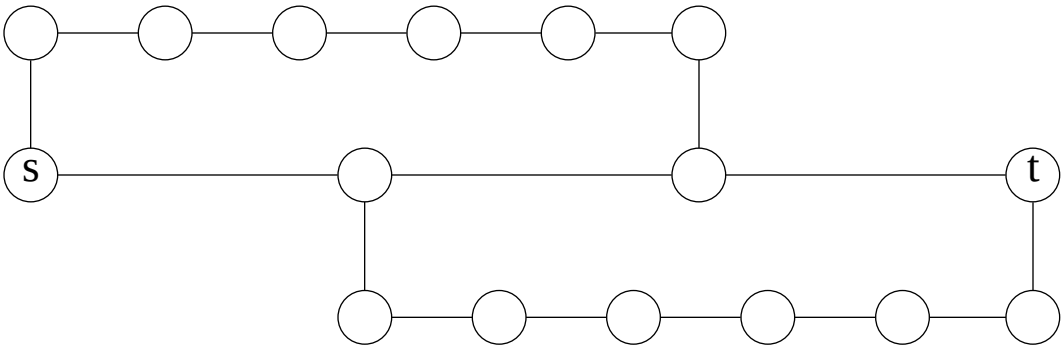


Figure 4.5

Bounds for Graph in Figure 4.5		
p	1 path	2 paths
0.10	0.00100000 *	0.00000011
0.20	0.00800000 *	0.00001536
0.30	0.02700000 *	0.00028430
0.40	0.06400000 *	0.00229269
0.50	0.12500000 *	0.01168823
0.60	0.21600000 *	0.04431957
0.70	0.34300000 *	0.13525475
0.80	0.51200000 *	0.34230299
0.90	0.72900000 *	0.70287298
0.91	0.75357100 *	0.74400537
0.92	0.77868800	* 0.78476807
0.93	0.80435700	* 0.82458182
0.94	0.83058400	* 0.86275473
0.95	0.85737500	* 0.89846650
0.96	0.88473600	* 0.93075068
0.97	0.91267300	* 0.95847502
0.98	0.94119200	* 0.98031945
0.99	0.97029900	* 0.99475169

Table 4.2

Note: * denotes the better of the two bounds

As can be seen in Table 4.2 the bound computed using one path is a better bound than the bound computed using two paths for values of $p < 0.92$. For this reason the best bound obtained over all numbers of paths from 1 to c is retained for each value of p. Since the mincost and maxflow algorithms find c edge-disjoint paths one at a time, the largest value for the bound over all possible numbers of paths can be obtained quite easily.

The bounds referred to as the *maxflow bounds* and the *mincost bounds* are bounds which are computed using the techniques outlined in this chapter. That is, using a greedy selection of paths after the original maxflow or mincost flow technique is used to determine the edges and then obtaining the best bound over all possible numbers of paths.

During preliminary testing, the bounds obtained using the greedy method to produce edge-disjoint paths were generally not as good as the *maxflow* or *mincost* bounds. For this reason the greedy method is not included in the discussion and comparison of results.

For testing and comparison purposes edges are assumed to fail with equal probability. This is especially useful for comparison with the subgraph counting techniques, since they are based on the assumption that all edges fail with equal probability.

4.5. Results

One might expect the mincost method of finding edge-disjoint paths to outperform the maxflow method in all cases, since the mincost method uses the minimum total number of edges such that the flow is maximized. In fact, it appears to be better than the maxflow algorithm for all but extreme cases. In general, edge-disjoint path bounds produce better bounds when more reliable (shorter) paths are used, even at the expense

of making the more unreliable (longer) paths even more unreliable (longer). That is, short paths are favoured, even when it means increasing the total number of edges used. The reason that the maxflow method doesn't normally produce better bounds than the mincost method is that there is no guarantee that the maxflow method will retain the short paths until all c paths have been found.

Since there is no guarantee on the performance of the maxflow algorithm, it could be used in conjunction with the mincost method by using the maximum of the two results. If only one bound is to be computed, however, the mincost algorithm should be used since it yields the better bound in the vast majority of the test cases.

To obtain the best possible bound using edge-disjoint paths requires finding the most reliable combination of edge-disjoint paths of lengths which lead to the greatest reliability values. A related problem, which, if used, would likely improve on the min-cost bound is one that involves finding a maximum number of edge-disjoint paths with length constraints on each path. That is, given a graph G with distinct nodes s and t, and an integer k, find the maximum number of edge-disjoint paths of length $\leq k$. It has been shown that except for values of $k \leq 4$ the problem is NP-complete [23]. It is therefore likely that finding the most reliable combination of paths and path lengths is also NP-complete.

4.5.1. Tabulation of Results

The bounds shown in Table 4.3 for the 1979 version of the Arpanet as depicted in [4], and shown in the appendix, were computed using the nodes ISI22 and CCA as the

source and target nodes respectively. The results illustrate how the mincost method performs as well or better than the maxflow method on an average graph of practical interest.

Results Using Edge Disjoint Path Bounds						
	Arpa			Example flow graph 1		
p	mincost	maxflow	#	mincost	maxflow	#
0.10	0.000001	0.000001	2	0.001001	0.001001	2
0.20	0.000064 *	0.000064	2	0.008063	0.008063	2
0.30	0.000735 *	0.000731	2	0.027709	0.027709	2
0.40	0.004200 *	0.004138	2	0.067834	0.067834	2
0.50	0.016586 *	0.016107	2	0.138672	0.138672	2
0.60	0.052421 *	0.050149	2	0.252578	0.252578	2
0.70	0.142573 *	0.135786	2	0.420295	0.420295	2
0.80	0.341371 *	0.333302	3	0.662010 *	0.639926	2
0.90	0.732100 *	0.717569	3	0.912096 *	0.911417	4
0.91	0.777147 *	0.763311	3	0.930844	* 0.933242	4
0.92	0.820938 *	0.808264	3	0.947559	* 0.952038	4
0.93	0.862411 *	0.851351	4	0.963511	* 0.967585	4
0.94	0.900367 *	0.891307	4	0.977469	* 0.979790	4
0.95	0.933533 *	0.926721	4	0.987559	* 0.988722	4
0.96	0.960673 *	0.956147	4	0.994160	* 0.994645	4
0.97	0.980780 *	0.978306	4	0.997880	* 0.998033	4
0.98	0.993388 *	0.992439	4	0.999519	* 0.999548	4
0.99	0.999038 *	0.998885	4	0.999965	* 0.999967	4

Table 4.3

Note:

* denotes the best of the two methods for that graph
if there is no * in either column the results were equal
# denotes the number of paths used to obtain the bound

Table 4.3 also shows the bounds obtained for the example flow graph 1, shown in the appendix. These results show that the maxflow method can provide a better bound

than the mincost method on certain types of graphs. The reason that the bound is better using the maxflow algorithm is because in this case it retains shorter paths at the expense of increasing the total number of edges used. The mincost method on the other hand maintains the fewest number of edges possible at each stage. The bound is computed for each method over all possible numbers of paths. The number of paths, the path lengths and the total number of paths obtained at each phase of the mincost method are shown in Table 4.4. The same results are shown for the maxflow method in Table 4.5.

Mincost Path Information		
# of paths	path lengths	total
1	3	3
2	3,6	9
3	3,8,8	19
4	7,7,9,9	32

Table 4.4

Maxflow Path Information		
# of paths	path lengths	total
1	3	3
2	3,6	9
3	4,8,8	20
4	4,8,8,15	35

Table 4.5

The bounds show how the lengths of the paths and the number of paths affect the bound obtained with each method. For up to two paths the path lengths of each path and

the total number of edges used are exactly the same for each method. This is reflected in the bounds in the fact that the bounds are the same for $0.10 \leq p \leq 0.70$. Once three paths have been determined the mincost method uses paths of lengths 3,8 and 8 while the maxflow method finds three paths of lengths 4,8 and 8. The difference in the total number of paths used and the difference in the length of the shortest path is reflected in the bounds obtained for $0.80 \leq p \leq 0.90$. For these values of p the mincost method produces better bounds than the maxflow algorithm.

When the algorithm reaches the four path stage the mincost method uses paths of lengths 7,7,9 and 9 for a total of 32 edges. Because the maxflow method is not concerned with minimizing overall path length it chooses paths of lengths 4,8,8 and 15 for a total of 35 edges. The interesting results in this case are reflected in the bounds for values $0.91 \leq p \leq 0.99$. For these values of p the maxflow method now yields a better bound than the mincost method.

Another interesting note is that the maxflow method obtains its best bounds using only two paths for $0.10 \leq p \leq 0.90$ and four paths for $0.91 \leq p \leq 0.99$. The reason that the use of three paths does not produce a better bound for any of the values of p tested, is because three of the four paths used at the four path stage of the algorithm are the same three paths used at the three path stage. Although the fourth path is one of length 15 the results indicate that the extra path helps to obtain a better bound when the probability of an edge being operational is relatively high (in this case, ≥ 0.90).

