SOME COMBINATORIAL ASPECTS OF NETWORK RELIABILITY

by

Rubin Johnson

Memorandum No. UCB/ERL M82/14

16 March 1982

ELECTRONICS RESEARCH LABORATORY
College of Engineering
University of California, Berkeley
94720

Some Combinatorial Aspects of Network Reliability

By

Rubin Johnson

A.B. (Harvard University) 1977
M.S. (University of California) 1978

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Engineering Science

in the

GRADUATE DIVISION

OF THE

UNIVERSITY OF CALIFORNIA, BERKELEY

Approved: Richard M. Karp March 9, 1982
Chairman Date
Richard E. Barlow March 9, 1982
Eugene L. Lawler March 9, 1982

Some Combinatorial Aspects

of

Network Reliability

by

Rubin Johnson

Some Combinatorial Aspects of Network Reliability

Rubin Johnson

Doctor of Philosophy

Operations Research

Sponsor:

National Science Foundation

Richard M. Karp

Chairman of Committee

Abstract

Let $G = (V,E)$ be an undirected graph with perfectly reliable vertices and unreliable edges whose failures are independent. The network reliability problem considered here is to find $R_k[G]$, the probability that a specified set of k vertices is connected with edges that are working.

Background concepts from graph theory, computational complexity, and combinatorics are reviewed. The literature of network reliability is surveyed with a focus on works related to the analysis of probabilistic networks.

Reduction techniques useful in solving network reliability problems are presented. Efficient algorithms are given for parallel edges reductions, degree two vertex reductions, biconnected component reductions, and special cases of these reductions. The Wheatstone bridge reduction is described more generally and in more detail than appears elsewhere. A different interpretation of the triconnected component reduction is given as well as a new theorem that states necessary and sufficient conditions for being able to perform this reduction.

A classification scheme for network reliability backtrack algorithms is offered. Five classes of algorithms, differing in the reduction techniques they perform, are described. Results presented about counting trees, counting acyclic orientations, and the Crapo β invariant are used in analyzing the complexity of these algorithms. For three classes of algorithms, new characterizations and proofs of optimal edge selection strategies are given. The complexity of optimal algorithms from the other two classes described is bounded.

Computational experience with programs implementing algorithms from four of the five classes is described.

To my mother and father

Acknowledgements

My gratitude goes to Richard Karp for listening and for his patience and example. I would like to thank Richard Barlow for his comments and for his role in maintaining an active community of interest in reliability at Berkeley. I thank Eugene Lawler and Roger Glassey for their suggestions and comments during various stages of this research.

Thanks go to A. Satyanarayana for insightful remarks and active research. I thank Jane Hagstrom for interesting and helpful discussions. I thank Mark Chang for long creative conversations that greatly influenced my thinking. Thanks go to Kevin Wood for his careful reading, suggestions for better algorithms, generally useful comments, and sometimes useful commas. Avinash Agrawal deserves thanks for his comments and suggestions for presenting the algorithm for biconnected component reductions.

Thanks go to Bernard Mont-Reynaud for his advice and encouragement when I was writing my first programs for network reliability. Special thanks go to Christos Papadimitriou for his encouragement and for introducing me to graph theory and combinatorics.

I would also like to thank Joerg Boysen for his advice on troff; Steve Jacobson for his graphics program that was used for the Chapter Four figures; and Toni Belcher and Diane Jones for their advice and help with all of the graphics.

Finally, I would like to acknowledge my family and friends and the National Science Foundation, without whose support all of this would have been far more difficult. (Grant Number MCS-8105217).

Table of Contents

List of Figures

Chapter One: An Introduction to Network Reliability

The network reliability problem considered here is to find the probability that a set of vertices in a network is connected. This first chapter presents various definitions and results from graph theory and computational complexity that are necessary background for the later chapters. The graph theory is necessary to understand the network model and to know the meaning of various combinatorial objects such as spanning trees and acyclic orientations. Results from computational complexity are necessary in understanding the intrinsic difficulty of some network reliability problems as well as in understanding how one quantifies the amount of work that an algorithm (a step-by-step procedure) performs in the solution of a problem. Results presented in this chapter about counting trees and acyclic orientations and the Crapo β invariant are later used in analyzing the complexity of backtrack algorithms for network reliability problems. The first chapter concludes with a survey of the literature of network reliability with a focus on works concerning the analysis of probabilistic networks.

The second chapter describes in detail various reduction techniques useful in the solution of network reliability problems. Efficient algorithms are given that perform parallel edges reductions, degree two vertex reductions, biconnected component reductions, and important special cases of these reductions. The Wheatstone bridge reduction is described more generally and in more detail than appears elsewhere. A different interpretation of the triconnected component reduction is given as well as a new theorem that states necessary and sufficient conditions for being able to perform this reduction.

Chapter Three offers a new classification scheme for network reliability backtrack algorithms and shows how classical combinatorial theory can be applied in the analysis of the complexity of these algorithms. It is shown that the essential aspects of these algorithms lie in the

reductions performed and the edge selection strategy employed. For three classes of algorithms, new characterizations and proofs of optimal edge selection strategies are given. It is shown that various combinatorial objects may be associated with each of these classes of algorithms thereby producing new proofs of algorithmic complexity and motivating algorithms to enumerate these objects. Two more classes of algorithms are presented along with some bounds on their complexity and thoughts about their optimal edge selection strategies.

Computer programs were implemented and tested for four classes of algorithms that were described in Chapter Three. Data structures, edge selection strategies, and computational experience are described in Chapter Four. This last chapter gives one an idea of some practical limitations of backtrack algorithms and suggests how one could better implement a program for network reliability problems.

1. Graph Theory Fundamentals

The reader is referred to Christofides[1975] and Harary[1969] for a more complete exposition on the fundamentals of graph theory if it is necessary to supplement the brief review of terms that follows.

A *graph* $G=(V,E)$ is a structure consisting of a finite set V of elements called *vertices* and a set E of *edges*. It will be assumed throughout that $n = |V|$ and $m = |E|$. Let the vertices be labeled $1, \cdots, n$. Each edge represents an ordered or unordered pair of vertices. If the pair of vertices (u,v) of the edge $e = (u,v)$ is unordered then the edge e is called *undirected*; otherwise, e is a *directed* edge whose *tail* is u and whose *head* is vertex v. *Undirected graphs* are those without directed edges. A *network* is distinguished from a graph in that additional information is specified about its vertices (also called terminals) and/or edges. The networks of concern in network reliability are often called *probabilistic* or *stochastic networks* because probabilistic information is supplied about the vertices and edges of the graph.

Graphs may be represented by drawings in which points or circles depict vertices and lines depict edges. Directed edges are represented by lines with arrows that point from tail to head.

An edge $e = (u, v)$ is said to be *incident* with vertices u and v. Vertices incident with the same edge are said to be *adjacent*. The *degree* of a vertex is the number of edges that are incident with it. The *in-degree* of a vertex is the number of edges directed toward a vertex; the *out-degree* is the number of edges directed out from a vertex. Vertices with in-degree zero are called *sources* while vertices with out-degree zero are *sinks*. The term *pendant* is used to describe a vertex of degree one or the edge incident on such a vertex.

Edges with the same vertex pair (u, v) are said to be in *parallel*. Replacing such edges with a single edge (u, v) is called a *parallel edges reduction*. Two edges (u, w) and (w, v) incident on the same degree two vertex are said to be in *series*; replacing them with a single edge (u, v) is called a *series reduction*.

A *path* between vertex s and vertex t is a set of edges of the form (s, v_1), (v_1, v_2), $\cdots, (v_l, t)$. If the vertices s, v_1, v_2, $\cdots, t$ are distinct then the path is called *simple*. If $s = t$, the path is a *cycle*. Two vertices x and y are *connected* if there is an (x,y)-path. A set of vertices is said to be connected if there is a path between any two vertices in the set; similarly, a graph is said to be connected if there is a path between every pair of vertices in the graph. A *subgraph* of G is a graph $G_s = (V_s, E_s)$such that $V_s \subseteq V$ and $E_s \subseteq E$. A maximally connected subgraph is called a *component*. Maximally connected means that the vertices of the component are connected and that all edges between vertices belonging to the component are in the edge set of the component. A *tree* is a connected subgraph without cycles. The term *spanning* describes a tree or subgraph in which all vertices of the graph are connected.

A component is *separable* if it is possible to separate it into more than one component by removal of a single vertex. A vertex whose removal creates components is called a *cut vertex*. An edge whose removal disconnects the graph (creates components) is called a *bridge*. A set of edges whose removal disconnects the graph is a *cutset*. A nonseparable component is called *biconnected*; at least two vertices must be removed before it is disconnected. If the removal of two vertices, neither of which is a cut vertex, can disconnect a component, those two vertices are called a *separation pair*. *Triconnected* components have no separation pair.

In defining paths and cycles, the directions of edges were ignored. Directed paths are those in which it is possible to sequentially traverse the edges of the path from tail to head. A directed graph without any directed cycles is called an *acyclic digraph*. If one orients every edge of an undirected graph without forming any directed cycles, one has created an *acyclic orientation*.

2. Graph Invariants: Factoring Theorems and Counting

A *graph invariant* of G is a number associated with the graph which has the same value for any graph *isomorphic* to G. Two graphs are said to be isomorphic if there is a one-to-one correspondence between their vertices and their edges such that the incidence relationship is preserved. The number of spanning trees in a graph is an example of such an invariant.

Let $X(G)$ be some real function of the graph $G = (V,E)$. It will be said that a *factoring theorem* holds for $X(G)$ if $X(G) = c(e)X(G^{\bullet}e) + d(e)X(G-e)$ where c and d are real functions, $G^{\bullet}e$ means that edge e has been contracted so that its two endpoints are now a single vertex, and $G-e$ means that e has been deleted from G. It has been shown by Moore and Shannon[1956] and Moskowitz[1958] that a factoring theorem holds for network reliability, that is,

$$R[G] = p_e R[G^{\bullet}e] + q_e R[G-e].$$

$R[G]$ is the probability that G has the desired connectivity properties (e.g. the existence of a path of working edges in the two terminal problem or the existence of a spanning tree of working edges in the all terminal problem). This factoring theorem motivates the backtrack algorithms that will be discussed and suggests that graph invariants for which similar theorems hold might help in determining the complexity of these algorithms.

Let $\tau(G)$ be the number of spanning trees of G. There are a number of interesting facts about this invariant. For *complete graphs* on n vertices, the number of trees is $\tau(K_n) = n^{n-2}$. (A complete graph, K_n, is one in which all n vertices are adjacent.) For arbitrary graphs one may calculate the number of spanning trees in time proportional to n^3 using Kirchoff's matrix-

tree theorem; it is only necessary to find the determinant of a matrix. Lastly, a factoring theorem $\tau(G) = \tau(G^{\bullet}e) + \tau(G-e)$ holds. Leggett[1968] calls this factoring theorem Feussner's Rule and cites a 1902 German source.

Let $\alpha(G)$ be the number of unique source acyclic orientations of G. These are those acyclic orientations where some designated vertex is the only vertex with in-degree zero. Satyanarayana and Procesi-Ciampi[1981] prove:

a) $\alpha(G)$ is independent of which vertex is the unique source;

b) $\alpha(G)$ is invariant under parallel edges reductions;

c) $\alpha(G) = 1$ if G is a tree;

d) $\alpha(G) = 0$ if G is not connected or is empty; and

e) $\alpha(G) = \alpha(G^{\bullet}e) + \alpha(G-e)$.

See also Stanley[1973] and Greene[1977]. Stanley[1977] shows that $\alpha(G)$ is equal to the absolute value of the chromatic polynomial evaluated at negative unity.

The following definitions are needed before introducing the next invariant. A *matroid* $M = (E,I)$ is a structure in which E is a finite set of elements and I is a family of subsets of E, such that

1) $\varnothing \in I$ and all proper subsets of a set I in I are also in I; and

2) if I_p and I_{p+1} are sets in I containing p and $p+1$ elements respectively, then there exists an element $e \in I_{p+1}-I_p$ such that $I_p+e \in I$.

The *rank* of $A \subseteq E$ is the cardinality of the largest subset of A that belongs to I. Sets belonging to I are called *independent*. These definitions are from Lawler[1976]. Also see Walsh[1976]. Graphs are matroids; E is the set of edges and acyclic subgraphs are independent sets.

Crapo[1967] defined an invariant for graphs (and all other finite matroids). The invariant is

$$\beta(G) = (-1)^{r(G)} \sum_{x \subseteq G} (-1)^{|x|} r(x)$$

where r is the matroid rank function. Also true is that :

a) $\beta(G)$ is invariant under parallel edges reductions;

b) $\beta(G)$ is invariant under series edges reductions;

c) $\beta(G) = 1$ if G is a single edge;

d) $\beta(G) = 0$ iff G is not biconnected or is empty; and

e) $\beta(G) = \beta(G^{\bullet}e) + \beta(G-e)$.

Further results are given by Greene[1977]. Associate with each edge $e = (a,b)$ of G the hyperplane $H_e : x_a = x_b$ in R^n. Let H represent this set of m hyperplanes. Perturb one of these hyperplanes (by letting $x_a = x_b + \epsilon$) to form H^*. Also let $N(u)$ be the number of unique source acyclic orientations rooted at u and $N(v,w)$ be the number of source-sink acyclic orientations with v the source and w the sink where $(v,w) \in E$. Greene shows:

a) $N(u)$ *is the same for all* $u \in V$;

b) $N(v,w) = \beta(G)$ *for all* $(v,w) \in E$;

c) $\alpha(G)$ = number of regions into which R^n is partitioned by H; and

d) $\beta(G)$ = number of bounded regions into which R^n is partitioned by H^*.

Graph invariants may be defined with respect to a subset of the vertices of G. Such a special subset of vertices will be denoted with K and will be assumed to contain k vertices. Such invariants have particular meaning for the k-terminal reliability problem which is to find the probability that a specified set K of k vertices is connected. The domination, $D_K(G)$, is one of these invariants. Domination theory has been important in analyzing network reliability algorithms (see Satyanarayana[1980], Satyanarayana and Hagstrom[1980a,b], Satyanarayana and Prabhakar[1978], Chang[1981], and Barlow[1982]).

This presentation of the domination follows Satyanarayana and Chang[1981]. In k-terminal network reliability, a *minimal success set* is a tree that connects all vertices in K such that all pendant vertices in this tree are also in K. This structure is also called a k-tree. If the union of a set of k-trees contains all edges in the graph then that set of k-trees is called a *formation*. If the cardinality of the set of trees is even, it is an *even formation*; otherwise, it is an *odd formation*. The *domination* of a graph G with respect to a set of vertices K is denoted $D_K(G)$ and is equal to the absolute value of the difference between the number of even and

odd formations of G. A factoring theorem is shown to hold and results are given referring to invariance under various reductions. Some of these results may be summarized as:

a) $D_K(G) = D_K(G^{\bullet}e) + D_K(G-e)$;

b) $D_V(G) = \alpha(G)$;

c) $D_{\underline{K}}(G) \leqslant D_K(G)$ if $\underline{K} \subseteq K$; and

d) $d(G) = \min_{K \subseteq G} D_K(G) = \beta(G)$.