Table 4.6 demonstrates bounds obtained on a complete graph on five vertices, K_5, and a seven by seven grid graph. These graphs demonstrate examples of graphs for

Results Using Edge Disjoint Path Bounds				
	K_5		7×7 grid	
p	mincost	maxflow	mincost	maxflow
0.10	0.12673090	0.12673090	0.00000000	0.00000000
0.20	0.29221120	0.29221120	0.00000001	0.00000001
0.30	0.47250030	0.47250030	0.00000106	0.00000106
0.40	0.64437760	0.64437760	0.00003355	0.00003355
0.50	0.78906250	0.78906250	0.00048822	0.00048822
0.60	0.89514240	0.89514240	0.00434883	0.00434883
0.70	0.96020470	0.96020470	0.02749099	0.02749099
0.80	0.99066880	0.99066880	0.13271659	0.13271659
0.90	0.99931410	0.99931410	0.48509263	0.48509263
0.91	0.99954284	0.99954284	0.54096053	0.54096053
0.92	0.99971009	0.99971009	0.60015420	0.60015420
0.93	0.99982739	0.99982739	0.66196973	0.66196973
0.94	0.99990537	0.99990537	0.72534048	0.72534048
0.95	0.99995366	0.99995366	0.78873115	0.78873115
0.96	0.99998072	0.99998072	0.85000627	0.85000627
0.97	0.99999381	0.99999381	0.90626750	0.90626750
0.98	0.99999876	0.99999876	0.95365311	0.95365311
0.99	0.99999992	0.99999992	0.98709160	0.98709160

Table 4.6

which the mincost and maxflow method produce paths of the same length for all numbers of paths and as a result produce the same bounds. Diagrams of both graphs can be found in the appendix.

Chapter 5

Discussion and Comparison of Results

5.1. Accuracy of Results

It is difficult to determine exactly the accuracy of the results obtained. An implementation of the bounds in a language such as Maple or Bc, that allow for calculations using much greater precision, would perhaps be better since they would ensure the precision of the results. This is especially true since for some large graphs with an extremely large number of shortest paths the subgraph bounds can not be computed because of arithmetic overflow. This is not a serious problem, however, since it only occurs when the number of shortest paths is in the millions.

All results are computed using double precision arithmetic (sixty-four bits) and are reported to only enough digits to demonstrate the difference between the bounds being compared. The values obtained for the edge-disjoint path bounds using the ‘‘C’’ implementations were checked against values obtained using the polynomials determined from the paths and a simple implementation done in Bc. It was determined that the original ‘‘C’’ implementation is in fact quite accurate.

5.2. Cases in Which the Bounds Produce Exact Values

The bounds discussed here actually produce exact reliability values for certain types of graphs. Some of these graphs are classified and explanations given as to why the method used produces a bound which is exact.

The edge-disjoint path bounds produce an exact measure of reliability if the graph contains only (s,t)-paths that are both edge and node disjoint. For example the edge-disjoint path bound computed for the graph in Figure 5.1 is an exact measure of the reliability of the graph.

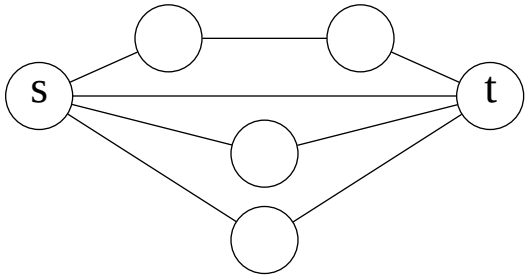


Figure 5.1

The results are exact because the graph contains only edges that are along a node and edge-disjoint path, and no other edges. The paths must be node-disjoint to prevent undercounting that would occur because alternative paths that exist at the point at which the paths are no longer node-disjoint, can not be considered.

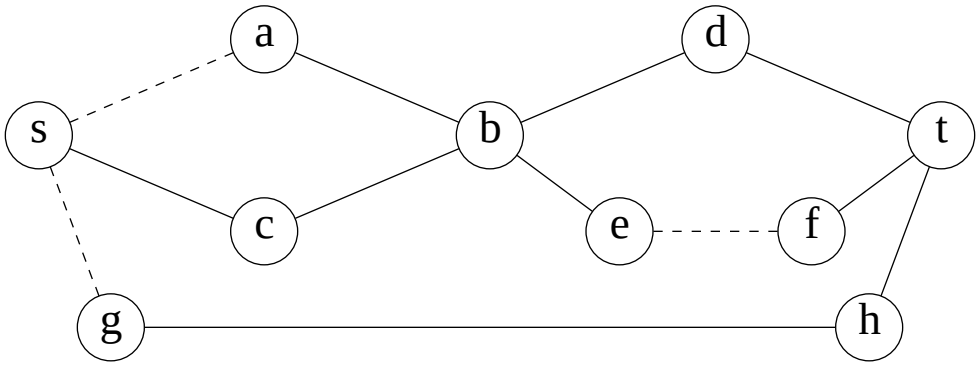


Figure 5.2

For example, consider the graph in Figure 5.2, an edge-disjoint path method for computing a lower bound for reliability finds the following three paths:

(s,g,h,t)
(s,a,b,d,t)
(s,c,b,e,f,t)

The three edges (s,a), (e,f) and (s,g) can all fail and the graph would still be operational, since the path (s,c,b,d,t) remains operational. However, the edge-disjoint path method is not able to take this into account and the result is therefore not exact.

The family of subgraph counting bounds produce an exact measure of reliability if $c \geq b - (l + k)$, where c is the size of a minimum cardinality (s,t)-cut, b is the number of edges in the graph, l is the length of a shortest path and $(l + k)$ indicates the use of exact values for pathsets of size $(l+k)$. The term k may be thought of as the value used in a KKk method. That is, for the KK0 method, $k = 0$.

For example, the KK2 method computes the exact reliability of a graph if $c \geq b - (l + 2)$. The graph in Figure 5.3 has parameters $b = 8$, $c = 4$ and $l = 2$. Therefore the KK2 method is able to compute the reliability exactly because $4 \geq 8 - (2 + 2)$.

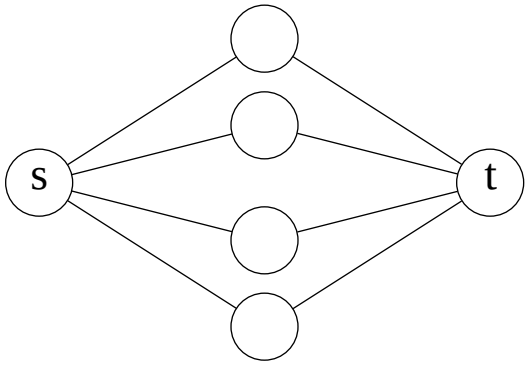


Figure 5.3

Note that the KK1 and KK0 methods do not compute the reliability of the graph in Figure 5.3 exactly. However, they do compute the exact reliability of the graphs of Figure 5.4 and 5.5 respectively. Note that the KK2 bound is exact for any graph that KK1 and KK0 measure exactly.

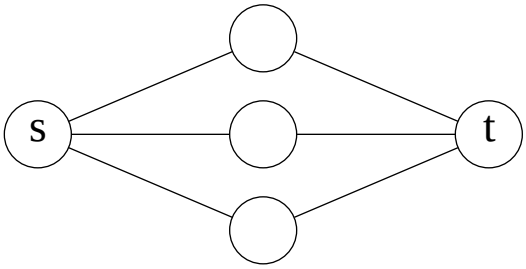


Figure 5.4

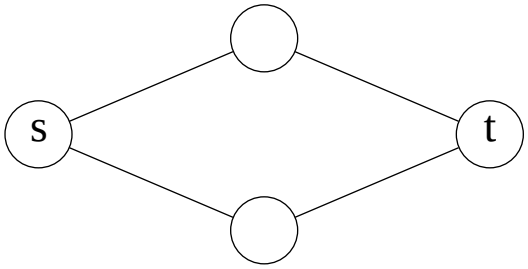


Figure 5.5

Since the graph in Figure 5.5 consists only of paths that are both node and edge-disjoint the edge-disjoint path bounds also produce exact results for this particular graph. Table 5.1 contains the results obtained by the subgraph counting method and the edge-disjoint path methods. It illustrates using a simple example that all methods produce the same exact reliability values.

5.3. The Effects of Various Parameters on the Bounds

Various parameters such as the size of the minimum cardinality (s,t)-cut and the number of edges in the graph affect the performance of each bound in different ways. The effect on each family of bounds by such parameters is discussed.

5.3.1. The Minimum Cardinality (s,t)-cutset

The accuracy of both sets of bounds is quite dependent upon the size of the

Bounds for Figure 5.5	
p	KK0=KK1=KK2=mincost=maxflow
0.10	0.0199000000
0.20	0.0784000000
0.30	0.1719000000
0.40	0.2944000000
0.50	0.4375000000
0.60	0.5904000000
0.70	0.7399000000
0.80	0.8704000000
0.90	0.9639000000
0.91	0.9704503900
0.92	0.9764070400
0.93	0.9817479900
0.94	0.9864510400
0.95	0.9904937500
0.96	0.9938534400
0.97	0.9965071900
0.98	0.9984318400
0.99	0.9996039900

Table 5.1

minimum cardinality (s,t)-cut. In the case of the edge-disjoint path bounds the size of the minimum (s,t)-cut is directly related to the number of edge-disjoint paths contained in the graph.