3. Computational Complexity

A *problem*, according to Garey and Johnson[1978], is a general question to be answered along with a description of the *parameters* of the question and a statement of what properties the solution is required to satisfy. An *instance* of a problem is one particular set of values for these parameters. *Algorithms* are general step-by-step procedures for solving a problem. Analyzing how much time and space is required by algorithms to solve problems is the essence of computational complexity.

The amount of time or space that an algorithm requires to solve a given problem is called the time complexity or the space complexity and is usually expressed as a function of the size of the problem. The *size* of the problem usually relates to the amount of space required to specify the problem. In general, one is concerned only with the fastest growing term of the function that relates running time to problem size. If this function is a polynomial, the algorithm is termed *good* or *efficient*. If this function cannot be bounded by some polynomial function of the problem size then the algorithm is *exponential*.

Throughout this dissertation, the time complexity will be expressed using the notation $O[x]$ (for order x). This means that the number of computational steps that must be performed is bounded by some number that is proportional to x. Moreover, the analysis will generally refer to the worst-case complexity, the number of steps that need be performed while solving the hardest problem instance of a given size.

As an example, consider the amount if work that it takes to sort a randomly arranged list

of the integers 1, $\cdots, n$ using a *bucket sort* algorithm. The algorithm first creates n different locations (buckets). It then examines each item and places it in the appropriate bucket. This algorithm is said to be $O[n]$ or *linear* since it takes time proportional to n which is the length of the list and the size of the problem.

At the foundation of computational complexity theory are models of computation. These models define an elementary computational step and how information is encoded. Most models assume that steps are performed sequentially, one at a time, and that the next step to be performed is determined by the previous steps and the input data. The model of computation being used here assumes that arithmetic operations and comparisons are elementary steps and that binary encoding is used. The complexity of algorithms using this model are polynomially related to the complexity results with other reasonable models like Turing and RAM models. The interested reader is referred to Aho, Hopcroft, and Ullman[1974].

An algorithm (or procedure or routine) that calls itself is called *recursive*. Complexity analysis for recursive routines can be performed by determining how many steps are executed in the routine ignoring the recursive calls and then determining the total number of calls that are made to the routine while it solves the problem. If a routine calls itself either twice or not at all then one can represent the behavior of the routine as a binary search structure. See Figure 1.1. Reserving the terms *vertex* and *edge* for graphs, the points and lines of the search structure will be referred to as *nodes* and *branches*, respectively.

Each node in the search structure represents a call on the routine. Branches indicate that new recursive calls are made. Nodes without descendants are called *leaves* and represent those occasions in which the routine made no recursive calls. It is an easy fact that the number of leaves in a binary search structure is exactly one more than the number of nodes that are not leaves, that is, the number of leaves represents half the number of calls made to the routine throughout the execution of the algorithm. If one can bound the amount of work performed at each node of the search structure, then one can determine the complexity by counting the number of leaves. In Chapter Three, this type of analysis will be used extensively.

Problems in which it is possible to decide in polynomial time (time proportional to some polynomial function of the size of the answer or problem) whether or not an answer is correct are said to be in the class NP. Any problem in this class could be solved easily by algorithms based on models of computation that include oracles that can guess correct solutions to the problem. After the guess, it would only be necessary to verify that the answer was correct. Cook showed that every problem in NP could be reduced to the **satisfiability** problem. This problem, SAT, is to decide whether or not it is possible to assign values to a set of true-false variables such that a set of clauses containing these variables and their complements will each have at least one variable set to true. Cook's work showed that SAT is as hard as any problem in NP (Garey and Johnson[1978]).

The theory was extended when Karp[1972] showed that SAT could be reduced to a number of other problems in NP. The class of NP problems all as hard as SAT are known as the *NP-complete* problems. If there exists an efficient algorithm for any one of these problems, then there exist good algorithms for them all. Unfortunately, no good algorithms are known for any of these problems and it is widely felt that no polynomial time algorithm can solve these problems, that is, these problems are felt to be *inherently intractable*.

Technically, the class NP refers only to decision problems--problems that require a yes-or-no answer. The term is used more loosely here since it can be shown that most search or optimization problems can be solved using a sequence of decision problems. A *search* problem is one in which one wishes to see an example of some type of object or to know that no such objects exists. In the *optimization* problem, one wishes to find the "best" of a certain type of object. The term NP-hard refers to those problems at least as hard as the NP-complete problems; if there is a good algorithm for an NP-hard problem then there are good algorithms for all NP problems although the reverse need not hold.

Enumeration problems are those in which one wishes to know how many objects of a certain type there are. One should not confuse these enumeration problems with those of enumerating or listing all such objects; enumeration problems are solved by counting algo-

rithms while listing problems are solved by enumeration algorithms. The matrix-tree theorem discussed earlier is an efficient algorithmic solution to the enumeration problem of counting the spanning trees in a graph. Many other enumeration problems (such as counting the number of (s,t)-paths) seem far more difficult. Valiant[1977a] posed a set of problems which are among the most difficult of enumeration problems. This class, called the #P-complete (number-P-complete) problems, includes the problem of counting the number of ways to assign values to true-false variables such that a set of clauses containing these variables and their complements will have at least one variable set to true in each clause. The #P-complete problems are NP-hard.

4. A Survey of Network Reliability

The literature of network reliability is surveyed from the 1950's to the present. Attention is focused on those works that are related to the reliability analysis of probabilistic networks. Computer science, especially the theory of NP-completeness, seems to have had a profound influence upon the field.

In the early 1950's, Von Neumann[1952] lectured on "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components." These lectures appear to contain the earliest technical formulations of the principle that it is possible to make the whole better than its parts. His ideas were developed modeling computing systems such as biological neural systems, automata (machines that implement logic functions), or the newly developing large scale computing machines. Studying the operations of systems where components continually failed (and failed routinely) was part of wondering about biological systems and of pondering how electronic brains should be built.

Inspired by the work of Von Neumann, Moore and Shannon[1956] wrote a paper to give mathematical and theoretical insight into the problem of building reliable relay circuits. These authors were able to prove stronger results for relay circuits than Von Neumann could prove for computing systems. Moore and Shannon showed that with redundancy it is possible to build relay circuits *arbitrarily* more reliable than their components.

The Moore and Shannon paper seems to be the beginning of the network reliability literature. Their model was time independent. Components were assumed to fail independently with a constant failure probability. They were concerned with the probability of the system functioning at an instant in time. They introduced the factoring theorem for network reliability although they did not explore its use for the analysis problem. Moore and Shannon also expressed the reliability as a polynomial which is important in bounding the reliability of networks. Trying to find the coefficients of such polynomials is one combinatorial aspect of network reliability problems.

Von Neumann and Moore and Shannon were most concerned with the synthesis problem; they wished to design reliable systems. Moskowitz[1958] was concerned with the analysis problem; he wished to determine the reliability of the systems designed. He makes it clear that graph theory is important for network reliability problems and explores more fully the use of the factoring theorem.

During the 1950's, the study of reliability became more important for reasons including bigger computers, guided missiles, and the space program. Birnbaum, Esary, and Saunders[1961] began refining reliability theory by formalizing notions presented earlier (i.e. series components, parallel components, structure functions) and introducing new ones such as coherence. Winograd and Cowan[1963] showed that the reliability of systems that compute is conceptually different from the reliability of systems that connect (i.e. communication or transportation systems). This distinction helps to distinguish network reliability as a special case of multi-component system reliability. Birnbaum and Esary[1965] presented more results applicable to network reliability that are discussed in the next chapter.

During the middle 1960's, the difficulty of network reliability problems was becoming apparent. It was being discovered that things were harder than was hoped or predicted. Wing and Demetriou[1964] briefly discussed the horrors of state enumeration before describing a Monte Carlo estimation technique for the two terminal problem. Kel'mans[1967] also noted that substantial computational difficulties were associated with the exact analysis of large net-

works although his paper was mostly concerned with questions of network synthesis.

Interestingly enough, Kel'mans discussed a factoring theorem for spanning trees. Spanning trees were also discussed by Leggett[1968] who showed that synthesizing the most reliable network with a fixed number of vertices and a fixed number of edges (all with the same failure probability) is equivalent to synthesizing a network with the maximum number of spanning trees so long as the failure probability is small enough. Combinatorial properties of a reliability polynomial were used in proving this result.

Other work in network reliability was based on deterministic criteria. Network elements were assumed to be subject to destruction by intelligent adversaries. Research was directed at designing networks that were most survivable or invulnerable to such attacks. The criterion for vulnerability or survivability was generally a graph theoretic measure such as the edge connectivity (the minimum number of edges whose removal disconnects the graph) or the diameter (the maximum length of any shortest path) or even a composite of such measures. Steiglitz, Weiner, and Kleitman[1969] discussed designing a network of minimum cost such that the number of node disjoint paths between vertex pairs satisfy minimality constraints. Wilkov[1972] and Frank and Frisch[1970a] are useful survey papers with ample bibliographies for network reliability problems with deterministic criteria.

By the early 1970's, network analysis had developed as a distinct discipline (see Frank and Frisch[1970b]). The networks of concern were typically large scale networks that connected things--usually to allow the flow of information, oil, or some other commodity. Network reliability models with probabilistic criteria often seemed more reasonable for many of these networks which were more likely to fail due to random natural forces than to a calculated attack.

Misra[1970] suggested the recursive use of series and parallel reductions for two terminal network reliability analysis problems with a probabilistic criterion. Hänsler[1972] also suggested that reductions should be done to decrease the computational burden. Hänsler, McAuliffe, and Wilkov[1974] gave a method of enumerating cutsets to find reliability and showed that it is often more effective than enumerating the success sets. Murchland and Shier[1973] proposed

an algorithm based on the factoring theorem and series and parallel reductions. Their algorithm, which is similar to the decomposition algorithm of Moskowitz, is applied to both the two terminal and the all terminal problems. Fratta and Montanari[1973] proposed Boolean algebra methods for finding reliability. Most of these authors gave some consideration to the computational difficulties associated with their schemes.

It should be noted that the use of terms such as "fast" and "efficient" in some of the papers discussed and in other (mostly older) papers does not correspond to the current usage in the operations research and computer science literature. Now, an *efficient* algorithm is one that does its work in time proportional to some polynomial function of the problem size. (*Fast* has no formal definition but usually connotes that an algorithm does not take much real time.) New concepts and language, which developed as the study of computational complexity continued, helped researchers in network reliability express their concerns about the computational demands of their algorithms more precisely.

In his dissertation, Rosenthal[1974] showed that solving network reliability problems could be very demanding; finding the probability that at least one set of working edges connects a set of k vertices is at least as hard as solving an NP-complete problem. NP-hardness is an indication to the researcher that an efficient algorithm for the general problem is unlikely. Far from being a signal to surrender, it is an invitation to find those classes of problems amenable to polynomial time solution. The works of Wood and Satyanarayana (Wood[1980] and Satyanarayana and Wood[1982]) are two examples of finding classes of problem instances solvable with efficient algorithms. Buzacott and Chang[1980] showed that the all terminal problem with equiprobable edge failures admits to an efficient solution. Combinatorial research indicating that β evaluated for certain classes of graphs (i.e. wheels) is a polynomial function of the size of the graph (Crapo[1967]) can be used to show that there exist efficient algorithms to solve reliability problems for networks with underlying graphs of those classes.

Rosenthal[1975] introduced the new ideas of NP-completeness and computational complexity into the mainstream of the (network) reliability literature. His paper, *A Computer*

Scientist Looks at Reliability Computations, was presented at the 1975 Berkeley conference on Reliability and Fault Tree Analysis and appears in a book of the same name published in the aftermath of the conference.

Buzacott[1976] presented an exponential algorithm for the all terminal problem that also requires an exponential amount of space. The algorithm takes a dynamic programming approach, solving small subproblems and using the results to solve larger and larger problems until the whole problem is solved. Such approachs are also called *composition* methods. Buzacott and Chang[1979] and Buzacott[1980] develop these methods further.

Ball and Van Slyke[1977] discussed backtrack methods for network reliability. In full recognition of the inherent intractability of network reliability problems, they suggested that backtrack algorithms are a solution although useless for large problems. Ball[1977] discussed backtrack algorithms with reductions as well as the fact that the all terminal and two terminal problems were both open questions (NP-hard or not?) during 1977. Rosenthal[1977] discussed decomposition techniques. Rosenthal and Frisque[1977] explored reliability preserving transformations of the underlying network that would make reliability calculations easier. These works seem to be efforts to improve the state-of-the-art and not necessarily to find efficient algorithms. Satya and Prabhakar[1978] presented an improvement on the inclusion-exclusion approach to reliability analysis (see Barlow and Proschan[1975]) by showing how to avoid cancelling terms in the inclusion-exclusion reliability expression. Each term in this expression is plus or minus the probability of the intersection of events. Cancelling terms occur because the same event is often the intersection of many different combinations of events. Satyanarayana and Prabhakar were able to show that each event need be generated only once and that the domination could be used to decide the net contribution of all terms that would have been associated with this event in the usual inclusion-exclusion expression.

After a seminar given by Satyanarayana at Berkeley in 1979, Karp suggested another partition obtained by grouping together certain events of the inclusion-exclusion partition. Each event corresponds to the union of subgraphs with the desired connectivity properties. By group-

ing together subgraphs with the same depth first search tree, it is possible to determine their correct contributions more quickly and without having to compute the domination. When this idea is applied to the all terminal problem, each depth first search tree corresponds to a spanning tree. Generating these trees is achieved by using backtrack algorithms. The development of these ideas is the subject of chapters two and three.

The dual to finding efficient algorithms is showing that a class of problem instances are as hard to solve as an NP-complete problem. After Rosenthal's result, Valiant[1977b] showed that the two terminal problem was NP-hard by showing that an associated enumeration problem was #P-complete. Ball and Provan[1981b] showed that the all terminal problem is NP-hard. Jerrum[1981] independently showed that the all terminal problem is NP-hard by showing how a solution to the reliability problem can be used to find a solution to a #P-complete problem. Even finding approximate solutions is NP-hard for the k-terminal (Rosenthal[1974]) and the two terminal and all terminal problems (Ball and Provan[1981b]).

5. Remarks

As it should, research continues even if a problem is shown to be NP-hard or to admit to a polynomial time solution. If an efficient algorithm is discovered, one may try to improve it and/or try to find the best algorithms either with respect to some measure of computational complexity (i.e. best case) or for some class of problem instances. If the problem is NP-hard, one must search for easy cases or be willing to accept less than the exact answer. In network reliability, significant avenues for research include investigating new methods of bounding the reliability, new ways of obtaining approximate solutions, and efficient ways of generating estimates.