Generally the edge-disjoint path bounds are improved by considering all of the possible edge-disjoint paths. This is especially true since the implementation of the edge-disjoint path bounds calculates a bound for each of the possible number of paths, and uses the best bound. If the larger number of paths are relatively long (unreliable) in comparison with the length (reliability) of the paths when only a few paths are known

the bound is not substantially improved, especially for values of p near 0. Although one expects to find a better bound when the maximum number of paths is used, this in not always the case.

In the case of the subgraph counting bounds it is more difficult to see the effects of changes in the the size of the minimum cutset. It is evident that it does affect the first term $\sum_{i=0}^{c-1} \binom{b}{i} p^i q^{b-i}$ of the Kruskal-Katona family of bounds. The greater the edge-connectivity between the specified nodes, the fewer terms that are estimated using the lower pseudopower. The more terms there are that are approximated, the weaker the bound. As a result the bounds are quite good for graphs with high edge-connectivity and relatively few edges.

5.3.2. Using More Paths

The edge-disjoint bounds are usually improved when more paths are used to compute the bound. The bound is improved over all values of p if the paths being added are relatively close in length to the existing paths. If the paths being added are relatively unreliable (longer), the overall bound may not be improved for all values of p. This is especially true for lower values of p, since knowing the existence of a path that is highly unreliable does not greatly increase the bound on the reliability. This was exhibited in Chapter 4, in Figure 4.4 and Table 4.1. It is clear that in this case the bound obtained using one path is better than the bound obtained using two paths for all but a few values of p that are near one.

5.3.3. Adding Terms to the Kruskal-Katona Bounds

one. having to settle for an estimate of their value is a goal often sought after. This was the goal in mind when initially pursuing methods for improving the Kruskal-Katona bounds.

The results obtained for each of the three subgraph counting methods demonstrate that knowing the exact values for the extra terms results in bounds that are equal in some cases, but are frequently better than the original Kruskal-Katona bounds. The sub-graph counting bounds form a strict hierarchy with respect to accuracy in the order in which they make use of the most information.

$$KK2 \geq KK1 \geq KK0.$$

5.4. Comparison of the Bounds

A few examples of some networks for which each type of bound produces an exact measure of a reliability were presented earlier in this chapter. What may not be evident in the discussion is the small number of graphs for which the subgraph counting bounds produce exact results. Of these bounds the KK2 bound is the best bound currently implemented. Consider the 3-edge connected graphs for which the KK2 bound will be exact. For the graph to be 3-edge connected both nodes s and t must have degree ≥3. Since only simple graphs are considered the configuration shown in Figure 5.6 would result.

The KK2 bounds are exact when $c \geq b - (l + 2)$. Therefore the KK2 bound is exact for $c = 3$, when $3 \geq b - (l + 2)$. This occurs when $l = 1$ and $b = 5$ or 6 and when

Figure 5.6

$l = 2$ and $b = 6$ or 7. Table 5.2 demonstrates the small number of graphs for which the KK2 bound computes the reliability exactly.

Parameters for Exact Bounds Using KK2		
c	l	b
$c = 1$	$l \le b \le l+3$	
$c = 2$	$l = 1$ $l = 2$ $l = 3$ $l = 4$	$b = 3$, 4, *or* 5 $b = 4$, 5, *or* 6 $b = 6$ *or* 7 $b = 8$
$c = 3$	$l = 1$ $l = 2$	$b = 5$ *or* 6 $b = 6$ *or* 7
$c = 4$	$l = 1$ $l = 2$	$b = 7$ $b = 8$

Table 5.2

The edge-disjoint bounds on the other hand may produce exact measures of reliability for any value of c. Again all that is required is that the paths be node and edge-disjoint. Even though the types of graphs that the edge-disjoint path bounds produce exact results for are quite limited, they do produce exact results for a much larger number of graphs than do the subgraph counting bounds. The purpose of developing

bounds is not, however, to find a class of bounds that produce exact values for the largest number of networks. A comparison of these bounds on various types of graph is therefore demonstrated.

Comparison of Results for example flow graph 1					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00100000	0.00127464	0.00100010	0.00100010	2,mM,0
0.20	0.00800000	0.01160676	0.00806349	0.00806349	2,mM,0
0.30	0.02700000	0.04113331	0.02770932	0.02770932	2,mM,0
0.40	0.06400000	0.09726857	0.06783386	0.06783386	2,mM,0
0.50	0.12500005	0.18353275	0.13867188	0.13867188	2,mM,0
0.60	0.21600886	0.30018081	0.25257830	0.25257830	2,mM,0
0.70	0.34354639	0.44544941	0.42029539	0.42029539	2,mM,0
0.80	0.52486032	0.62325968	0.66200965	0.63992627	m,M,2,0
0.90	0.82236186	0.86692239	0.91209638	0.91141714	m,M,2,0
0.91	0.85669412	0.89311148	0.93084404	0.93324177	M,m,2,0
0.92	0.88952119	0.91793251	0.94755882	0.95203751	M,m,2,0
0.93	0.91970746	0.94058186	0.96351142	0.96758464	M,m,2,0
0.94	0.94606693	0.96022819	0.97746942	0.97978954	M,m,2,0
0.95	0.96752419	0.97612861	0.98755884	0.98872170	M,m,2,0
0.96	0.98334019	0.98779040	0.99415976	0.99464503	M,m,2,0
0.97	0.99338021	0.99516163	0.99788034	0.99803257	M,m,2,0
0.98	0.99835394	0.99879987	0.99951932	0.99954799	M,m,2,0
0.99	0.99987022	0.99990559	0.99996548	0.99996709	M,m,2,0

Table 5.3

Comparison of Results for example flow graph 2					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00100000	0.00100000	0.00100000	0.00100000	20mM
0.20	0.00800000	0.00800000	0.00800000	0.00800000	20mM
0.30	0.02700000	0.02700000	0.02700000	0.02700000	20mM
0.40	0.06400005	0.06400005	0.06400000	0.06400000	20,mM
0.50	0.12500286	0.12500286	0.12500000	0.12500000	20,mM
0.60	0.21607312	0.21607312	0.21600000	0.21600000	20,mM
0.70	0.34402590	0.34402590	0.34300000	0.34300000	20,mM
0.80	0.52064691	0.52064691	0.51200000	0.51200000	20,mM
0.90	0.76952555	0.76952555	0.72900000	0.72900000	20,mM
0.91	0.79856455	0.79856455	0.75357100	0.75357100	20,mM
0.92	0.82791235	0.82791235	0.77868800	0.77868800	20,mM
0.93	0.85724965	0.85724965	0.80435700	0.80435700	20,mM
0.94	0.88613626	0.88613626	0.83058400	0.83058400	20,mM
0.95	0.91397804	0.91397804	0.85737500	0.85737500	20,mM
0.96	0.93998630	0.93998630	0.89694743	0.89694743	20,mM
0.97	0.96312815	0.96312815	0.93704246	0.93704246	20,mM
0.98	0.98206596	0.98206596	0.96958794	0.96958794	20,mM
0.99	0.99508406	0.99508406	0.99173070	0.99173070	20,mM

Table 5.4

Comparison of Results for 7×7 Grid Graph					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00000000	0.00000000	0.00000000	0.00000000	20mM
0.20	0.00000010	0.00000021	0.00000001	0.00000001	2,0,mM
0.30	0.00000586	0.00001190	0.00000106	0.00000106	2,0,mM
0.40	0.00010478	0.00019722	0.00003355	0.00003355	2,0,mM
0.50	0.00097650	0.00167823	0.00048822	0.00048822	2,0,mM
0.60	0.00604660	0.00937065	0.00434883	0.00434883	2,0,mM
0.70	0.02824752	0.03919538	0.02749099	0.02749099	2,0,mM
0.80	0.10737420	0.13310964	0.13271659	0.13271659	2,mM,0
0.90	0.34883572	0.38717093	0.48509263	0.48509263	mM,2,0
0.91	0.38977084	0.42793439	0.54096053	0.54096053	mM,2,0
0.92	0.43517066	0.47262143	0.60015420	0.60015420	mM,2,0
0.93	0.48566356	0.52174352	0.66196973	0.66196973	mM,2,0
0.94	0.54212095	0.57602346	0.72534048	0.72534048	mM,2,0
0.95	0.60577754	0.63650305	0.78873115	0.78873115	mM,2,0
0.96	0.67828133	0.70458755	0.85000627	0.85000627	mM,2,0
0.97	0.76129053	0.78168263	0.90626750	0.90626750	mM,2,0
0.98	0.85438825	0.86731719	0.95365311	0.95365311	mM,2,0
0.99	0.94776090	0.95255284	0.98709160	0.98709160	mM,2,0