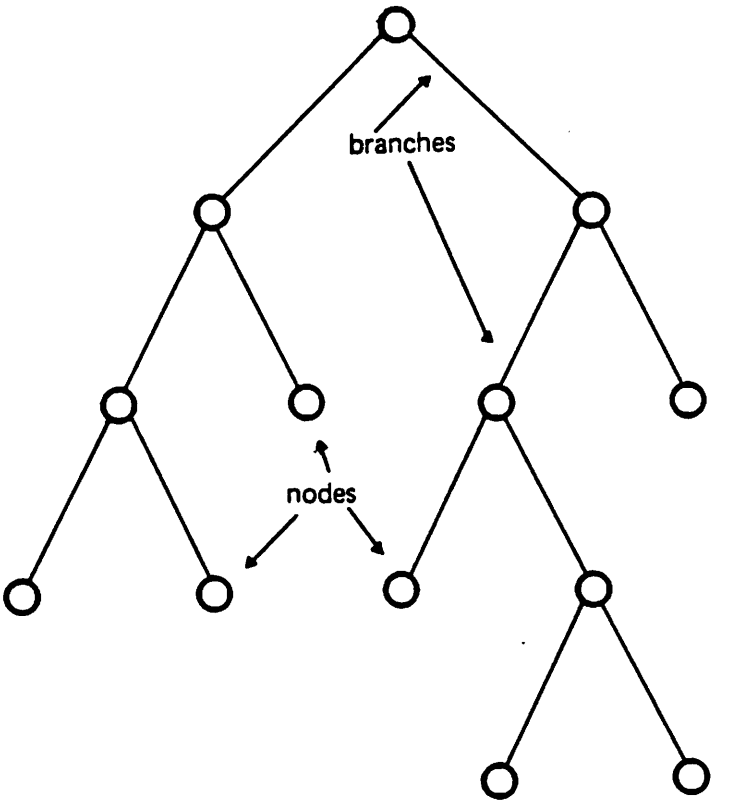


Figure 1.1 A Binary Search Structure

Chapter Two : Reduction Techniques

Let $G = (V,E)$ be an undirected network with perfectly reliable vertices and unreliable edges. Let p_i = Prob[edge e_i is working] where $0 \leqslant p_i \leqslant 1$. Let q_i = Prob[edge e_i is failed] where $0 \leqslant q_i \leqslant 1$. Assume that edge failures are independent. The k-terminal network reliability problem considered here is to calculate $R_k[G]$ = Prob[there exists a path of working edges between every pair of vertices in a set K of k vertices]. Vertices belonging to set K will be called K-vertices. $R[G]$ will be used more loosely as the probability that all K-vertices in a graph are connected.

Backtrack algorithms are useful in solving network reliability problems. The reductions (reliability preserving network transformations) discussed in this chapter are important components of these backtrack algorithms. The motivation for these backtracking algorithms is that the probability of an event is equal to the sum of the probabilities of the events of its partition. (A partition is a mutually exclusive and collectively exhaustive set of events.) More formally, if $T = T_1 \cup T_2 \cup \cdots \cup T_l$ where $T_i \cap T_j = \varnothing$ for all $i \neq j$ then Prob[T] = Prob[T_1] + Prob[T_2] + ... + Prob[T_l]. For the network reliability problem, the partition is created by a complementary branching scheme whereby edges are included (contracted) and excluded (deleted) until the set of included edges has the desired connectivity properties or the edges excluded make the desired event impossible. The branching scheme may be represented as a binary search structure.

Each node in the search structure represents a unique collection of excluded and included edges. (Note that the terms node and branch will be used in describing the search structure while the terms vertex, edge, and tree will be reserved for the description of the network and its subgraphs.) The term *good leaf* is used to designate a terminal node in the search structure

in which the set of events represented by the node imply that the network has the desired connectivity properties. The term *deadend* is used to designate a terminal node in the search structure in which the set of events represented by that node make it impossible to obtain the desired connectivity properties. Complementary branching schemes with termination conditions as described above insure that the good leaves of the search structure are a mutually exclusive and collectively exhaustive set of events whose union is the event that the k vertices of K are connected by a set of working edges.

Complementary branching schemes are the basis of the algorithms to be described. These algorithms represent an analytic approach to the network reliability problem. No attempt is made to simulate the network; rather it is attempted to continue to divide the problem into subproblems by branching. Reductions are used to transform these subproblems to smaller problems. By branching and reductions, it is possible to enumerate, recursively, a partition of events whose probabilities are easy to calculate.

Reductions play an important role in solving the network reliability problem. A reduction is any transformation that does not change the reliability of a network. A *simple reduction* is any transformation that gives a single new network and a weight w such that the product of this weight and the reliability of this new network is equal to the reliability of the original network. A *complex reduction* is any transformation that gives a set of networks whose reliabilities may be somehow combined to give the reliability of the original network. *Useful reductions* allow one to find the reliability of a graph by solving one or more smaller problems which collectively take less work to solve than solving the original graph without performing reductions. Reductions will be called *generally useful* if they seem useful for large classes of graphs although they may not be useful for all graphs. Various types of reductions include parallel edges reductions, biconnected component reductions, and triconnected component reductions. Two special cases of the triconnected component reduction are the degree two vertex reduction and the Wheatstone bridge reduction. Reductions that can be implemented easily and efficiently will be referred to as *quick reductions*.

A reduction may lead to the creation of an edge e that works with probability p and fails with probability q where $p + q \neq 1$. Such an edge is called *defective*. One could interpret p and q as probabilities conditioned on an uncertain event. The following lemma shows that it is possible to form a new network without defective edges whose reliability differs by a multiplicative factor that is easily determined.

Lemma 2.1 Given network $G = (V,E)$ with a defective edge e that works with probability p and fails with probability q, there exists a network G' with weight w' where edge e is replaced by a single non-defective edge e' such that

$$R_k[G] = w'R_k[G'].$$

Proof. By the factoring theorem for network reliability,

$$R[G] = pR[G*e] + qR[G-e].$$

To get G' replace e of G with e' where

$$p' = \frac{p}{(p+q)} \text{ and } q' = \frac{q}{(p+q)}.$$

Let $w' = p+q$. Another application of the factoring theorem yields the desired result. □

1. Parallel Edges Reductions

An important well-known type of reduction is the parallel edges reduction. Parallel edges reductions are quick, simple, and useful.

The effect of the parallel edges reduction is illustrated with the following example. See Figure 2.1. Suppose that edges $e_i = (x,y)$ for $i = 1, \cdots, l$. By definition these l edges are in parallel. They may be replaced by a single edge $e_m = (x,y)$ with

$$p_m = 1 - \prod_{i=1}^{l}(1-p_i) = 1-q_m.$$

The graph with e_m will have $l-1$ fewer edges but the same reliability as the graph with the l parallel edges.

It is important to see that each of the parallel edges performs the same structural function in that adding any or all of these parallel edges to the set of working edges causes exactly the

same set of vertices to be connected. For example, if vertex x is adjacent to an edge of the working set and the vertex y is not, then choosing any of the l parallel edges would cause vertex y to join the set of vertices connected by working edges.

This reduction may be performed on the entire network by sorting edges at each vertex. The following algorithm is due to Satyanarayana and Wood [1982]. The algorithm will be denoted PER(G) (for Parallel Edges Reduction) and will use the routine *per*(u) which finds parallel edges incident with the vertex u. Two n-dimensional arrays x and y are used.

per(u)

for every edge $e = (u, v)$ incident with vertex u

do if ($x[v] = u$)

then perform reduction on parallel edges e and $y[v]$

else set $x[v] = u$ and $y[v] = e$.

PER(G)

1. Set $x = \bar{0}$ and $y = \bar{0}$.

2. For every vertex $u \in G$ do *per*(u).

Since *per*(u) is called only once for each vertex and in all calls to this routine an edge is examined at most twice, PER(G) takes no more than $O[m+n]$ time to find and reduce all parallel edges in G.

2. Biconnected Component Reductions

The biconnected component reduction is a complex reduction. Recall that a biconnected component is a connected subgraph with no cut vertices. A biconnected component is also known as a *block*. A block is connected to the rest of the graph by vertices known as cut vertices or articulation points. Removal of any cut vertex disconnects the graph. Many authors including Ball[1977], Chang[1981], Hagstrom[1980], Murchland and Shier[1973], Rosenthal[1974], Satyanarayana[1978], and Moore and Shannon[1956] have recognized that the structure of biconnected components might aid in network reliability analysis.

The reliability of a graph is closely related to the product of the reliabilities of its blocks. If $k = |V|$, then $R_k[G] = \prod_{i=1}^{b} R[B_i]$ where B_i is the i^{th} block of G and $R[B_i]$ is the probability that all vertices of block i are connected with working edges. See Figure 2.2. When $k \neq |V|$, the relationship is more complicated and depends on the location of the K-vertices. These complications are handled recursively in the algorithm BCD(G) that is described below. By continuing to focus attention on blocks with only one cut vertex, one need not be concerned about K-vertices in neighboring blocks.

It is important to note that if a block contains none of the specified k vertices and has no edges as elements of some simple path that connects any pair of these specified vertices, then that block may be ignored in the reliability calculations. Such a block is said to be *irrelevant*.

BCD(G)

1. Find all blocks of G.
2. Set $w = 1.0$.
3. If there is only one block B then return($wR_k[B]$).
4. Find a block with one articulation point v.
    a) If this block contains no K-vertices then remove it and go to step 3.
    b) If this block contains a K-vertex then
        1) let v be a K-vertex,
        2) set $w = wR[B]$,
        3) remove the block, and
        4) go to step 3.

It is possible to find all the blocks in $O[m+n]$ time using algorithms based on depth first search. Therefore, this is a quick reduction although hard reliability problems remain to be solved in each block. The reduction is generally useful in that the work necessary to solve the problem becomes proportional to the sum of the work to solve the subproblems as opposed to being proportional to the product of the work required to solve the subproblems. However, it is

not possible to guarantee that there will be more than one block. It is possible to save no work at the expense of finding only one block.

An important special case of the biconnected component reduction is the bridge reduction. Unlike the general biconnected component reduction, the bridge reduction is simple. If a bridge is a member of every simple path between two K-vertices then the bridge must be included in the set of working edges. Excluding such a bridge would lead to a deadend. See Figure 2.3. Implementing the bridge reduction may be done efficiently, requiring effort proportional to the number of edges and vertices of the graph. One may think of the bridge reduction as including all bridges in the working set without branching.

The degree one vertex reduction or the pendant reduction is a particularly easy case of a bridge reduction. If the degree one vertex is not a K-vertex, the vertex and the adjacent edge may simply be removed and ignored. If the degree one vertex is a K-vertex, the adjacent edge is removed and the weight associated with the network must be multiplied by the working probability of the edge removed. The following two algorithms, DOR(G) and *dor*(v), show one possible way to implement degree one vertex reductions.

dor(v)

if v is a pendant incident with edge e then

1) if $v \in K$ then $w = wp_e$;

2) remove edge $e = (v, u)$; and

3) dor(u).

DOR(G)

1. For every vertex $v \in G$ do *dor*(v).

3. Degree Two Vertex Reductions

The degree two vertex reduction is a useful reduction in which a degree two vertex and the two edges adjacent to it are replaced by a single edge. Many authors have described these reductions which are usually called series reductions or series edges reductions. This section

describes two degree two vertex reductions and a situation in which such a reduction cannot be performed.

Suppose vertex v has degree two and is adjacent to edges $e_1 = (v, v_1)$ and $e_2 = (v, v_2)$. One wishes to replace edges e_1 and e_2 with the single edge $e = (v_1, v_2)$. See Figure 2.4a. Different situations dictate what calculations should be performed to determine the working and failure probabilities of the edge e or if the reduction can be performed at all. The situation depends on which vertices of v, v_1, and v_2 are in the set K.

Case 1. Vertex v is not a K-vertex.

If v is not of K then the network can operate even if both e_1 and e_2 fail. Moreover, neither of these edges can be on any path of working edges connecting two K-vertices unless both are working. It is possible to replace e_1, e_2, and v with the single edge $e = (v_1, v_2)$ that works with the probability $p = p_1 p_2$ and fails with probability $q = p_1 q_2 + p_2 q_1 + q_1 q_2$.

Case 2. Vertex v is a K-vertex. Vertices v_1 and v_2 are both K-vertices.

At least one of the edges e_1 and e_2 must work else vertex v will be disconnected and the network will fail. Only if both edges work will these edges aid in connecting the rest of the network. It is possible to replace e_1, e_2, and v with the single edge $e = (v_1, v_2)$ that works with probability $p = p_1 p_2$ and fails with probability $q = p_1 q_2 + p_2 q_1$. Such an edge may be defective.

Case 3. Vertex v is a K-vertex. Vertices v_1 and v_2 are not both K-vertices.

No reduction is possible.

A proper understanding of the degree two reduction in k-terminal network reliability may be obtained by an analysis of the four states associated with the working and failing of the edges e_1 and e_2 in the three cases described above. See Figure 2.4b.

This reduction may be performed in time proportional to the number of vertices and is quick and simple. There exist efficient algorithms that perform parallel edges reductions and degree two vertex reductions. The following is a sketch of DTR(G), an $O[n^2]$ algorithm that performs degree two vertex reductions and reduces any resulting parallel edges. It uses

PER(G), $per(u)$, and $dtr(z,j)$ which is a routine that examines vertex z to perform a degree two vertex reduction. This routine dtr also performs any resulting parallel edges reduction and causes a neighboring vertex of z to be examined if a new degree two vertex may have been created.

$dtr(z,j)$

If ((degree(z) = 2) and (case 1 or case 2)) then

1) remove the two incident edges (x,z) and (y,z);

2) add edge $e = (x,y)$ appropriately updating the weight w and finding p_e and q_e;

3) $per(x)$; and

4) if a parallel edges reduction can be performed then

if $(x \neq j)$ then $dtr(x,y)$ else $dtr(y,x)$.

DTR(G)

1. PER(G)

2. For all vertices $z = 1, \cdots, n$ do $dtr(z,z-1)$.

First perform all parallel edges reductions. This takes no more than $O[n+m]$ time. In $O[n]$ time, construct a list of the degree two vertices. Each vertex on the list can be examined in constant time to determine if a degree two vertex reduction is possible. If a reduction can be performed, a vertex and an edge need be removed and a pair of parallel edges may be created. If z, a degree two vertex adjacent to vertices x and y, is reduced then the only place parallel edges may occur is between x and y. (As the algorithm is described, it takes $O[n]$ time to find this edge although it may be possible to perform this reduction in constant time.) If there are no parallel edges, the examinations should proceed, otherwise either x or y or both may be eligible for inclusion on the list of vertices to be examined.

The list should be implemented as a stack with the vertex most recently added to the stack being the first one examined. Note that when examining vertices that have been added to the list only one adjacent vertex need be considered for inclusion in the list; the other neighbor

will have already been considered. Thus the net effect of an examination and all subsequent examinations it induces, does not increase the length of the list. Examined vertices should be removed from the list. Examinations should continue until the list is empty.

Algorithms implementing the above can perform all parallel edges and degree two vertex reductions in $O[n^2]$ time. The cumulative number of degree two vertices examined will be $O[n]$ and the search for parallel edges at each vertex is $O[n]$ so the result is obtained. Additional research might be directed at finding more efficient methods for performing these reductions, especially on sparse graphs.

4. Wheatstone Bridge Reductions

The subgraph configuration known as the Wheatstone bridge can sometimes be replaced by a single edge in a reliability preserving network transformation. This reduction is noted implicitly in many works in which authors write of triconnected component decompositions. It is mentioned explicitly by Murchland and Shier[1973] although they did not include the formulas. No one seems to have applied this reduction to k-terminal network reliability; neither does it seem that any authors have analyzed the effects this reduction has on algorithmic complexity as is done in the next chapter.