Table 5.5

Comparison of Results for K_7					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.10000000	0.14601438	0.14410896	0.14410896	2,mM,0
0.20	0.20000001	0.32980446	0.34770184	0.34770184	mM,2,0
0.30	0.30000389	0.49681702	0.56317750	0.56317750	mM,2,0
0.40	0.40019022	0.63577530	0.74907283	0.74907283	mM,2,0
0.50	0.50295448	0.74954176	0.88134766	0.88134766	mM,2,0
0.60	0.62038078	0.84350833	0.95705033	0.95705033	mM,2,0
0.70	0.77125233	0.92196549	0.98964924	0.98964924	mM,2,0
0.80	0.92592965	0.97820014	0.99879068	0.99879068	mM,2,0
0.90	0.99568255	0.99884979	0.99997524	0.99997524	mM,2,0
0.91	0.99738669	0.99930915	0.99998649	0.99998649	mM,2,0
0.92	0.99853245	0.99961489	0.99999316	0.99999316	mM,2,0
0.93	0.99925052	0.99980470	0.99999685	0.99999685	mM,2,0
0.94	0.99966195	0.99991250	0.99999872	0.99999872	mM,2,0
0.95	0.99987130	0.99996690	0.99999956	0.99999956	mM,2,0
0.96	0.99996167	0.99999020	0.99999988	0.99999988	mM,2,0
0.97	0.99999225	0.99999803	0.99999998	0.99999998	mM,2,0
0.98	0.99999923	0.99999981	1.00000000	1.00000000	mM,2,0
0.99	0.99999999	1.00000000	1.00000000	1.00000000	mM,2,0

Table 5.6

Comparison of Results for 10 Node Ladder					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00004095	0.00006878	0.00002000	0.00002000	2,0,mM
0.20	0.00107571	0.00221652	0.00063990	0.00063990	2,0,mM
0.30	0.00673858	0.01417603	0.00485410	0.00485410	2,0,mM
0.40	0.02360854	0.04627488	0.02037514	0.02037514	2,0,mM
0.50	0.06055450	0.10651398	0.06152344	0.06152344	2,mM,0
0.60	0.12856546	0.20076268	0.14947338	0.14947338	2,mM,0
0.70	0.24281563	0.33537840	0.30789248	0.30789248	2,mM,0
0.80	0.43013975	0.52278441	0.54798582	0.54798582	mM,2,0
0.90	0.72790115	0.78153877	0.83230156	0.83230156	mM,2,0
0.91	0.76317275	0.81054030	0.85864817	0.85864817	mM,2,0
0.92	0.79867108	0.83949462	0.88377459	0.88377459	mM,2,0
0.93	0.83394430	0.86805340	0.90739443	0.90739443	mM,2,0
0.94	0.86840650	0.89576728	0.92919293	0.92919293	mM,2,0
0.95	0.90130893	0.92206400	0.94882494	0.94882494	mM,2,0
0.96	0.93170596	0.94622259	0.96591276	0.96591276	mM,2,0
0.97	0.95841513	0.96734305	0.98004392	0.98004392	mM,2,0
0.98	0.97997039	0.98431075	0.99076879	0.99076879	mM,2,0
0.99	0.99456744	0.99575488	0.99759802	0.99759802	mM,2,0

Table 5.7

The tables of this section compare the results obtained using the KK0, KK2, min-cost and maxflow methods. The results of the KK1 method are not shown since KK0 is the base method and the KK2 method improves on the KK1 method for all but a few cases. The networks used are depicted in the appendix. The title of each table indicates which network was used to obtain the results and in cases where the source and target nodes are not indicated on the figures of the appendix, they are also specified in the title of the table.

The column labelled ‘‘ranking’’ displays the order of the accuracy of the various methods. The entries of the ranking column are separated by commas unless the results are equal, in which case the entries are simply concatenated. The entries in the ranking column are as follows:

m = mincost
M = maxflow
2 = KK2
0 = KK0

Most of the tables presented here are self-explanatory. However Tables 5.3 and 5.4 are of particular interest. Table 5.3 demonstrates how the maxflow method can, on occasion, produce better estimates of reliability than the mincost or any of the other methods. It is also a fine example of various properties of each of the methods used. In this case the KK2 bound is the best bound for values of $p \leq 0.70$. The mincost method gives the best bound for values of p between 0.80 and 0.90 and the maxflow method produces the best results for $p \geq 0.91$.

Table 5.4 illustrates an example of how the subgraph counting bounds can occasionally outperform the edge-disjoint path bounds by producing bounds that are at least as good or better than those produced by the edge-disjoint path methods over all values of p tested.

Arpanet 1979 s = ISI22 t = CCA					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00000100	0.00000100	0.00000100	0.00000100	mM20
0.20	0.00006400	0.00006400	0.00006410	0.00006402	m,M,20
0.30	0.00072900	0.00072900	0.00073490	0.00073077	m,M,20
0.40	0.00409600	0.00409600	0.00420043	0.00413778	m,M,20
0.50	0.01562500	0.01562500	0.01658630	0.01610659	m,M,20
0.60	0.04665600	0.04665600	0.05242051	0.05014943	m,M,20
0.70	0.11764900	0.11764900	0.14257323	0.13578609	m,M,20
0.80	0.26214753	0.26214753	0.34137068	0.33330151	m,M,20
0.90	0.53463655	0.53463655	0.73210029	0.71756913	m,M,20
0.91	0.57349806	0.57349806	0.77714686	0.76331135	m,M,20
0.92	0.61597818	0.61597818	0.82093818	0.80826410	m,M,20
0.93	0.66287585	0.66287585	0.86241100	0.85135133	m,M,20
0.94	0.71500166	0.71500166	0.90036703	0.89130686	m,M,20
0.95	0.77276540	0.77276540	0.93353284	0.92672090	m,M,20
0.96	0.83528779	0.83528779	0.96067268	0.95614715	m,M,20
0.97	0.89887710	0.89887710	0.98078046	0.97830622	m,M,20
0.98	0.95520674	0.95520674	0.99338764	0.99243873	m,M,20
0.99	0.99141387	0.99141387	0.99903806	0.99888472	m,M,20

Table 5.8

Tables 5.5, 5.6 and 5.7 present results produced for a few special classes of graphs, such as a grid graph, a complete graph and a ladder. Tables 5.8, 5.9, 5.10, 5.11, 5.12 and 5,13 are examples of fairly typical results obtained using these bounds on a network of practical importance. The results were obtained by computing two-terminal bounds for the 1979 representation of the Arpanet [4]. Each table heading specifies which node is the source and which is the target.

Arpanet 1979 s = STANFORD t = ABERDEEN					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00000001	0.00000001	0.00000001	0.00000001	2mM0
0.20	0.00000256	0.00000347	0.00000307	0.00000307	2,mM,0
0.30	0.00006561	0.00009465	0.00008545	0.00008529	2,m,M,0
0.40	0.00065536	0.00095924	0.00092404	0.00091733	2,m,M,0
0.50	0.00390625	0.00560760	0.00597310	0.00585180	m,M,2,0
0.60	0.01679616	0.02307804	0.02797578	0.02670920	m,M,2,0
0.70	0.05764801	0.07447282	0.10443722	0.09584856	m,M,2,0
0.80	0.16777680	0.20106191	0.31908334	0.28287449	m,M,2,0
0.90	0.43467225	0.47718075	0.73979754	0.67894513	m,M,2,0
0.91	0.47766090	0.51907979	0.78587890	0.72846185	m,M,2,0
0.92	0.52588736	0.56542971	0.82996950	0.77779115	m,M,2,0
0.93	0.58050051	0.61718777	0.87100699	0.82579083	m,M,2,0
0.94	0.64267519	0.67532635	0.90786687	0.87104513	m,M,2,0
0.95	0.71307107	0.74035424	0.93943518	0.91188768	m,M,2,0
0.96	0.79066412	0.81128495	0.96472400	0.94649588	m,M,2,0
0.97	0.87073662	0.88386573	0.98304592	0.97311064	m,M,2,0
0.98	0.94245497	0.94845308	0.99426922	0.99046713	m,M,2,0
0.99	0.98892312	0.99010267	0.99918169	0.99856806	m,M,2,0

Table 5.9

These tables indicate that of the methods considered the mincost method performs better than the other bounds over a wider range of success probabilities and over a larger variety of graphs. They also show that the KK2 method performs quite well when the edges have a low probability of being operational. The KK2 method often outperforms the mincost method for values of p close to zero.