A Wheatstone bridge may be described as a subgraph $W = (V_W, E_W)$ with $V_W = \{1,2,3,4\}$ and $E_W = \{a,b,c,d,e\}$ where $a = (1,3)$, $b = (1,4)$, $c = (2,4)$, $d = (2,3)$, and $e = (3,4)$ where vertices 3 and 4 are of degree exactly 3. See Figure 2.5a.

The important characteristic of the Wheatstone bridge is that there are two vertices (necessarily of degree three) that will be adjacent only to each other and two other vertices. In the case of W, vertices 3 and 4 are adjacent only to themselves and vertices 1 and 2. There are no restrictions on how many edges may be adjacent to vertices 1 and 2. Another observation is that the Wheatstone bridge is the only configuration possible for a triconnected component with four vertices (including the two vertices of the separation pair associated with the component).

There are different kinds of Wheatstone bridges in k-terminal network reliability prob-

lems; they differ in the location of the K-vertices. The transformation to a single edge is possible if and only if either both separation pair vertices are K-vertices or neither of the interior vertices of the Wheatstone bridge is a K-vertex. In all other cases, the transformation does not preserve network reliability as non-redundant information about the location of K-vertices is lost. The inability to perform these reductions is related to the inability to perform degree two vertex reductions on a K-vertex adjacent to one or more vertices not belonging to K.

The following theorem shows how to replace a Wheatstone bridge with a single edge in a reliability preserving network transformation. The formulas for calculating the working and failure probabilities of this edge are presented.

Theorem 2.2 If W is a Wheatstone bridge that is reducible then $R[G] = R[G - E_W + w]$ where edge $w = (1,2)$ with

$$p_w = p_e p_{w_c} + q_e p_{w_d} \text{ and } q_w = p_e q_{w_c} + q_e q_{w_d}$$

where

$$p_{w_c} = p_{w_{c1}} p_{w_{c2}}$$

and

$$q_{w_c} = p_{w_{c1}} q_{w_{c2}} + p_{w_{c2}} q_{w_{c1}} + (q_{w_{c1}} q_{w_{c2}} \text{ if } \{3,4\} \cap K = \varnothing)$$

with

$$p_{w_{c1}} = p_a + q_a p_b,\ q_{w_{c1}} = q_a q_b,$$
$$p_{w_{c2}} = p_d + q_d p_c,\ \text{and } q_{w_{c2}} = q_c q_d$$

and where

$$p_{w_d} = p_{w_{d1}} p_{w_{d2}} + p_{w_{d1}} q_{w_{d2}} + q_{w_{d1}} p_{w_{d2}}$$

and

$$q_{w_d} = q_{w_{d1}} q_{w_{d2}}$$

with

$$p_{w_{d1}} = p_a p_d,$$
$$q_{w_{d1}} = p_a q_d + p_d q_a + (q_a q_d \text{ if } 3 \in K)$$

$$p_{w_{d2}} = p_b p_c,$$

and

$$q_{w_{d2}} = p_b q_c + p_c q_b + (q_b q_c \text{ if } 4 \in K).$$

Proof. When edge e is contracted in G, two parallel edges reductions and one degree two vertex reduction transform the Wheatstone bridge W into a single edge w_c with p_{w_c} and q_{w_c} as in the theorem statement. Let G_c be the name of the network obtained after these reductions are applied to G^*e. See Figure 2.5b.

When the edge e is deleted from G, two degree two vertex reductions and one parallel edges reduction transform the Wheatstone bridge W into a single edge w_d with p_{w_d} and q_{w_d} as in the theorem statement. Let G_d be the network obtained after these reductions are applied to $G-e$. See Figure 2.5c.

The reductions used in obtaining G_c and G_d insure that $R[G^*e] = R[G_c]$ and $R[G-e] = R[G_d]$. Using this fact and the factoring theorem again, one can obtain

$$\begin{aligned} R[G] &= p_e R[G_c] + q_e R[G_d] \\ &= p_e(p_{w_c} R[G_c{}^*w_c] + q_{w_c} R[G_c - w_c]) \\ &+ q_e(p_{w_d} R[G_d{}^*w_d] + q_{w_d} R[G_d - w_d]). \end{aligned}$$

Now observe that

$$G_c{}^*w_c = G_d{}^*w_d = G^*E_W$$

and

$$G_c - w_c = G_d - w_d = G - E_W.$$

Rewriting and combining terms yields

$$R[G] = (p_e p_{w_c} + q_e p_{w_d}) R[G^*E_W] + (p_e q_{w_c} + q_e q_{w_d}) R[G - E_W].$$

Observe now that $G^*E_W = G^*w$ and $G - E_W = G - w$ where w is the single edge replacing the Wheatstone bridge W.

Clearly, W may be replaced by w with

$$p_w = p_e p_{w_c} + q_e p_{w_d} \text{ and } q_w = p_e q_{w_c} + q_e q_{w_d}$$

while preserving network reliability. □

In general, edge w will be defective with $p_w + q_w < 1$.

One may find Wheatstone bridges by identifying those triconnected components with exactly five edges. Recall that a triconnected component may be separated from the rest of the graph by the removal of two vertices. Updating the triconnected component decomposition of the original network makes it possible to identify new Wheatstone bridges without having to do very many decompositions. Although this decomposition may be done in time proportional to $m+n$, the constant of proportionality is high enough to discourage doing it often.

Another way to find Wheatstone bridges is to examine all pairs of vertices corresponding to edges incident on vertices of degree three. If both vertices are of degree three, one can check their edge lists to see if they have two vertices in common. At worst, this is an $O[n]$ computation. Further research might be directed at finding better ways to identify Wheatstone bridge subgraphs.

It is possible to perform all parallel edges, degree two vertex, and Wheatstone bridge reductions in $O[n^2]$ time. First perform all parallel edges and degree two vertex reductions. This can be done in $O[n^2]$ time. It is now necessary only to show that all Wheatstone bridge reductions and reductions that they induce can be resolved in $O[n^2]$ time.

Create a list of all pairs of degree three vertices by culling the list of edges incident on degree three vertices. The length of this list is at worst $O[n]$. Each vertex pair is examined in turn with a constant time procedure. If the vertices are not the interior vertices of a Wheatstone bridge, the pair is removed from the list and the next pair examined. If a Wheatstone bridge with separation pair vertices u and v is discovered, both vertices and five edges are replaced by a single edge $e=(u,v)$. All pairs of vertices on the list containing one of the removed vertices will never belong to a Wheatstone bridge and will be removed when they are examined. If parallel edges are created by the reduction, it will be a pair of edges between vertices u and v. A list of degree two vertices requiring examination will be started with vertices u and v as its first two members. Any degree two reduction will lead to the removal of at least one vertex and

one edge. Such a reduction may also lead to a parallel edges reduction and perhaps a single degree two vertex being added to the list of vertices to be examined. After all possible degree two vertex and parallel edges reductions have been performed, at most one new vertex pair can be added to the list of degree three vertex pairs to be examined.

The following analysis will show that algorithms that implement the above ideas run in $O[n^2]$ time. Even when additions to the list of pairs of vertices are considered, the total number of pairs ever on the list is $O[n]$. There is a constant amount of time associated with the examination of each pair and doing a Wheatstone bridge reduction. A constant amount of time is associated with examining each degree two vertex. The list of degree two vertices never has more than two vertices since a new vertex isn't added until a vertex and an edge have been removed. There are at most $O[n]$ lists whose cumulative membership is also $O[n]$. The total work associated with parallel edges reductions is $O[n^2]$ since the $O[n]$ routine *per*() is used on each of the $O[n]$ times that a parallel edge may be created after a Wheatstone bridge or a degree two reduction. Therefore such an algorithm will perform all of these three reductions in $O[n^2]$ time. Described below is such an algorithm.

wbr(u, v)

1) Reduce the Wheatstone bridge with separation pair (u, v).

2) Add the edge (u, v).

3) *per*(u); *dtr*(u, v). If reductions were done and a new bridge formed then *wbr*(u, v). (Note that u or v might have been relabeled.)

WBR(G)

1. PER(G)

2. DTR(G)

3. Until there are no more pairs of degree three vertices adjacent to each other and vertices u and v do *wbr*(u, v).

Note that this reduction is a special case of a triconnected component reduction. Similar analysis yields a similar reduction for triconnected components with five vertices, two of which

belong to a separation pair. More work is necessary to determine exactly how these reductions affect algorithmic complexity.

5. Triconnected Component Reductions

The triconnected component reduction is a quick reduction. As described here, it is also a complex reduction. A triconnected component may be isolated from the rest of the graph by the removal of two vertices, neither of which is a cut vertex. Such a pair of vertices is known as a separation pair.

The idea of using triconnected component decomposition in the analysis of network reliability problems has attracted the attention of numerous authors: Birnbaum and Esary[1965], Rosenthal[1974], Ball[1977], and Hagstrom[1981]. This section will survey past use of triconnected components in network reliability analysis as well as present a different view of this idea. Implications of this reduction to the complexity of algorithms are discussed in Chapter 3.

Ball notes that Birnbaum and Esary consider replacing an entire triconnected component (with separation pair vertices u and v) by a single edge (u, v) in the two terminal network reliability problem. One would then set the working probability of this edge to the two terminal (u, v) network reliability of the triconnected component. Neither Birnbaum and Esary nor Ball specified the use of this decomposition for the k-terminal network reliability problem.

Rosenthal describes triconnected component decomposition as one of the simpler cases of replacing subgraphs with what he calls hyperedges; it is a case of using a single edge with three states (working, failed, network fails) to replace a subgraph that can be separated from the rest of the graph by the removal of one or two vertices. Rosenthal presents an algorithm for the k-terminal network reliability problem that recursively uses biconnected component reduction, simple reductions (series-parallel), and what may be called triconnected component decomposition.

Hagstrom uses the Hopcroft-Tarjan triconnected component decomposition tree to represent a graph uniquely factored into its triconnected components. She then shows how one

can solve the two terminal network reliability problem (as well as some others) by solving subproblems for each triconnected component and then using the decomposition tree to properly combine these various results to obtain the solution to the original problem. Hagstrom also points out that the concept of a triconnected component in a network is a specialization of the concept of a module in a binary coherent system.

A number of graph theoretic lemmas, a conditioning argument, and an observation will be used to motivate a technique of triconnected component decomposition for the all terminal network reliability problem. It will be assumed that the network under consideration is a block whose solution is called for in a biconnected component decomposition. Further it will be assumed that the network will have been preprocessed with parallel edges reductions and degree two vertex reductions. For this reason, the network may be assumed to be biconnected, without parallel edges, and without vertices of degree less than three.

Let a, b be a pair of vertices in a simple biconnected graph G. Suppose that the edges of G are divided into equivalence classes $E_1, E_2, \cdots, E_c$ such that two edges which lie on a common path not containing any vertex of a, b except as endpoints are in the same class. The classes E_i are called *separation classes* of G with respect to a, b. If there are at least two separation classes then a, b is called a *separation pair* unless there are exactly two separation classes and one class is a single edge. The above definitions are from Hopcroft and Tarjan[1973] and are specialized to the case where G has no parallel edges.

A triconnected decomposition partitions the edges of a graph into equivalence classes. The next lemma and its corollary help characterize these classes.

Lemma 2.3 If G is simple, biconnected, and with all vertices of degree at least three then no separation class E_i with separation pair a, b will have two, three, or four edges.

Proof. If $|E_i| = 2$ then either the two edges are in series or parallel, contrary to assumption.

If $|E_i| = 3$ then there must be exactly one vertex in $V(E_i)-a-b$ where $V(E_i)$ is the set of vertices incident with some edge of E_i. If there were more than one then some vertex would

have degree less than three. With only three vertices there is still no configuration of three edges that allows them all to be in the same equivalence class and not form parallel edges.

If $|E_i| = 4$ then there must be at least four vertices in $V(E_i)$. Since there is no way for both non-separation pair vertices to have degree three without creating parallel edges or violating the equivalence relation, this case is also impossible.

Therefore, either $|E_i| = 1$ or $|E_i| \geq 5$. □

Corollary 2.4 If $|E_i| = 1$ then the edge connects the separation pair vertices.

The next three lemmas show that the effects of branching and performing parallel edges and degree two vertex reductions are local to the triconnected component in which these operations are performed.

Lemma 2.5 Let $|E_i| \geq 5$ and $|E_j| \geq 5$. After deleting or contracting $e \in E_i$, E_j will still be a separation class with the same set of edges.

Proof. The common paths that determine membership in E_j use neither edge e nor more than one of its vertices so these paths will not be affected by deleting or contracting e. □

Lemma 2.6 Let P be a set of parallel edges in G and let E_i be a separation class with $|E_i| \geq 5$. If $P \bigcap E_i = \varnothing$ then after performing parallel reductions on P, E_i will still be a separation class with the same set of edges E_i.

Proof. None of the paths within E_i will be affected. □

Lemma 2.7 Let v be a degree two vertex of G not adjacent to any edge of E_i. After performing a degree two reduction on v, E_i will still be a separation class with the same set of edges.

Proof. The reduction will not affect any path of E_i. □

The following theorem lays the foundation for triconnected component decomposition. It

shows how one can find the combined reliabilities of a set of similar networks by solving a reliability problem on only one of these similar networks.

Theorem 2.8 Suppose t networks $G_1, G_2, \cdots, G_t$ are isomorphic to the simple network $G=(V,E)$. Associated with each network is a weight w_i. The t networks differ only in these weights and the probabilities of edge e_1.

Let p_{ij} = Prob[e_j works in G_i] and q_{ij} = Prob[e_j fails in G_i].

By assumption $p_{ij} = p_j$ and $q_{ij} = q_j$ for $j \neq 1$. Then

$$\sum_i w_i R_k(G_i) = R_k(G_{t+1})$$

where $G_{t+1} = (V,E)$ with

$$p_{t+1,j} = p_j \text{ and } q_{t+1,j} = q_j \text{ for } j \neq 1,$$
$$\text{and } p_{t+1,1} = \sum_i w_i p_{i1}, \text{ and } q_{t+1,1} = \sum_i w_i q_{i1}.$$

Proof. Note that $G^*e_1 = G_i{}^*e_1$ and $G-e_1 = G_i-e_1$ for all i and condition.

$$w_i R_k(G_i) = w_i p_{i1} R_k(G_i{}^*e_1) + w_i q_{i1} R_k(G_i-e_1)$$
$$\sum_i w_i R_k(G_i) = \sum_i w_i p_{i1} R_k(G^*e_1) + \sum_i w_i q_{i1} R_k(G-e_1)$$
$$\sum_i w_i R_k(G_i) = p_{t+1,1} R_k(G^*e_1) + q_{t+1,1} R_k(G-e_1) = R_k(G_{t+1}).\square$$

The basic principles of this technique of triconnected component decomposition are most easily explained for the network reliability problems where all vertices are K-vertices. For now, attention will be restricted to the all terminal case.