Arpanet 1979 s = MOFFET t = COLLINS					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00000100	0.00000111	0.00000110	0.00000100	2,m,M0
0.20	0.00006400	0.00007719	0.00007680	0.00006451	2,m,M,0
0.30	0.00072900	0.00093674	0.00094754	0.00074867	m,2,M,0
0.40	0.00409600	0.00545653	0.00572769	0.00435707	m,2,M,0
0.50	0.01562500	0.02099609	0.02331543	0.01754761	m,2,M,0
0.60	0.04665600	0.06161578	0.07334353	0.05626351	m,2,M,0
0.70	0.11764900	0.14910011	0.19031440	0.15325505	m,M,2,0
0.80	0.26214420	0.31214026	0.41688362	0.36117736	m,M,2,0
0.90	0.53181695	0.58431138	0.75555132	0.71297036	m,M,2,0
0.91	0.56860257	0.61912742	0.79117758	0.75279088	m,M,2,0
0.92	0.60775584	0.65567540	0.82594853	0.79221896	m,M,2,0
0.93	0.64960268	0.69418497	0.85939650	0.83070040	m,M,2,0
0.94	0.69460404	0.73499619	0.89098228	0.86757278	m,M,2,0
0.95	0.74336700	0.77856975	0.92008710	0.90205007	m,M,2,0
0.96	0.79653615	0.82539838	0.94600390	0.93320541	m,M,2,0
0.97	0.85431689	0.87561200	0.96792776	0.95995187	m,M,2,0
0.98	0.91501709	0.92776300	0.98494552	0.98102104	m,M,2,0
0.99	0.97117047	0.97558544	0.99602447	0.99493904	m,M,2,0

Table 5.10

Arpanet 1979 s = MOFFET t = XEROX					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00010000	0.00011908	0.00011000	0.00001000	2,m,0M
0.20	0.00160000	0.00214075	0.00191949	0.00032000	2,m,0,M
0.30	0.00810000	0.01153930	0.01051032	0.00243000	2,m,0,M
0.40	0.02560000	0.03716448	0.03557786	0.01024017	2,m,0,M
0.50	0.06250000	0.08935547	0.09179688	0.03125739	m,2,0,M
0.60	0.12960000	0.17780374	0.19728230	0.07791610	m,2,0,M
0.70	0.24010000	0.31006267	0.36781639	0.17000532	m,2,0,M
0.80	0.40960013	0.49084376	0.60306227	0.34281930	m,2,0,M
0.90	0.65635063	0.72182394	0.85916951	0.65878473	m,2,M,0
0.91	0.68623849	0.74777500	0.88185195	0.69969006	m,2,M,0
0.92	0.71732685	0.77435307	0.90331312	0.74169361	m,2,M,0
0.93	0.74979367	0.80166842	0.92332930	0.78430774	m,2,M,0
0.94	0.78390513	0.82989708	0.94165818	0.82684598	m,2,M,0
0.95	0.82002299	0.85928739	0.95803778	0.86836773	m,M,2,0
0.96	0.85853213	0.89010758	0.97218526	0.90761006	m,M,2,0
0.97	0.89952274	0.92240978	0.98379578	0.94290397	m,M,2,0
0.98	0.94181797	0.95530143	0.99254119	0.97207187	m,M,2,0
0.99	0.98038956	0.98499663	0.99806881	0.99230265	m,M,2,0

Table 5.11

Arpanet 1979 s = ISI22 t = NPS					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00010000	0.00011120	0.00011100	0.00001101	2,m,0,M
0.20	0.00160000	0.00195594	0.00198337	0.00038654	m,2,0,M
0.30	0.00810000	0.01058329	0.01123165	0.00322263	m,2,0,M
0.40	0.02560000	0.03463414	0.03952813	0.01494005	m,2,0,M
0.50	0.06250000	0.08496094	0.10598755	0.05011177	m,2,0,M
0.60	0.12960000	0.17234934	0.23473390	0.13555541	m,2,M,0
0.70	0.24010000	0.30523889	0.44219216	0.30826250	m,M,2,0
0.80	0.40960239	0.48814015	0.70711712	0.58715229	m,M,2,0
0.90	0.65825822	0.72268320	0.93401261	0.89071828	m,M,2,0
0.91	0.68955050	0.74984305	0.94894460	0.91393337	m,M,2,0
0.92	0.72288962	0.77825677	0.96193969	0.93467356	m,M,2,0
0.93	0.75877355	0.80827833	0.97293449	0.95268808	m,M,2,0
0.94	0.79770500	0.84026421	0.98190644	0.96777991	m,M,2,0
0.95	0.83991228	0.87436148	0.98888387	0.97982971	m,M,2,0
0.96	0.88474929	0.91005309	0.99395746	0.98882526	m,M,2,0
0.97	0.92966964	0.94538185	0.99729344	0.99489734	m,M,2,0
0.98	0.96900800	0.97603404	0.99914852	0.99836314	m,M,2,0
0.99	0.99408509	0.99544227	0.99988699	0.99977843	m,M,2,0

Table 5.12

Arpanet 1979 s = ISI22 t = OTI					
p	KK0	KK2	mincost	maxflow	ranking
0.10	0.00001000	0.00001009	0.00001001	0.00001001	2,mM,0
0.20	0.00032000	0.00033024	0.00032256	0.00032256	2,mM,0
0.30	0.00243000	0.00258309	0.00249546	0.00249545	2,m,M,0
0.40	0.01024000	0.01122304	0.01088971	0.01088907	2,m,M,0
0.50	0.03125000	0.03515625	0.03506363	0.03504890	2,m,M,0
0.60	0.07776000	0.08895744	0.09367643	0.09350589	m,M,2,0
0.70	0.16807000	0.19277629	0.21975106	0.21863447	m,M,2,0
0.80	0.32768296	0.36962545	0.46016306	0.45622576	m,M,2,0
0.90	0.59317036	0.64049879	0.81479049	0.80998850	m,M,2,0
0.91	0.62875304	0.67437779	0.84923144	0.84487549	m,M,2,0
0.92	0.66715159	0.71026682	0.88155914	0.87775820	m,M,2,0
0.93	0.70900836	0.74862729	0.91110175	0.90794292	m,M,2,0
0.94	0.75497317	0.78991981	0.93717557	0.93471151	m,M,2,0
0.95	0.80535700	0.83432160	0.95913461	0.95737084	m,M,2,0
0.96	0.85938691	0.88112038	0.97644534	0.97532996	m,M,2,0
0.97	0.91393000	0.92768042	0.98879593	0.98821556	m,M,2,0
0.98	0.96197252	0.96822085	0.99625145	0.99603965	m,M,2,0
0.99	0.99272644	0.99394974	0.99947014	0.99943758	m,M,2,0

Table 5.13

Chapter 6

Conclusions and Future Research

6.1. Conclusions

This thesis investigates efficiently computable lower bounds for the two-terminal reliability of a given network. The two-terminal reliability problem concerns methods for measuring the reliability with which two specified communication centers of a network can communicate. The measure of reliability used is the probabilistic connectedness, *R*, which is simply the probability that the two specified centers can communicate.

A network is modelled using a probabilistic graph in which the communication centers are represented by *n* nodes and the links between the communication centers are represented by *b* edges. Nodes are assumed to be perfectly reliable and edge failures are assumed to be statistically independent. The reliability is computed as a probability that the specified source node *s* and the target node *t* can communicate.

Chapter 1 introduces the problem of network reliability and provides some of the background necessary for the rest of the thesis. Chapter 2 discusses basic graph theoretic definitions as well as definitions specific to the problem of computing reliability.

In Chapter 3 the method used by Van Slyke and Frank [46] to develop a class of bounds known as the Kruskal-Katona bounds is discussed. These bounds, which are part of a larger family of bounds known as subgraph counting bounds, use a theorem developed by Kruskal [30] and Katona [27] to obtain bounds on the coefficients of the

reliability polynomial.

Practical methods for improving these bounds are developed by counting pathsets of size $(l+1)$ and $(l+2)$, where l is the number of edges in a shortest (s,t)-path. The practical use and improvements obtained using this method are demonstrated by comparing the bounds obtained using the original method KK0 with those obtained using exact values rather than estimates for the number of pathset of size $l+1$, (KK1) and $l+2$, (KK2). The results show that the improvements can be quite substantial.

In Chapter 4 a lower bound is described that makes use of edge-disjoint paths. Various methods for finding edge-disjoint paths are investigated and a comparison is made of some of the corresponding bounds. One possible approach is a greedy method. However, this approach does not guarantee that the maximum number of paths will be found. In order to find the maximum number of edge-disjoint paths, a maximum flow algorithm is used to obtain the edges that make up a maximum flow. The bound referred to as the *maxflow bound* is a bound that results from using the Ford and Fulkerson *maxflow labelling algorithm* [18] as the underlying method of determining the edges of a maximum flow and a greedy selection to determine the paths used to obtain the bound. Since each new path is computed in an iterative fashion a potential bound is computed after each iteration and the maximum of the potential bounds is used for the actual bound. The minimum-cost maximum flow method minimizes the total number of edges used while maximizing the number of edge-disjoint paths. Since this method minimizes the total number of edges used, it produces paths that are quite reliable. The bound referred to as the *mincost bound* is computed in a manner similar to the *maxflow*

bound, except that the underlying method used to obtain the edges of the flow is a minimum-cost maximum flow algorithm.

During preliminary tests the bounds obtained using the greedy method were not as good as bounds obtained using the *maxflow* and *mincost* bounds, for graphs in which it could not find the maximum number of edge-disjoint paths. As a result the greedy method is not discussed in the comparison of the various results. The *mincost* and *max-flow* bounds are compared and the *mincost* is shown to be the better of the two bounds for a vast majority of graphs although the *maxflow* bound is better in some instances.

In Chapter 5 the subgraph counting bounds are compared with the edge-disjoint path bounds. One advantage of the edge-disjoint path bounds is that they are far simpler to compute than the subgraph counting bounds. The edge-disjoint path bounds also have the added advantage of being applicable to networks for which failure probabilities of links are known to be different, while the subgraph counting bounds are only applicable if the edges have equal failure probability.

The results obtained using each bound show that the *mincost* bound produces the best bound for a larger variety of graphs and a wider range of success probabilities than the other methods. Although, the *mincost* bound is occasionally outperformed by the KK2 bound for small values of p.

Since no one bound produces the best result for all networks, and each of the methods does in some instance produce the best bound, it is desirable to be able to combine the bounds to obtain the best possible bound. One possible method involves the calculation of all of the bounds and keeping track of the best bound for each value of p.

A method developed recently by Colbourn and Harms [11] uses linear programming to obtain bounds that are tighter than the best of the existing all-terminal bounds. This method uses the best of the all-terminal bounds as objective functions to a linear program to produce a uniform bound that is at least as good as the best of all the incorporated bounds and occasionally better. This linear programming technique can be applied in the two-terminal case by incorporating all of these bounds into a linear program to obtain a bound that is at least as good as the best basic bound for that particular network, and better in some cases.

The linear programming bound has great potential for improvement since any new bound can also be incorporated to obtain a better bound. As well the linear programming bound will also profit from any improvements that are made to the existing basic bounds. Possible methods for improving the existing bounds are a source of problems to be considered for future research.

6.2. Future Research

Techniques have been developed here for improving the basic Kruskal-Katona bounds by computing the number of pathsets of size $(l+k)$ for fixed values of k. The method used to compute pathsets of size $(l+2)$ is relatively complex since it involves allowing for overcounting that occurs. It is apparent that this technique can be applied to obtain an exact count of the pathsets of size $(l+3)$. However, the amount of overcounting that occurs increases dramatically as k increases. Therefore, in order to make the computation practical for large networks, clever counting techniques are required for counting pathsets of size $(l+k)$. Since obtaining exact counts for any of the unknown

coefficients will improve the bound, an interesting area of research would be complexity results concerning the unknown coefficients of the reliability polynomial.

Another possibility for improving the subgraph counting bounds is to find better approximations of the number of minimum cardinality (s,t)-cuts so that a better estimate of $\bar{F}_c$ can be obtained. In general better approximations for (s,t)-cutsets of size $(c+k)$ might be obtained to improve the bound from the lower terms of the reliability polynomial. These results appear to be most useful for improving the upper bounds. However, if the new estimated values are better than those estimated using lower pseudopowers the lower bound could also be improved.

In light of the complexity result reported by Itai, Perl and Shiloach [23] it appears that the problem of finding the most reliable combination of paths and path lengths for each value of p might be computationally intractable. However, a complexity result would be valuable in this area of research. Alternatively, a method of computing a combination of paths and path lengths which improves on the mincost method would be of substantial interest. The Edmonds and Karp maxflow method [14] might be of some interest for producing sets of edge-disjoint paths that produce good bounds. As well other variations of flow techniques might prove fruitful, especially if used in conjunction with the linear programming technique that combines all of the bounds.

Another avenue of possible research is decomposition methods that allow a graph to be decomposed into smaller subgraphs. The reliability of the small subgraphs could be computed and then combined to form a bound on the reliability of the entire graph. There is a greater potential for using this technique in conjunction with the edge-disjoint

path bounds than with the subgraph counting bounds due to the property of statistical independence of the edge-disjoint paths.

Naturally the lower bounds would be complemented by good techniques for computing upper bounds. As mentioned earlier the Kruskal-Katona methods do produce upper and lower bounds. It is important to note that the methods suggested for improving the Kruskal-Katona lower bounds also improve the upper bounds. As a result any improvements to the Kruskal-Katona lower bounds actually serve also to tighten the upper bounds.

As well, different techniques for computing upper bounds should be investigated. Any bound that makes use of different information than the subgraph counting bound, but produces a bound which is good, will certainly be useful, especially when combined with other bounds by using the linear programming technique.

Not much work has been done on computing the reliability of computer networks for which the assumptions of statistical independence and perfectly reliable nodes are relaxed. Some promising work has been done by Zemel [49] and Assous [2] using the assumption that edge failures are not statistically independent. However, the dependence of edge failures has only been extended to pairs of edges. If the assumption of perfectly reliable nodes is relaxed, a bound can be obtained on the reliability of a network by noting that node-disjoint paths are also edge-disjoint. A node would simply be another component in the path. If node failures are assumed to be statistically independent node-disjoint paths can be used in much the same way edge-disjoint paths are used to obtain a bound.

Any developments in these areas of research could dramatically improve the bounds developed here. Some of the problems mentioned would surely have practical uses outside the realm of network reliability and might be interesting results in themselves. Moreover, any positive results in any of these areas would nicely complement the work done in this thesis.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, **Data Structures and Algorithms**, Addison-Wesley Publishing Company, Reading, Massachusetts, 1983.

[2] J.Y. Assous, ‘‘Bounds for Terminal Reliability’’, Temple University, Philadelphia, Pennsylvania, preprint, 1984.

[3] M.O. Ball and J.S. Provan, ‘‘Bounds on the Reliability Polynomial for Shellable Independence Systems’’, *SIAM Journal on Algebraic and Discrete Methods, Vol.* 3, pp. 166-181, 1982.

[4] M.O. Ball and J.S. Provan, ‘‘Calculating Bounds on Reachability and Connectedness in Stochastic Networks’’, *Networks, Vol.* 13, pp. 253-278, 1983.

[5] R.E. Bixby, ‘‘The Minimum Number of Edges and Vertices in a Graph with Edge Connectivity N and M N-Bonds’’, *Networks, Vol.* 5, pp. 253-298, 1975.

[6] J.A. Bondy and U.S.R. Murty, **Graph Theory with Applications**, MacMillan Press Ltd., Great Britain, 1976.

[7] R.L. Brooks, C.A.B. Smith, A.H. Stone, and W.T. Tutte, ‘‘The Dissection of Rectangles into Squares’’, *Duke Mathematical Journal, Vol.* 7, pp. 312 - 340, 1940.

[8] R.G. Busacker and P.J. Gowan, ‘‘A Procedure for Determining a Family of Minimal-Cost Network Flow Patterns’’, ORO Technical Report 15, Operations Research Office, John Hopkins University, Baltimore, Maryland, 1961.

[9] J.A. Buzacott, ‘‘A Recursive Algorithm for Finding Reliability Measures Related to the Connection of Nodes in a Graph’’, *Networks, Vol.* 10, pp. 311 - 327, 1980.

[10] M. Carey and C. Hendrickson, ‘‘Bounds on Expected Performance of Networks with Links Subject to Failure’’, *Networks, Vol.* 14, pp. 439 - 456, 1984.

[11] C.J. Colbourn and D.D. Harms, ‘‘Bounding All-Terminal Reliability in Computer Networks’’, Technical Report E-123, Computer Communications Network Group, University of Waterloo, Waterloo, Ontario, 1985.

[12] E.W. Dijkstra, ‘‘A Note on Two Problems in Connexion with Graphs’’, *Numerische Mathematik, Vol.* 1, pp. 269-271, 1959.

[13] J. Edmonds, ‘‘Matroid Partitions’’, **Mathematics of the Decision Sciences: Part I**, ed. G.B. Dantzig and A.F. Veinott Jr., American Mathematical Society, Providence, Rhode Island, pp. 335 - 346, 1968.

[14] J. Edmonds and R.M. Karp, ‘‘Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems’’, *Journal of the ACM, Vol.* 19, *No.* 2, pp. 248 - 264, April, 1972.

[15] E. El Mallah and C.J. Colbourn, ‘‘Reliability of Δ-Y Reducible Networks’’, **Proceedings of the 16th International Southeastern Conference on Combinatorics, Graph Theory and Computing,** to appear, 1985.

[16] J.D. Esary and F. Proschan, ‘‘A Reliability Bound for Systems of Maintained Interdependent Components’’, *Journal of the American Statistical Association*, *Vol.* 65, pp. 329 - 338, 1970.

[17] S. Even and R.E. Tarjan, ‘‘Network Flow and Testing Graph Connectivity’’, *SIAM Journal on Computing*, *Vol.* 4, pp. 507-518, 1975.

[18] L.R. Ford Jr. and D.R. Fulkerson, **Flows in Networks**, Princeton Univ. Press, Princeton, New Jersey, 1962.

[19] H. Frank and W. Chou, ‘‘Network Properties of the ARPA Computer Network’’, *Networks*, *Vol.* 4, pp. 213 - 239, 1974.

[20] H. Frank and I.T. Frisch, ‘‘Analysis and Design of Survivable Networks’’, *IEEE Transactions on Communications*, *Vol.* COM-18, pp. 501 - 519, 1970.

[21] F. Harary, **Graph Theory**, Addison-Wesley, Reading, Massachusetts, 1969.

[22] D.D. Harms, ‘‘An Investigation into Bounds on Network Reliability’’, M.Sc. thesis, Department of Computational Science, University of Saskatchewan, Saskatoon, Saskatchewan, 1983.