Consider the execution of a backtrack algorithm that uses both parallel edges and degree two vertex reductions. Further suppose that the graph has been partitioned into its separation classes and that branching is limited to edges that were elements of E_i ($|E_i| \geq 5$) with separation pair vertices a, b. So long as one does not select edges that are bridges with respect to $V(E_i)$, one will create a search structure whose leaves represent graphs in which E_i is replaced by a single edge. These leaves represent a partition of the success events of the network. Some linear combination of the reliabilities of the graphs represented by the graphs at these leaves is

the reliability of the original network. The probability of reaching a given leaf is the appropriate weight to be assigned to these networks that differ in one edge.

Application of Theorem 2.8 is now possible. The subproblems associated with the leaves of the search structure developed by branching on edges of the same triconnected component may be solved together. This is an example of what Chang[1981] calls backtrack fusion; subproblems associated with different nodes of the search structure are fused. This process of fusion may be continued on the various triconnected components one at a time and in any order.

Unfortunately, in the k-terminal network reliability problem, it may not always be possible to reduce all triconnected components to a single edge. The difficulties with performing the reduction are generalizations of the cases in which it is not possible to perform degree two vertex reductions and Wheatstone bridge reductions. These difficulties must be discussed in reference to particular classes of algorithms. It is assumed here that the algorithms posed for the solution of the k-terminal reliability problem are backtrack algorithms that alternately include and exclude edges in order to create a partition of events whose probabilities are easily deduced. The algorithms may employ techniques of parallel edges reductions and degree two vertex reductions. The following theorem gives necessary and sufficient conditions for performing the triconnected component decomposition when using these algorithms.

Theorem 2.9 Let T be a triconnected component of G with separation pair vertices u and v. It is possible to replace T with a single edge $t = (u, v)$ in a network reliability preserving transformation where $p_t = P(T)$ and $q_t = Q(T)$ iff

(i) both u and v are K-vertices, or

(ii)no vertex in $\{V(T)-u-v\}$ is a K-vertex.

Proof. If (i) is true, $p_t = R_{K \cap T}[T]$ and $q_t = R_{K \cap T}[T^*(u, v)]$.

If (ii) is true, $p_t = R_{u,v}[T]$ and $q_t = 1 - p_t$.

The difficult part of the proof lies in showing that if both (i) and (ii) are false, then it is impossible to find functions $P(T)$ and $Q(T)$ such that one may perform a network reliability preserving transformation replacing T by the edge $t = (u, v)$. Call this transformed graph $\underline{G}$.

If both (i) and (ii) are false then one or more (but not all) interior vertices of T are K-vertices and at least one of u and v is not a K-vertex. Without loss of generality, suppose u is not a K-vertex.

Consider all possible sequences of branching exclusively among the edges of T until T is empty and such that no K-vertex of T is ever disconnected. The leaves of such a search structure will represent one of three graphs: $G_1 = \underline{G}^{\bullet}t$ where the composite vertex of u and v is a K-vertex; $G_2 = \underline{G}-t$ where u is not a K-vertex and v is; and $G_3 = \underline{G}-t$ where u is a K-vertex. By unifying the subproblems at these leaves one gets

$$(1)\quad R_k[G] = w_1(T)R[G_1] + w_2(T)R[G_2] + w_3(T)R[G_3]$$

where $w_i(T)$ is the weight assigned to each graph. Each weight should be the sum of the probabilities associated with leaves representing the particular graph. Assume that each $w_i(T) > 0$. This assumption may be assured by initially contracting all edges in T that never fail and deleting all edges in T that never work.

When replacing T by the single edge t, it is necessary to choose between G_2 and G_3. One has the choice of

$$(2)\quad R_k[G] = P(T)R[G_1] + Q_2(T)R[G_2]$$

or

$$(3)\quad R_k[G] = P(T)R[G_1] + Q_3(T)R[G_3].$$

The facts that $R[G_1] \geqslant R[G_2]$ and $R[G_1] \geqslant R[G_3]$ are two things that can be said about the reliability of these three graphs. However, since it is possible that $R[G_1] > 0$ and $R[G_2] = R[G_3] = 0$, it must be that $P(T) = w_1(T)$.

Unfortunately, it is also possible that $R[G_2] > R[G_3] = 0$ or $R[G_3] > R[G_2] = 0$. Both cases lead to a contradiction of (1) with (2) or (3).

Hence it has been shown that, in general, T cannot be replaced by a single edge t with $p_t = P(T)$ and $q_t = Q(T)$ in a reliability preserving transformation. □

There are lots of questions about the use of this technique. Should it be used recursively? When is it worthwhile to do a triconnected component decomposition? Is is useful? These and similar questions are addressed in the next chapter where backtrack algorithms that use this and other reduction techniques are analyzed.

6. Remarks

Without reductions, backtrack algorithms for network reliability problems would be little more than naive state enumerators. Although the reductions described here have improved these algorithms considerably, they cannot stop the combinatorial explosion in the backtrack search structure generated when applying the algorithms to arbitrary networks.