[23] A. Itai, Y. Perl, and Y. Shiloach, ‘‘The Complexity of Finding Maximum Disjoint Paths with Length Constraints’’, *Networks, Vol.* 12, pp. 277 - 286, 1982.

[24] B.D. Jensen, ‘‘A Technique for Predicting the Reliability of Large Complex Systems’’, *Bell Laboratories Memorandum*, Holmdel, New Jersey, Oct. 25, 1972.

[25] W.S. Jewell, ‘‘Optimal Flow Through Networks’’, Interim Technical Report No. 8, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1958.

[26] R.M. Karp and M.G. Luby, ‘‘Monte Carlo Algorithms for Enumeration and Reliability Problems’’, **Twenty-Fourth Annual Symposium on the Foundations of Computer Science**, Tucson, Arizona, pp. 56 - 64, Nov, 1983.

[27] G. Katona, ‘‘A Theorem on Finite Sets’’, **Theory of Graphs (Proceedings of Tihany Colloquium, September, 1966)**, ed. P. Erdös and G. Katona, Academic Press, New York, New York, pp. 187-207, 1966.

[28] A.K. Kel’mans, ‘‘Connectivity of Probabilistic Networks’’, *Automation and Remote Control, No.* 3, pp. 98 - 116, 1967.

[29] G. Kirchhoff, ‘‘Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung Galvanischer Ströme geführt wird.’’, *Annalen der Physik und Chemie, Vol.* 72, pp. 497-508, 1847. [English translation, *IRE Transactions on Circuit Theory*, CT-5, pp. 4-8, 1958.]

[30] J.B. Kruskal, ‘‘The Number of Simplices in a Complex’’, **Mathematical Optimization Techniques**, ed. R. Bellman, University of California Press, Berkeley, California, pp. 251-278, 1963.

[31] E. Lawler, **Combinatorial Optimization: Networks and Matroids**, Holt, Rinehart and Winston, San Francisco, California, 1976.

[32] C.Y. Lee, ‘‘Analysis of Switching Networks’’, *Bell Systems Technical Journal, Vol.* 34, pp. 1287 - 1315, 1955.

[33] M.V. Lomonosov and V.P. Polesskii, ‘‘Lower Bound of Network Reliability’’, *Problems of Information Transmission, Vol.* 8, pp. 118 - 123, 1972.

[34] M. Messinger and M. Shooman, ‘‘Reliability Approximations for Complex Structures’’, *IEEE Proceedings of the Annual Symposium on Reliability,* pp. 292 - 301, Washington, D.C., 1967.

[35] E.F. Moore and C.E. Shannon, ‘‘Reliable Circuits Using Less Reliable Relays’’, *Journal of the Franklin Institute, Vol.* 262, pp. 191 - 208, 281 - 297, 1956.

[36] C. St. J.A. Nash-Williams, ‘‘Edge-Disjoint Spanning Trees of Finite Graphs’’, *Journal of the London Mathematical Society, Vol.* 36, pp. 445-450, 1961.

[37] V.P. Polesskii, ‘‘A Method of Constructing a Structurally Reliable Communication Network’’, **Discrete Automata and Communication Networks** [in Russian],

Moscow, pp. 13-19, 1970.

[38] V.P. Polesskii, ‘‘A Lower Boundary for the Reliability of Information Networks’’, *Problems of Information Transmission, Vol.* 7, pp. 165-171, 1971.

[39] J.S. Provan, ‘‘The Complexity of Reliability Computation on Planar and Acyclic Networks’’, Technical Report No. UNC/ORSA/TR-83/12, University of North Carolina at Chapel Hill, November, 1983.

[40] J.S. Provan and M.O. Ball, ‘‘The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected’’, *SIAM Journal on Computing, Vol.* 12, *No.* 4, pp. 777-788, November, 1983.

[41] A. Satyanarayana and K. Wood, ‘‘Polygon-to-Chain Reductions and Network Reliability’’, Technical Report ORC 82-4, Operations Research Center, University of California, Berkeley, California, March, 1982.

[42] A.W. Shogun, ‘‘Sequential Bounding of the Reliability of a Stochastic Network’’, *Operations Research, Vol.* 34, *No.* 6, pp. 1027 - 1044, Nov. - Dec., 1976.

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

[44] W.T. Tutte, ‘‘On the Problem of Decomposing a Graph into N Connected Factors’’, *Journal of the London Mathematical Society, Vol.* 36, pp. 221-230, 1961.

[45] L.G. Valiant, ‘‘The Complexity of Enumeration and Reliability Problems’’, *SIAM Journal on Computing, Vol.* 8, pp. 410 - 421, 1979.

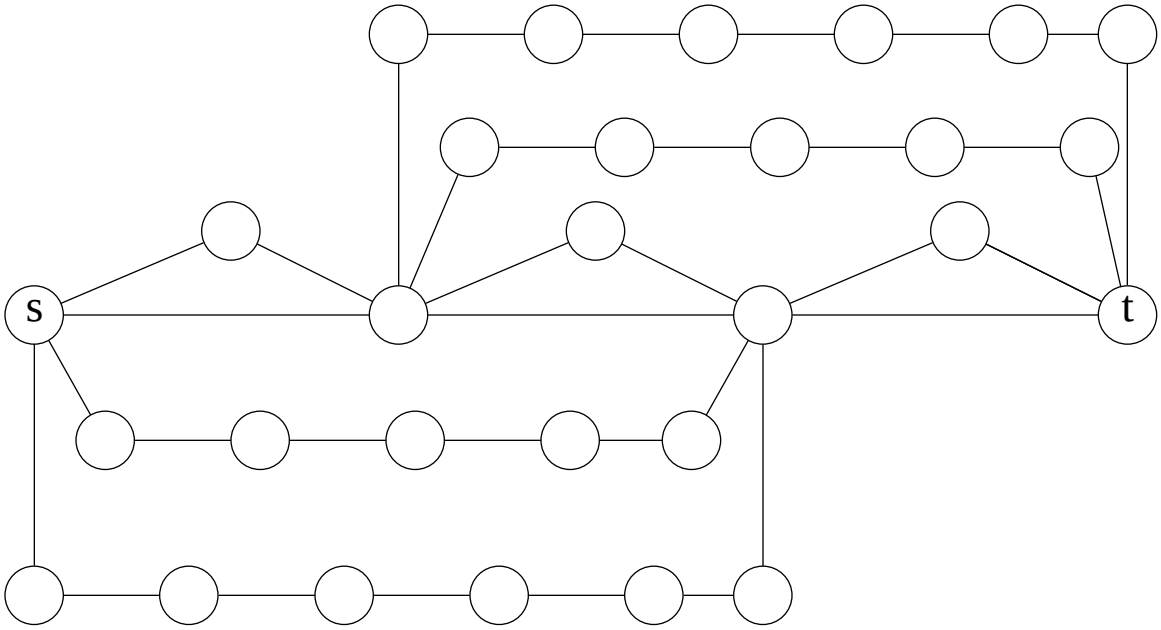
[46] R. Van Slyke and H. Frank, ‘‘Network Reliability Analysis: Part I’’, *Networks, Vol.* 1, pp. 279-290, 1972.

[47] J.A. Wald and C.J. Colbourn, ‘‘Steiner Trees in Probabilistic Networks’’, *Microelectronics and Reliability, Vol.* 23, *No.* 5, pp. 837 - 840, 1983.

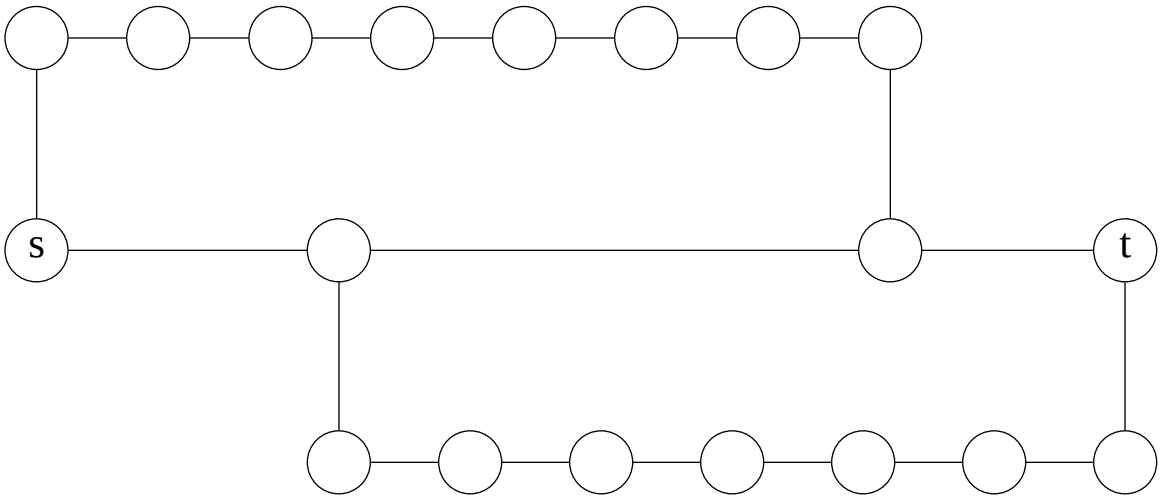
[48] R.S. Wilkov, ‘‘Analysis and Design of Reliable Computer Networks’’, *IEEE Transactions on Communications, Vol.* COM-20, pp. 660 - 678, 1972.