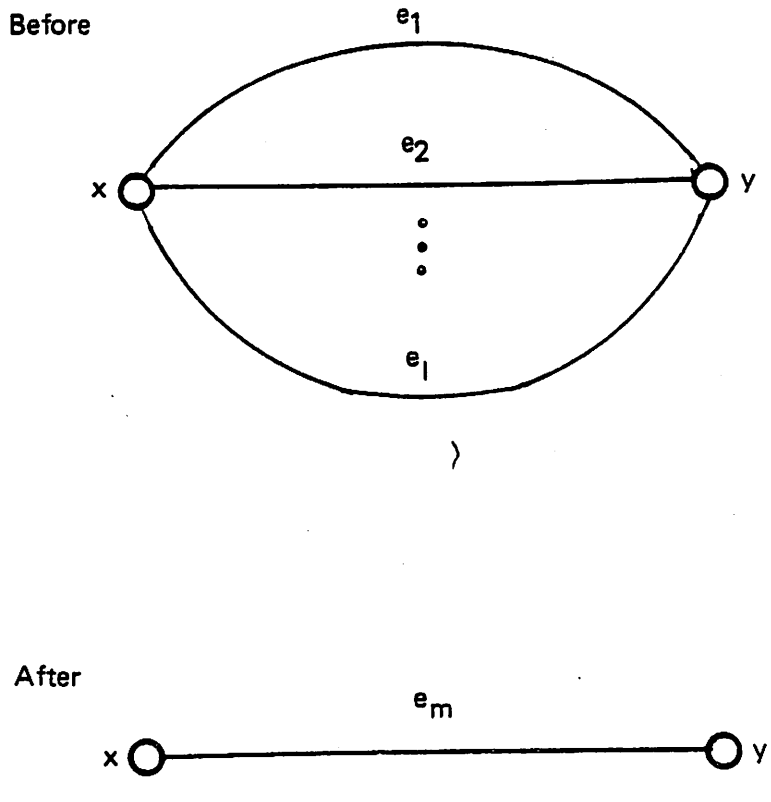


Figure 2.1 The Parallel Edges Reduction

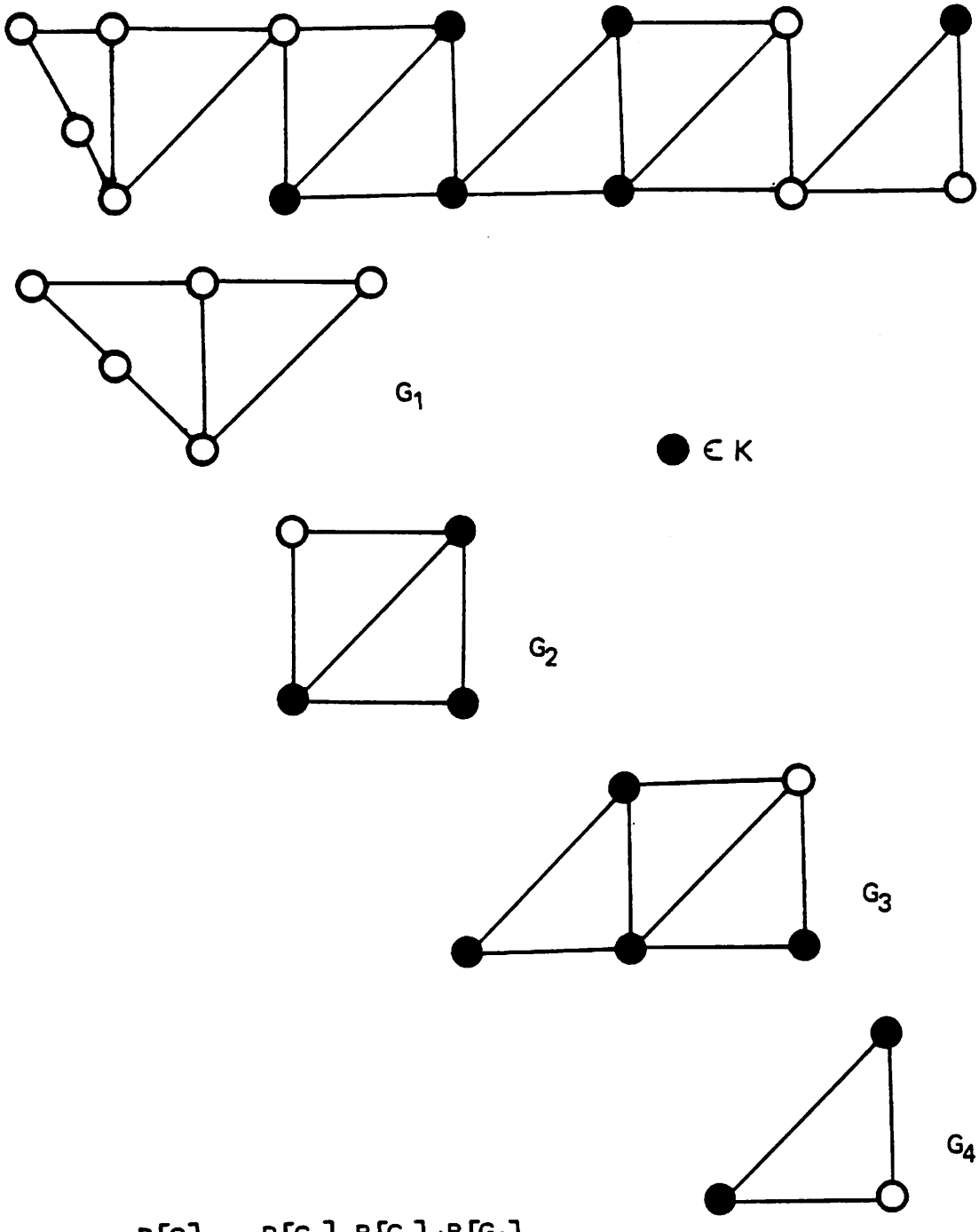


Figure 2.2 The Biconnected Components Reduction

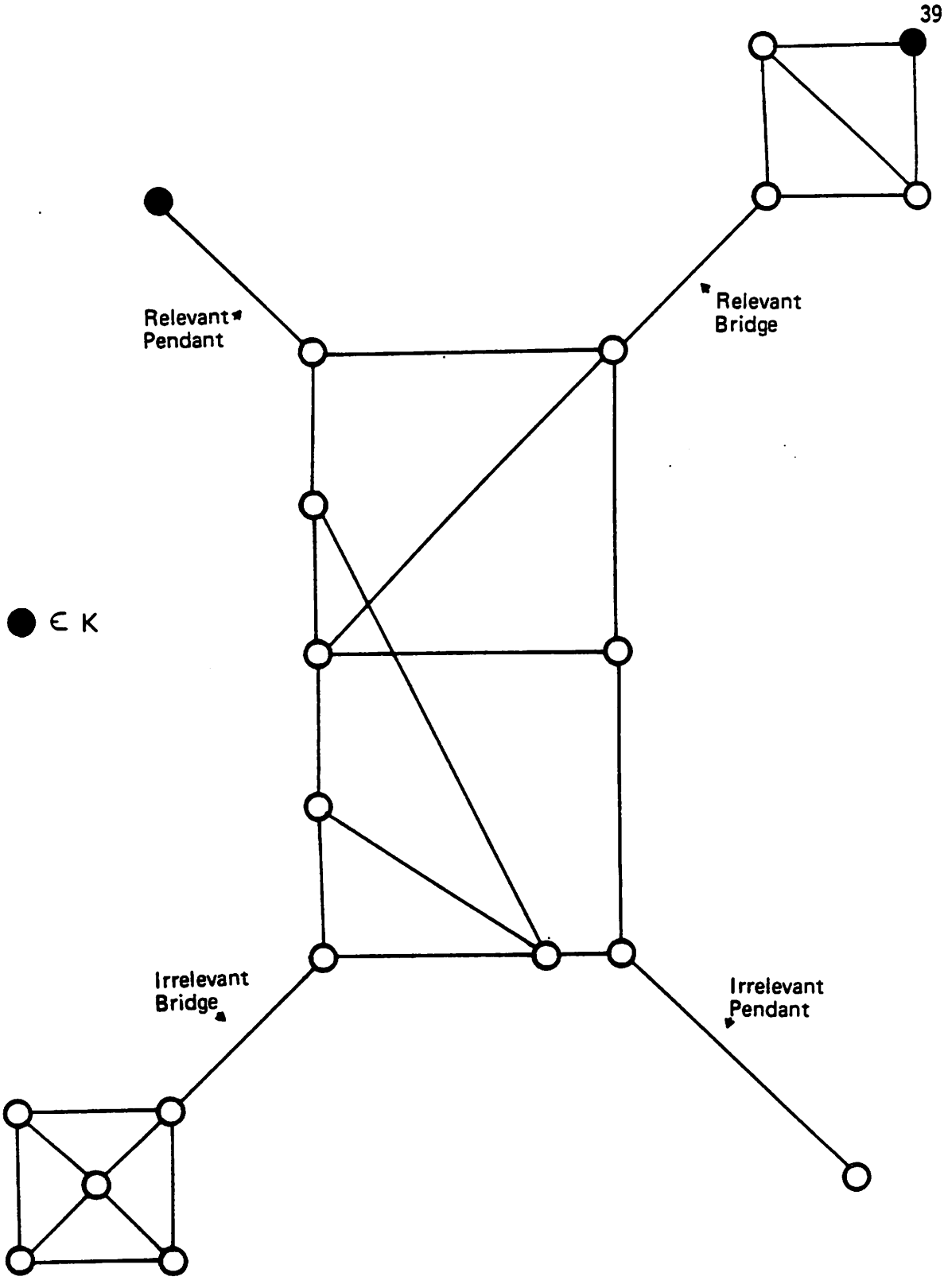


Figure 2.3 Relevant Bridge Edges

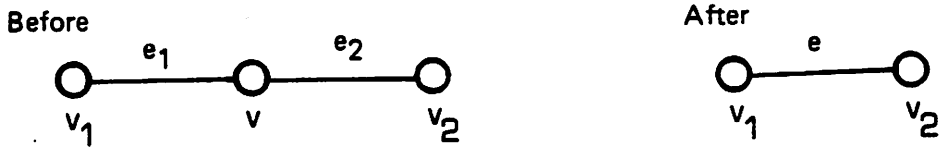


Figure 2.4a The Degree Two Vertex Reduction

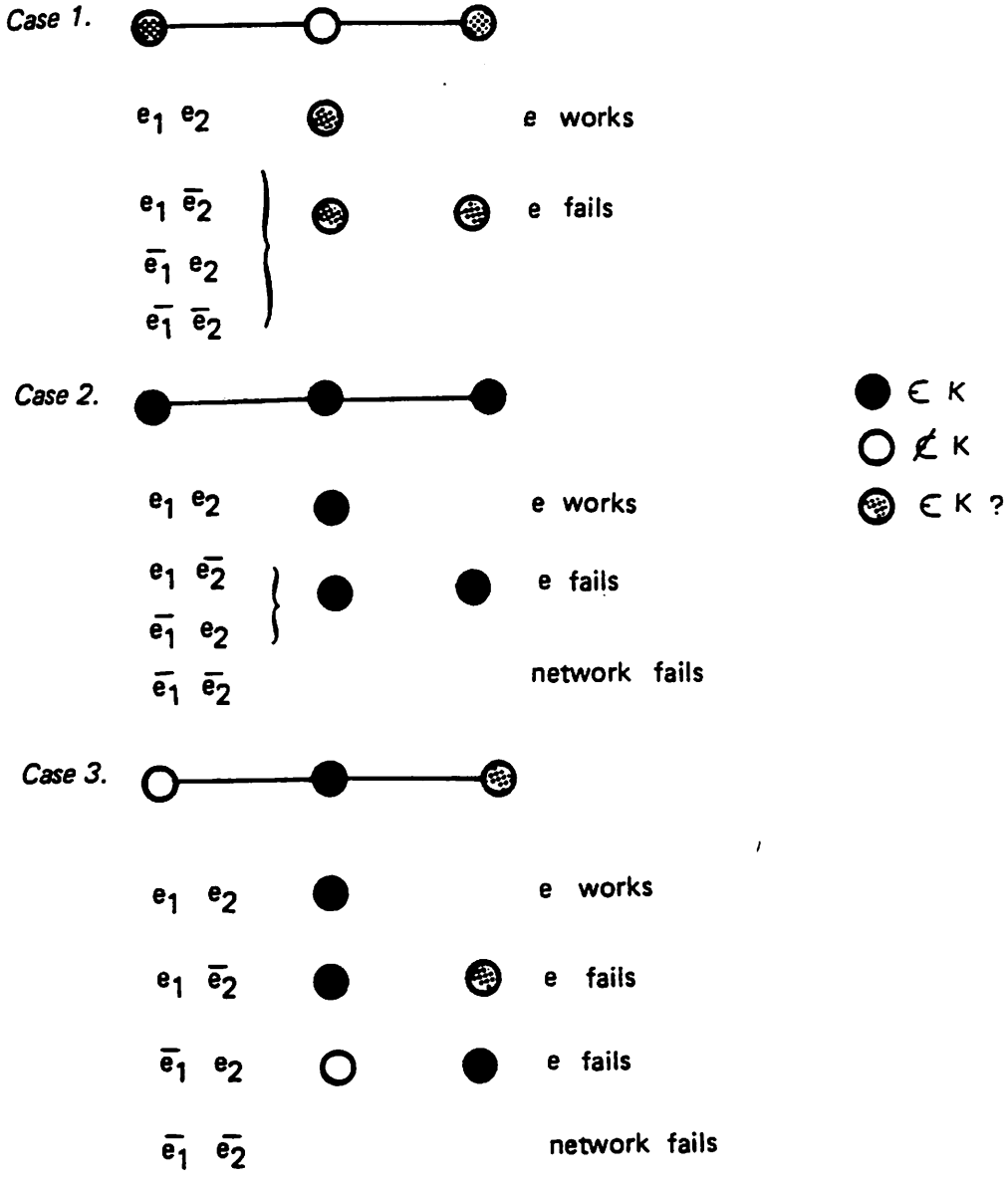


Figure 2.4b Degree Two Vertex Reductions

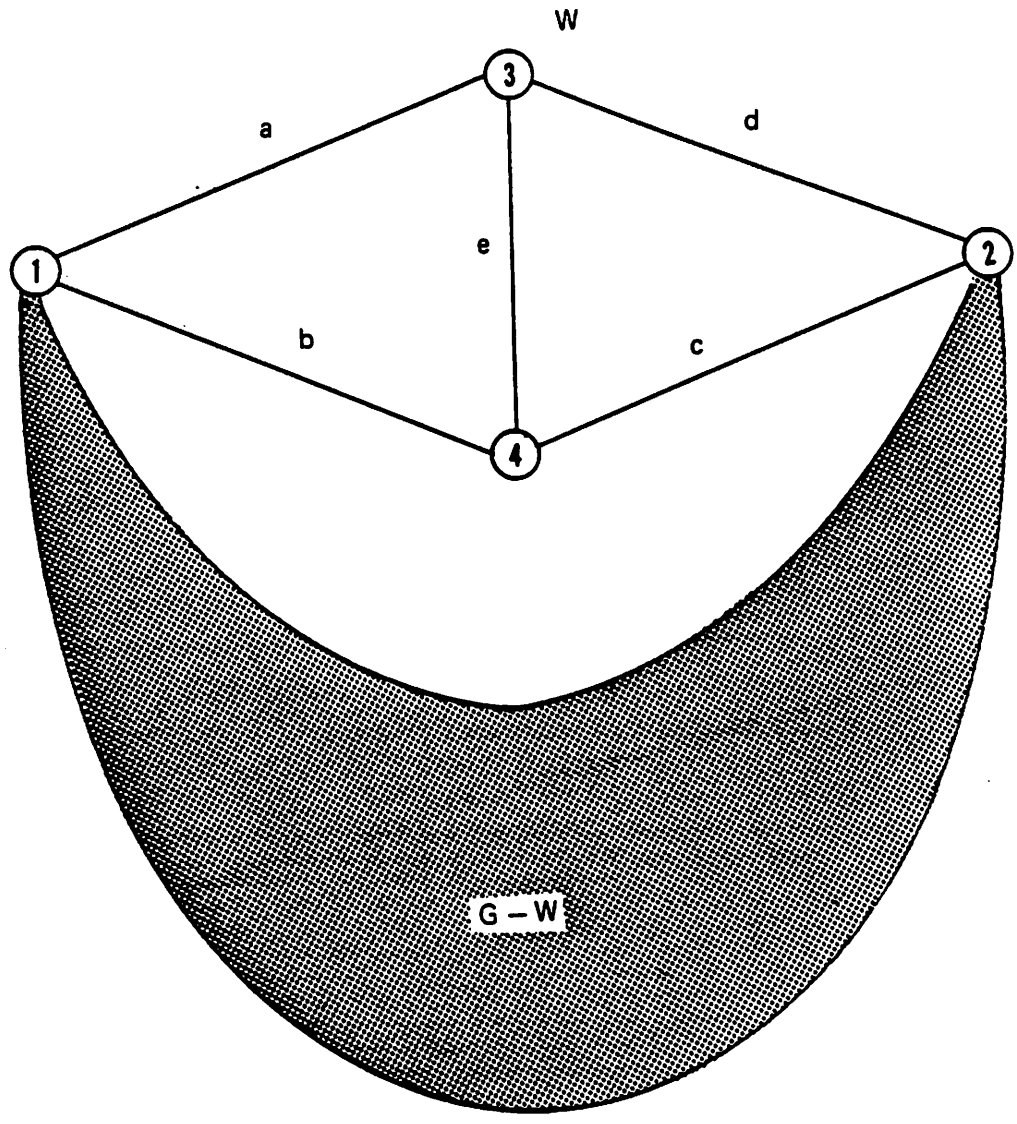


Figure 2.5a The Wheatstone Bridge Reduction

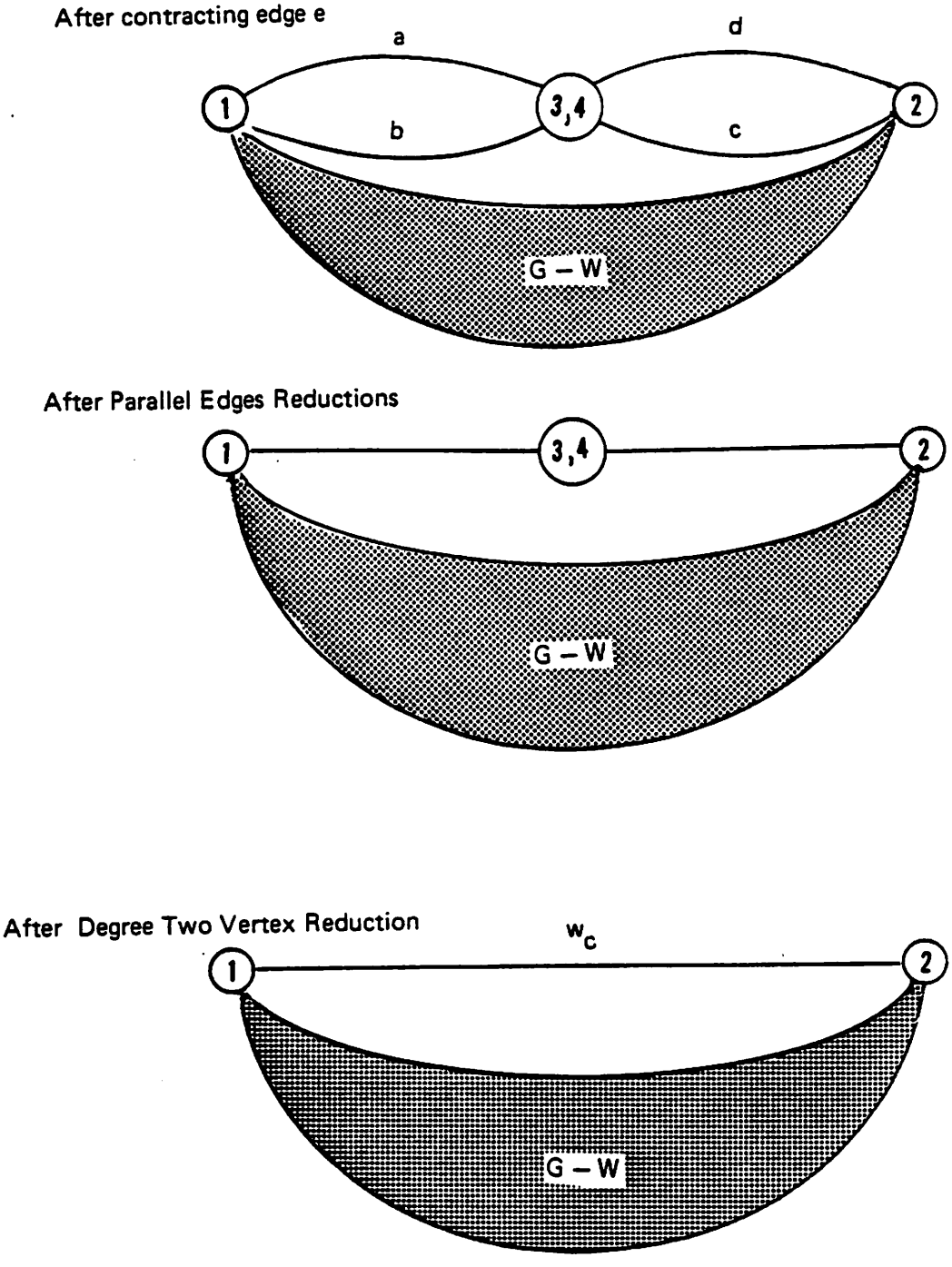
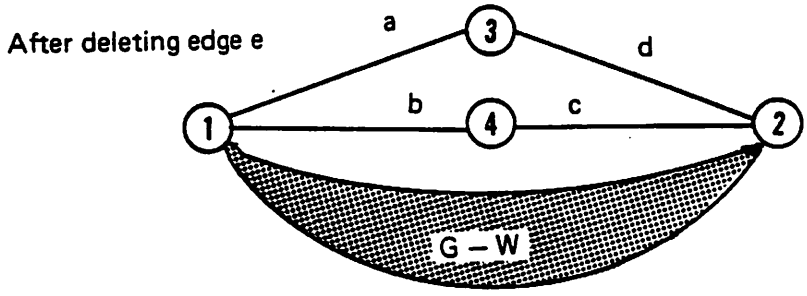
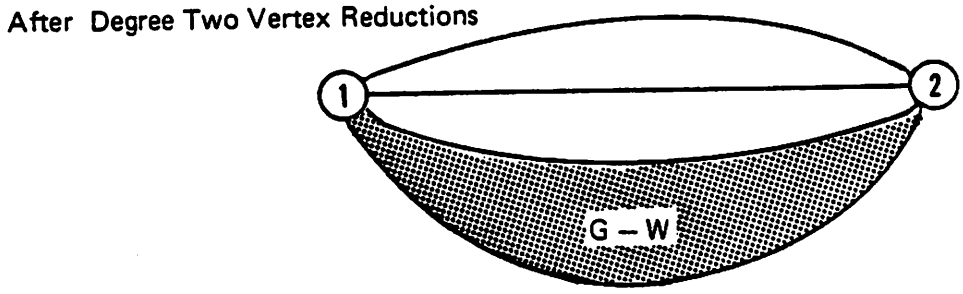


Figure 2.5b Effects of Contraction





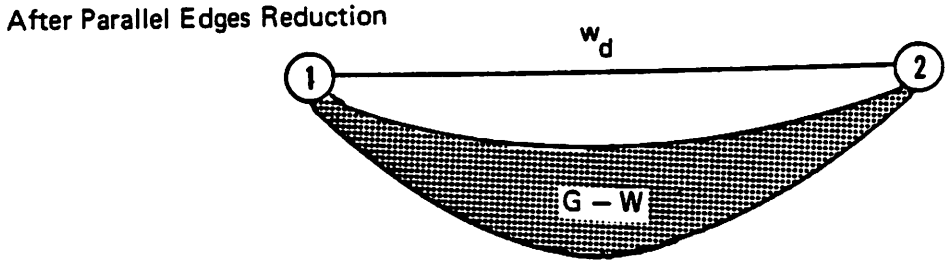


Figure 2.5c Effects of Deletion

Chapter Three : Algorithms and Complexity

Let $G = (V,E)$ be an undirected network with perfectly reliable vertices and unreliable edges. Let p_i = Prob[edge e_i is working] and q_i = Prob[edge e_i is failed]. Assume that edge failures are independent. The network reliability problem considered here is to calculate $R_k[G]$ = Prob[there exists a path of working edges between every pair of a set K of k vertices]. Five classes of backtrack algorithms for this problem are presented along with a discussion of their algorithmic complexity.

Complementary branching schemes are the basis of the backtrack algorithms described in this chapter. These algorithms find analytic solutions to network reliability problems. By branching and performing reductions, each algorithm enumerates a partition of events whose probabilities are easy to calculate. These algorithms differ from naive state enumeration in that they recognize if the graph is connected or if it cannot be connected before a state is completely specified. The reductions used and the strategies for selecting edges on which to branch are what differentiates these classes of algorithms.

Algorithms of the first class, A1, differ from blind branching in that they do not branch on bridge edges. This is achieved by performing bridge reductions. A modification to a particular algorithm of this class allows one to enumerate all the spanning trees of a graph.

Algorithms of class A2 perform parallel edges reductions. It is shown that slightly modified A2 algorithms may be used to enumerate the spanning trees of a graph in a coded form. Other arguments show that the number of leaves generated by these algorithms is equal to the number of single source acyclic orientations of the graph which equals the number of regions created by a certain arrangement of hyperplanes. Satyanarayana and Chang[1981] show that the number of leaves generated is equal to the domination, a graph invariant. It is also

possible to enumerate these acyclic orientations using slightly modified A2 algorithms.

All algorithms of A3 perform parallel edges reductions and degree two vertex reductions while attempting to select edges for branching so as not to form separable graphs. The number of leaves generated by these algorithms is equal to the minimum number of source-sink acyclic orientations of the graph which is equal to the Crapo beta invariant evaluated on the associated graphic matroid. This is the same number as the number of bounded regions in a certain arrangement of hyperplanes (see Zaslavsky[1977]) which is also the same number as the minimum domination as described in Chang[1981]. For any given graph, it is possible to give a problem instance and an edge selection strategy that will allow the enumeration of the source-sink acyclic orientations.

In addition to performing all the reductions of A3, algorithms of A4 also perform the Wheatstone bridge reduction. Bounds for the number of leaves generated are presented as well as ideas about optimal edge selection strategies.

Less is known about A5 algorithms which incorporate general triconnected component reductions into A3 algorithms. Ideas about edge selection strategies and the frequency and method of finding triconnected components are discussed in relation to algorithmic complexity.

None of the ideas embodied by any of these algorithms is novel in itself. Algorithms of classes A1 and A2 have been described by others (i.e. Murchland[1975] and Ball[1977]) although most other researchers were concerned mostly with the all terminal problem or the two terminal problem and not the *k*-terminal reliability problem.

Although all types of degree two vertex reductions have been described before, until recently no one author seems to have discussed them in their full generality. The characterization of the optimal edge selection strategy when these reductions are performed is new. The complexity analysis of class A3 algorithms which use this strategy is also new. An independent proof of this result using domination theory is in Chang[1981]. No one else seems to have noticed the relation between the complexity of these algorithms and the Crapo beta invariant.

The Wheatstone bridge reduction of the A4 algorithms is mentioned by Murchland and Shier[1973] but it does not seem that anyone developed the idea further. The bounds for the complexity of A4 algorithms are new. No one has described optimal edge selection strategies for algorithms that include Wheatstone bridge reductions or triconnected component reductions. The speculations presented here about the prospects of A5 algorithms diverge from the views of other researchers.

These backtrack algorithms enumerate a partition of events. It is sometimes possible to associate combinatorial objects with these events. In this chapter, it will be shown that trees, acyclic orientations, and regions in arrangements of hyperplanes may be associated with the leaves generated by algorithms of classes A1, A2, and A3. Moreover, it is often possible to transform enumeration algorithms into reliability algorithms and vice-versa if one knows what combinatorial objects may be associated with the events of the partition used in the reliability analysis.

1. Backtrack Algorithms with Bridge Reductions

Let A1 be a class of backtrack algorithms for the k-terminal network reliability problem. The distinguishing characteristic of this class of algorithms is that no edge that is a bridge with respect to the set of K-vertices is ever deleted. This may be accomplished by not selecting such edges for branching or by performing bridge reductions. REL1 describes algorithms of this class.

REL1(G, w)

1. Remove all irrelevant edges.
2. Perform bridge reductions. If B is the set of bridges contracted, set $w = w\prod_{e \in B} p_e$.
3. If G is a vertex then return(w).
4. Select an edge e such that $G^{\bullet}e$ and $G-e$ are connected.
5. Return(REL1$(G^{\bullet}e, wp_e)$ + REL1$(G-e, wq_e)$).

The first call should be REL1(G,1.0). Steps 1 and 2 may be done in O[$m+n$] time using a variant of a biconnected component decomposition algorithm. Recall that an irrelevant edge is one that lies on no simple path between K-vertices. In step 2, it is necessary to update w by multiplying it by the probability that all bridge edges are working. One can easily modify this algorithm so that bridges are not actually contracted in step 2. If this is done, it is necessary to modify step 3 to check if G is a tree and to return(w Prob[the tree consists of all working edges]). The edge selection strategy requires no additional labor as any non-bridge edge will satisfy the conditions of step 4. A more strenuous condition would be to require that both $G \cdot e$ and $G-e$ are coherent (have no irrelevant edges).

Of interest for complexity analysis is $L1_A(G)$ = the number of leaves generated by some algorithm $A \epsilon A1$ when calculating $R_k[G]$. First consider the problem of counting the leaves generated when solving the all terminal network reliability problem. Recall from the first chapter that $\tau(G)$ is the number of spanning trees of G.

Theorem 3.1 $L1_A(G) = \tau(G)$ for all $A \epsilon$ A1 when solving the all terminal network reliability problem.

Proof. Use induction on the number of edges. The inductive hypothesis is that the theorem holds for all graphs with $|E| \leqslant m$. This is certainly true if $m = 1$ or G is a tree.

Note that the only irrelevant edges possible in the all terminal problem are self loops. The number of spanning trees is invariant to the removal of self loops because no self loop appears in a tree. The number of spanning trees is also invariant to the reduction (contraction) of bridges since every bridge is in every spanning tree.

Consider some graph G with $m+1$ edges. If G has self loops or bridges, then the removal of a loop or the reduction of a bridge will yield a graph with m edges and the same number of trees and give the desired result. If there are no bridges or self loops, an edge will be selected and two subproblems created. The number of leaves will then be the sum of the leaves generated by these subproblems, that is,

$$L1_A(G) = L1_A(G^*e) + L1_A(G-e).$$

Since neither G^*e nor $G-e$ has more than m edges, the inductive hypothesis may be used to give

$$\tau(G^*e) = L1_A(G^*e) \text{ and } \tau(G-e) = L1_A(G-e).$$

Recall from Chapter 1 that a factoring theorem which states $\tau(G) = \tau(G^*e) + \tau(G-e)$ holds for $\tau(G)$ and the result is obtained. □

This theorem does not answer the more general question of how algorithms of this class perform for k-terminal problems. Since the removal of irrelevant edges can not increase the number of trees, it is easy to establish that the number of trees represents an upper bound on the number of leaves generated by any algorithm of the class A1 for arbitrary k-terminal network reliability problems.

One might expect that the number of leaves generated for the k-terminal problem need not exceed $\tau_k(G)$ = the number of k-trees of the graph G. This is not so, primarily because $\tau_k(G) \neq \tau_k(G^*e) + \tau_k(G-e)$. A k-tree is a tree, all of whose pendant vertices are K-vertices, that connects all vertices in K. Such a tree is a minimal success set. Closely related is what will be called a k_Γ-tree. For some total ordering imposed upon the edges of G, a k_Γ-tree is an acyclic subgraph that connects all the vertices in K but contains no lexicographically smaller subgraph that also connects all the vertices of K. (Lexicographic ordering is analagous to alphabetic ordering; however, rather than using the alphabet one uses some other ordering of basic elements (i.e. edges) to sort objects composed of those elements.) The number of leaves generated by an A1 algorithm for a k-terminal problem need not exceed $\tau_{k_\Gamma}(G)$ = the number of k_Γ-trees of G where this number depends on the ordering imposed on the edges.

Let the vertices of G be labeled $1,2,\cdots,k,k+1,\cdots,n$ where the first k vertices are the K-vertices. Let the m edges of G be sorted lexicographically by their vertex pairs. Let each k_Γ-tree be represented by a lexicographically ordered list of edges. If the τ_{k_Γ} trees are listed lexicographically it is possible to construct a binary search structure where each leaf represents a different tree. This same binary search structure may be used to calculate network reliability.

These ideas can be developed further to show that it is possible to find the reliability from the list of k_r-trees without sorting the trees or constructing a binary search structure.

One constructs this search structure as follows. Divide the list of trees into two parts such that the lexicographic smallest edge that is not in every tree, is in all trees of one part of the list and is in no tree of the other part of the list. This splitting of the list corresponds to a binary branch in the search structure with one part of the list corresponding to the lexicographically smallest non-bridge edge being included (contracted) and the other part of the list corresponding to that same edge being deleted. One applies this procedure recursively, treating each part as a list to be split in two until each list consists of one tree. When this has been achieved, the binary search structure that has an associated branch for every splitting of a list will have a leaf that corresponds to a different k_r-tree. For an example, see Figure 3.1.

Every arc of the search structure corresponds to some edge working or failing and may be labeled with the probability of that event. Each node may be labeled with the probability that all pertinent bridge edges work. Here pertinent bridge edges are those lexicographically smaller than the edge represented by the arcs leaving the node. Each leaf may be labeled with the product of the arc and node probabilities on the path to that leaf. Each leaf represents the event that the set of working and failed edges is such that the k_r-tree corresponding to the leaf is the lexicographical smallest k_r-tree all of whose edges are working. The leaf label represents the probability of this event.

It is possible to generate the binary search structure described above by using a lexicographic edge selection rule in a backtrack algorithm. One simply selects the lexicographical non-bridge edge for branching. After the contraction of some edge (u, v) it is necessary to relabel the new composite vertex with the minimum of u and v and to change the ordering of the edges accordingly. Branching according to this scheme corresponds to the splitting scheme described previously. The arcs and nodes of this search structure may be labeled with the probabilities of the appropriate events and the reliability calculated as the structure is generated. Thus, the following results have been established.

Lemma 3.2 There exists an algorithm A ϵ A1 that can solve $R_k[G]$ while generating exactly $\tau_{k_l}(G)$ leaves.□

Lemma 3.3 There exists an algorithm A ϵ A1 that solves $R_k[G]$ whose leaves are in 1-1 correspondence with the k_l-trees of G.□

It should be noted that for many backtrack algorithms, one may label nodes and arcs appropriately and sum the leaf probabilities to obtain the reliability. This may be done only when the backtrack algorithm generates a partition of the success sets. Moreover, note that the reliability calculation requires work proportional to the size of the search structure and therefore is no more complex than the generation of the structure. Hence, one may adapt Gabow's spanning tree enumeration algorithm to solve the all terminal problem in $O[n\tau]$ time.

Enumeration of the k_l-trees of G is possible by modifying reliability algorithms of the class A1. An enumeration algorithm may be specified as follows. The initial call should be ENUM1$(G,\varnothing)$.

ENUM1(G,S)

1. Remove all irrelevant edges.
2. Perform bridge reductions. B is the set of bridges contracted. Let $S = S \bigcup B$.
3. If G is a vertex then output(S) and return.
4. Use the lexicographic edge selection rule. Let e be the edge selected.
5. ENUM1$(G^{\bullet}e, S \bigcup e)$; ENUM1$(G-e,S)$.

2. Backtrack Algorithms with Parallel Edges Reductions

Let A2 be a class of backtrack algorithms for the k-terminal network reliability problem. The important feature of this class of algorithms is that each set of parallel edges is reduced to a single edge in a reliability preserving network transformation. The sketch that follows describes these algorithms.

REL2(G, w)

1. Remove irrelevant edges.

2. Perform parallel edges reductions.

3. Perform bridge reductions. If B is the set of bridges contracted, set $w = w\prod_{e \in B} p_e$.

4. If G is a vertex then return(w).

5. Select an edge e such that $G^{\bullet}e$ and $G-e$ are connected.

6. Return(REL2($G^{\bullet}e, p_e w$) + REL2($G-e, q_e w$)).

REL2 differs from REL1 in that REL2 performs parallel edges reductions. Of interest is $L2_A(G)$ = the number of leaves in the search structure generated by algorithm $A \in$ A2 when solving for $R_k[G]$. Again, the number of leaves generated when solving the all terminal problem will be considered first. Recall that $\alpha(G)$ is the number of single source acyclic orientations of G.

Theorem 3.4 $L2_A(G) = \alpha(G)$ for all $A \in$ A2 when solving the all terminal network reliability problem.

Proof. Use induction on the number of edges. The inductive hypothesis is that the theorem holds for all graphs with $|E| \leqslant m$. This is certainly true if $|E| = 1$ or G is a tree.

Now consider what happens if G has $m+1$ edges. Note that there will be no irrelevant edges. If a bridge or a parallel edges reduction is possible then the invariance of α to such reductions (Satyanarayana and Procesi-Ciampi[1981], Stanley[1977]) and the fact that the reduced graph has at least one fewer edge gives the desired result. If no reduction is possible, an edge will be selected and two subproblems will be created. The number of leaves generated is clearly the sum of the leaves generated in each of these subproblems, that is, $L2_A(G) = L2_A(G^{\bullet}e) + L2_A(G-e)$. Since neither $G^{\bullet}e$ nor $G-e$ has more than m edges, the inductive hypothesis may be applied to give

$$\alpha(G^{\bullet}e) = L2_A(G^{\bullet}e) \text{ and } \alpha(G-e) = L2_A(G-e).$$

Combining these equations with the acyclic orientation factoring theorem of Chapter 1 yields

$$\alpha(G) = \alpha(G^*e) + \alpha(G-e) = L2_A(G^*e) + L2_A(G-e) = L2_A(G).$$

This proves the theorem. □

It is easy to show that there must exist edges that satisfy the requirements of step 5. All edges that are not bridges are eligible. If all edges of G were bridges then G would be a tree and $L2_A(G) = \alpha(G) = 1$. This edge selection criterion eliminates the possibility of creating disconnected graphs which become deadends in the search structure. If a disconnected graph were to be created then $\alpha(G)$ would underestimate the complexity of the algorithm because it takes work to discover that the graph is disconnected although α for a disconnected graph is zero.

The combinatorial object associated with A2 algorithms is the unique source acyclic orientation. Let $\alpha_k(G)$ be the number of acyclic orientations with the same source and whose source and sinks are all K-vertices. Recall from Chapter 1 that this number is independent of which K-vertex is chosen to be the unique source. Let $A_k = \{A_k^1, A_k^2, \cdots, A_k^{\alpha_k}\}$ represent the set of acyclic orientations of G whose source is vertex 1 and all of whose sinks are K-vertices.

Associate with each A_k^i a permutation of a subset of the vertex labels, P_k^i. These permutations may be described constructively as follows. Mark vertex 1 and let it be the first element of the permutation. Greedily choose the lowest labeled unmarked vertex adjacent to a marked vertex by way of an oriented edge. Mark this vertex and append it to P_k^i. Continue this procedure until all K-vertices have been marked. These permutations represent the lexicographic minimal ordering of vertex labels consistent with the partial ordering imposed by the orientations of the edges. Note that these permutations may be of different lengths and that the last vertex will always be a K-vertex. A feasible permutation is one that may be derived from some acyclic orientation using the greedy method just described.

In the last section, it was shown that an ordered list of the k_f-trees could be used to find $R_k[G]$. In this section, the relationship between enumeration algorithms and network reliability analysis will be furthered by showing how to find $R_k[G]$ from any list of A_k, the acyclic orientations of G.