[49] E. Zemel, ‘‘Polynomial Algorithms for Estimating Network Reliability’’, *Networks, Vol.* 12, pp. 439 - 452, 1982.

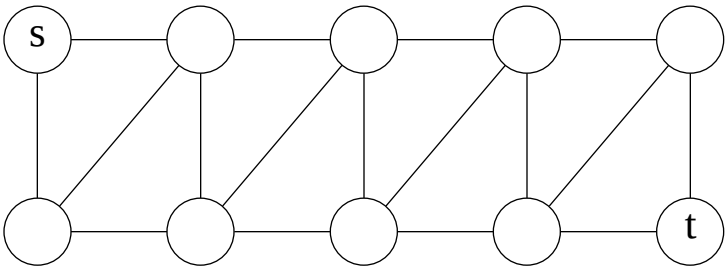
Appendix



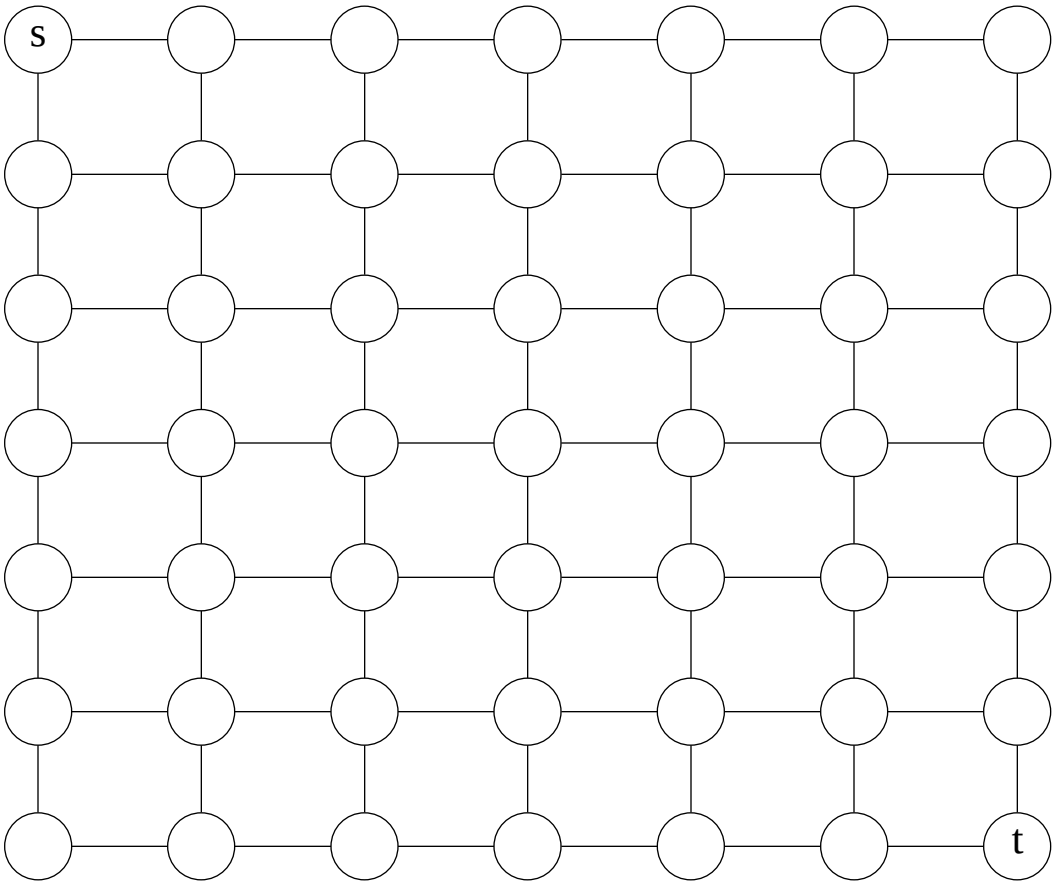
Example Flow Graph 1



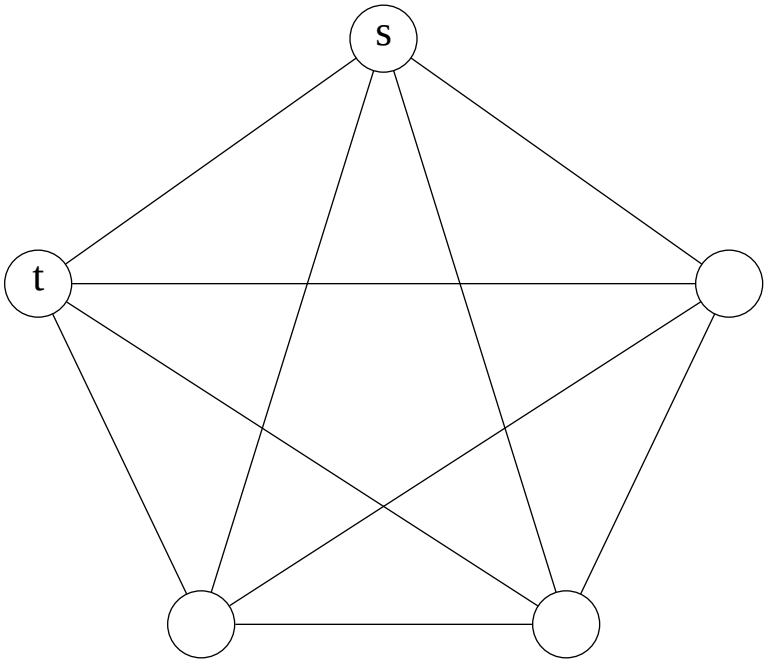
Example Flow Graph 2



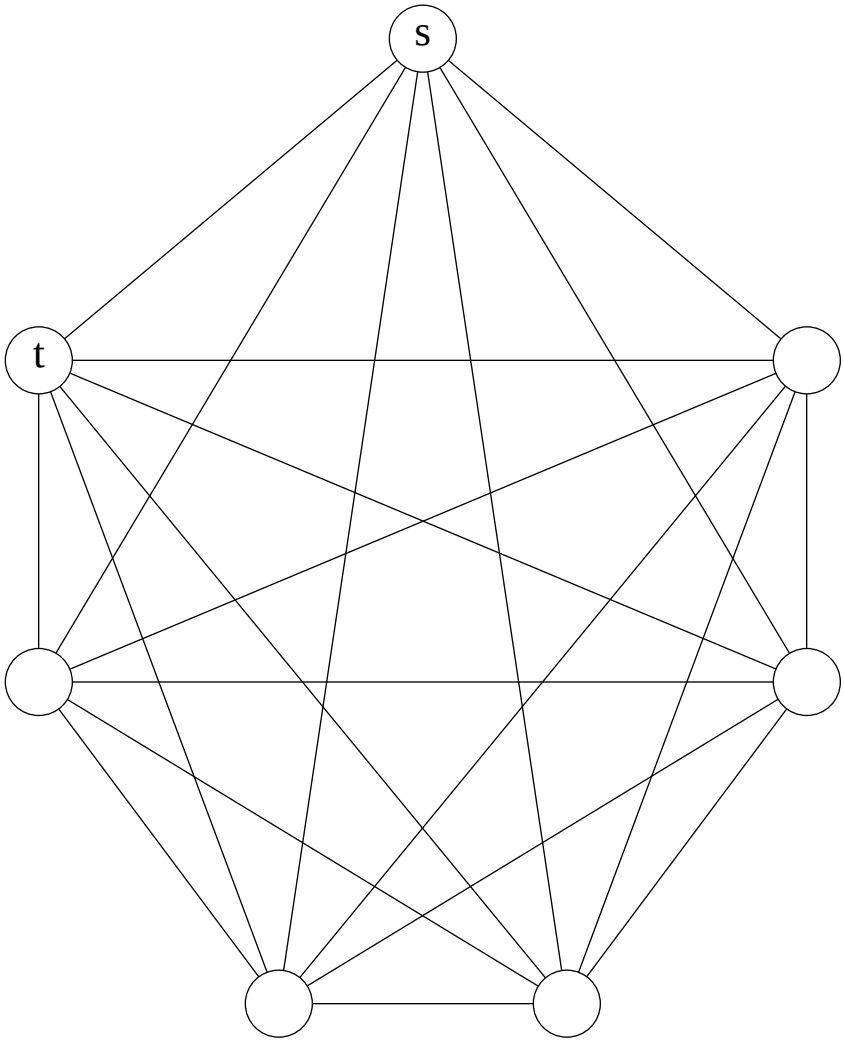
10 Node Ladder



7 × 7 Grid Graph



Complete Graph on 5 Vertices



Complete Graph on 7 Vertices

1979 Arpanet