At least one acyclic orientation must be associated with any set of edges that contains a k-tree. One may partition the set of successful events by the lexicographic minimum feasible permutation that may be associated with the working edges of some successful event. Let F_k^i be the event that P_k^i is feasible and $\bar{F}_k^i$ be its complement. Then

$$R_k[G] = Prob[F_k^1] + Prob[\bar{F}_k^1 F_k^2] + \cdots + Prob[\bar{F}_k^1 \cdots \bar{F}_k^{\alpha_k - 1} F_k^{\alpha_k}]$$

or

$$R_k[G] = \sum_{i=1}^{i=\alpha_k} Prob[P_k^i \text{ is the lexicographic minimal feasible permutation}].$$

Given an acyclic orientation, it is an easy matter to construct the associated permutation and calculate the probability that it is the lexicographic minimal feasible permutation. Let $P_k^i = (v_1, v_2, \cdots, v_l)$. Then Prob[$P_k^i$ is the lexicographic minimal feasible permutation] $= \prod_{j=1}^{i-1}$ (Prob[there exists an edge from at least one of $\{v_1, v_2, \cdots, v_j\}$ to v_{j+1}] Prob[there is no edge from $\{v_1, v_2, \cdots, v_j\}$ to any $v < v_{j+1}$, $v \in V - \{v_1, \cdots, v_j\}$]). One may compute this probability in $O[n^2]$ time. Therefore, the following has been shown.

Lemma 3.5 Given a list of all α_k acyclic orientations with the same K-vertex as the source and only K-vertices as sinks, it is possible to find $R_k[G]$ in $O[n^2\alpha_k]$ additional time.□

An enumeration of the k_f-trees of G in a coded form may be obtained in a manner analogous to computing the reliability. Each permutation represents a set of k_f-trees. When considering the event that there exists at one edge from one of $\{v_1, v_2, \cdots, v_j\}$ to v_{j+1}, one lists each such edge. The coded form enumeration is realized by listing each such set of edges. One derives the individual k_f-trees by taking all trees where one chooses exactly one edge from each of these sets. See Figure 3.2.

It is possible to enumerate the acyclic orientations in A_k using a backtrack algorithm. This can be done by listing the acyclic orientations lexicographically and demonstrating that one can find an underlying binary structure or by using an algorithm like the following.

ENUM2(G,S)

1. Remove all irrelevant edges.

2. Perform parallel edges reductions.

3. If the edge $(1,u)$ is the only edge from vertex 1, then contract this edge and let S = concatenate(S,u).

4. If G is a vertex then output(S) and return.

5. Select the lexicographic minimal edge $e=(1,v)$.

6. ENUM2$(G^*e$,concatenate$(S,v))$; ENUM2$(G-e,S)$.

The first call should be ENUM2$(G,1)$. The function concatenate adds the second argument to the end of the first. The output is a list of permutations of vertex labels. To derive the acyclic orientations, one simply orients the edges in a manner consistent with the total ordering imposed by the permutation.

Using the fact that the number of permutations beginning with the same label is less than or equal to $(n$-1)! or that $\alpha_k(G) \leqslant (n-1)!$, one obtains the result that A2 algorithms generate no more than $(n$-1)! leaves. A stronger result is that

$$\alpha_k(G) \leqslant \alpha(G) \leqslant (n-1)!$$

which implies that k-terminal problems are never harder than all terminal problems on the same network when algorithms of this class are used (Procesi-Ciampi[1981]). Others have presented similar results for the all terminal network reliability problem. Murchland and Shier[1973] developed an algorithm that generates a search structure by including and excluding edges as is done here. He also discussed series-parallel reductions. Later, Buzacott[1976] showed that algorithm of Murchland and Shier has $(n$-1)! leaves when a certain edge selection strategy is used. Ball[1977,1979] has developed an algorithm with the same complexity that implicitly performs parallel edges reductions.

3. Backtrack Algorithms with Degree Two Vertex Reductions

The following sketch describes A3, a class of backtrack algorithms for the k-terminal network reliability problem. Algorithms in this class perform degree two vertex reductions as well as parallel edges reductions. It is assumed that graphs input to these algorithms are simple (no

parallel edges) and biconnected.

REL3(G, w)

1. Perform parallel edges reductions and degree two vertex reductions. Update w.

2. If G is an edge f then return($p_f w$).

3. Select an edge e such that both G^*e *and* $G-e$ are biconnected.

4. Return(REL3($G^*e, p_e w$) + REL3($G-e, q_e w$)).

The reductions in step 1 may be performed in $O[n^2]$ time. Step 2 may be performed in constant time. In the all terminal problem, step 3 may be performed in $O[m+n]$ time using a variant of a triconnected component decomposition algorithm or in $O[n]$ time using a lexicographic edge selection rule. The $O[m+n]$ method gives all possible edges for branching and allows the option of using this information to help select edges at future nodes of the search structure with little additional work. In the k-terminal case, one must modify step 3 to include the phrase "if possible." It may not be possible to select an edge that meets these conditions. It may be that every edge is adjacent to an irreducible degree two vertex in which case the deletion of any edge creates a separable graph. If it is possible to meet the conditions of step 3, it should be possible to find the edge in $O[m+n]$ time.

Of interest is $L3_A(G)$ = the number of leaves in the search structure generated by algorithm $A \in$ A3 when solving for $R_k[G]$. It is necessary to distinguish the all terminal problem and the k-terminal problem in analyzing this class of algorithms. The first part of the following discussion is devoted to the all terminal case. Note that the existence of a degree two vertex that cannot be reduced is possible only in the k-terminal problem and not in the all terminal problem.

The All Terminal Network Reliability Problem for A3 Algorithms

Theorem 3.6. $L3_A(G) = \beta(G)$ for *all* $A \in$ A3 when solving the all terminal problem.

Proof. Use induction on the number of edges. The inductive hypothesis is that the theorem

holds for all graphs with $|E| \leq m$. This is certainly true if $|E| = 1$.

Now consider what happens if G has $m+1$ edges. If a parallel edges reduction or a degree two vertex reduction is possible then the invariance of β to such reductions and the fact that the reduced graph has at least one fewer edge gives the desired result. If no reduction is possible, and edge will be selected and two subproblems created. The number of leaves generated is the sum of the leaves generated by these subproblems, that is,

$$L3_A(G) = L3_A(G^*e) + L3_A(G-e).$$

Since neither G^*e nor $G-e$ has more than m edges, the inductive hypothesis may be used to give

$$\beta(G^*e) = L3_A(G^*e) \text{ and } \beta(G-e) = L3_A(G-e).$$

Recall from Chapter 1 that a factoring theorem which states $\beta(G) = \beta(G^*e) + \beta(G-e)$ holds for $\beta(G)$. Combining these equations yields

$$\beta(G) = \beta(G^*e) + \beta(G-e) = L3_A(G^*e) + L3_A(G-e) = L3_A(G).$$

This proves the theorem. □

Note that $\beta(G)$ may grossly underestimate the number of leaves generated if the edges selected do not satisfy the conditions of step 3. This is because if the edge selected does not satisfy those conditions a separable graph will be created. A finite amount of work is necessary to resolve the separable graph although β will be zero. A proof that proper edges exist and a characterization of those edges is presented next.

Showing that it is possible to select proper edges for algorithms of the class A3 is done using results from the triconnected decomposition of graphs as discussed by Hopcroft and Tarjan[1973]. A number of definitions and lemmas are presented to show that there exist edges e such that both G^*e and $G-e$ are biconnected.

Let a, b be a pair of vertices in a simple biconnected graph G. Suppose that the edges of G are divided into equivalence classes $E_1, E_2, \cdots, E_c$ such that two edges which lie on a common path not containing any vertex of a, b except as endpoints are in the same class. The classes E_i are called separation classes of G with respect to a, b. If there are at least two

separation classes then a,b is called a separation pair unless there are exactly two separation classes and one class is a single edge.

A graph is triconnected if and only if it is biconnected and has no separation pair.

A graph may be split with respect to a separation pair into two split graphs G^1 and G^2 such that $E^1 = \bigcup_{i=1}^{k} E_i$, $E^2 = E - E^1$, $|E^1| \geqslant |E^2|$, $G^1 = (V(E^1), E^1 \bigcup (a,b))$, and $G^2 = (V(E^2), E^2 \bigcup (a,b))$ where (a,b) is a virtual edge. The way in which G^1 is chosen insures that $|E^1| \geqslant 2$.

Lemma 3.7 If $G = (V,E)$ is simple and biconnected with all vertices having degree at least three then at least one of the two split graphs has at least five edges.

Proof. By constraining $|E^1| \geqslant |E^2| \geqslant 1$ and recalling the conditions for a vertex pair to be a separation pair it is easy to see that $|E^1| \geqslant 2$.

If $|E^1| = 2$ then $|V(E^1)| = 1$ and this single vertex will have degree two which is contrary to assumption. It is therefore necesssary that $|V(E^1)| > 1$. If $|V(E^1)| \geqslant 2$ then $|E^1| \geqslant 5$ if all vertices have degree at least three and G is simple. □

When G has been split until no more splits are possible, each split graph is called a split component. Lemma 3.7 may be applied inductively to obtain the result that at least one split graph has at least five edges. Virtual edges are considered part of each split component. A split component is either a triconnected graph, a triangle, or a triple bond (three parallel edges).

Lemma 3.8 If G is simple and biconnected with each vertex having degree at least three then there will exist a split component with at least five edges.

Proof. At each splitting there will always be at least one split graph with at least five edges that will also be simple and biconnected with all vertices having degree at least three. When no more splitting is possible such a split graph will become a split component. □

Lemma 3.9 If G (V,E) is triconnected then both $G^{\bullet}e$ and $G-e$ are biconnected for all edges

$e \in E$.

Proof. G is triconnected iff there exist three disjoint paths for all (u, v) vertex pairs. Since the contraction of any edge e can at most join two of these paths, two disjoint u, v paths will remain. Deleting any edge e will destroy at most one path which also leaves at least two disjoint u, v paths. This proves the claim since any graph with at least two disjoint paths between any vertex pair is biconnected. □

Theorem 3.10 If $G = (V,E)$ is simple and biconnected with all vertices having degree at least three then there exist edges $e \in E$ such that both $G^{\bullet}e$ and $G-e$ are biconnected. Moreover, if e is an edge (not a virtual edge) from a split component with at least five edges then both $G^{\bullet}e$ and $G-e$ will be biconnected.

Proof. The proof will be based on showing that edges belonging to split components with at least five edges satisfy the theorem. The existence of such edges has already been proven in Lemma 3.8. It is necessary to show that there exist two disjoint paths for all (u, v) vertex pairs in both $G^{\bullet}e$ and $G-e$.

If neither u nor v belongs to the split component to which e belongs then there will be two disjoint u, v paths in both $G^{\bullet}e$ and $G-e$. There must exist two disjoint u, v paths in G because G is biconnected. If neither path involves the split component to which e belongs then the same two paths may be used in both $G^{\bullet}e$ and $G-e$. If either of these paths uses the split component note that it must contain both vertices of the separation pair. The fact that the paths are disjoint implies that at most one of the paths will use edges in the split component to which e belongs. Clearly a path will remain between the vertices of the separation pair after both contraction and deletion of e guaranteeing that there will be two disjoint u, v paths.

If only one of u and v (say u) belongs to the split component to which e belongs, use the fact that there always exists a path containing any given three vertices in a biconnected graph. Let $\{a, b\}$ be a separation pair such that u and v are in different equivalence classes with respect to $\{u, v\}$. After both contraction and deletion, the split component to which e and

vertex u belong will be biconnected and there will be a path with a and b as endpoints that contains u. Some other path containing v with a and b as endpoints will exist in G, G^*e, and $G-e$. Combining these two paths insures that there will be two disjoint u,v paths.

If both u and v belong to the split component that contains e, recall that this component is triconnected and use Lemma 3.9. □

An algorithm of the class A3 may be used to enumerate all source-sink acyclic orientations of the graph as follows. Label the source vertex 1 and label the sink vertex n. Call ENUM3(G,{1}).

ENUM3(G,S)

1. Perform parallel edges reductions.

2. If vertex 1 is of degree two and is adjacent to vertices u and v with $u < v$ then perform the degree two vertex reduction, relabel u to 1, and let S = concatenate(S,u).

3. If G is a single edge $(1,w)$ then output(concatenate(S,w)) and return.

4. Select the lexicographic minimal edge $e=(1,x)$.

5. ENUM3(G^*e,concatenate(S,x)); ENUM3($G-e,S$).

The output will be permutations. One derives the acyclic orientations by orienting the edges in a manner that is consistent with the total ordering of the permutation.

The k-Terminal Network Reliability Problem for A3 Algorithms

The analysis of A3 algorithms applied to k-terminal problems is more difficult than the analysis of the all terminal case. Because it is not always possible to perform degree two vertex reductions, there is no equivalence between $\beta(G)$ and the number of leaves generated. This is easily seen by considering the case of the Wheatstone bridge wherein the degree two vertices are the K-vertices in a two terminal reliability problem. Although $\beta(G) = 1$, class A3 algorithms must generate two leaves in finding the solution.

The lemmas of the first part of this discussion do not apply because there can be degree two vertices. An irreducible biconnected network may be composed of triconnected components all of which have two or three edges in which case it may not be possible to branch without creating separable graphs. For these reasons, the analysis done in the all terminal case no longer is valid.

Simple biconnected graphs in which all possible degree two vertex reductions have been performed cannot have triconnected components with exactly four edges. If such a graph has no triconnected components with five or more edges, there are unification (backtrack fusion) techniques that aid in solving the problem. Observe that these graphs are composed of chains of two or three edges. The endpoints of these chains are separation pair vertices and the interior vertices are irreducible degree two K-vertices. Some of these structures may be reduced using techniques described by Satyanarayana and Wood[1982]. There may also be single edges joining separation pair vertices.

If the graph consists only of two edge chains, consider a set of l two edge chains between vertex i and vertex j, a separation pair of non-K-vertices. Let the interior vertices be labeled 1 through l. The entire set of edges may be reduced to one or less edges with three possible designations of vertices i and j: either i is a K-vertex or j is a K-vertex or both are. This result is obtained by considering the effects of branching on edges of the form $e=(i,l_x)$.

If one contracts e then i becomes a K-vertex adjacent to j via the edge (l_x,j). If e is deleted then vertex j becomes a K-vertex after a bridge (pendant) reduction of the edge (l_x,j). If edges $(i,1),(i,2),\cdots,(i,l)$ all are contracted then i becomes a K-vertex adjacent to the non-K-vertex j after parallel edges reductions. If edges $(i,1),(i,2),\cdots,(i,l)$ all are deleted then j becomes a K-vertex not adjacent to i. In all other cases (some edge (i,l_x) is deleted and some edge (i,l_y) is contracted), both i and j will be K-vertices. If this happens it will be possible to perform all degree two vertex reductions. (See Theorem 2.9.) An additional parallel edges reduction leaves a graph with a single edge between the two K-vertices i and j. This is represented by the search structure in Figure 3.3.

At each leaf there is at most one edge (i,j) as well as the probability of reaching that leaf. One may unify problems with the same topology by taking the sum of the edges' working (failing) probabilities weighted by the probability of reaching the leaf that represents the graph that contains the edge. To obtain the three graphs (with the correct edge probabilities) for the three cases requires $O[n^2]$ work.

If s is the number of separation pair vertices, then one need consider at most 2^s different cases of each vertex being a K-vertex or not. For each case, one chooses the appropriate edge between every separation pair and solves for the reliability. Summing the reliability over all cases gives the reliability of the original network.

The overhead in implementing the above unification techniques is likely to outweigh any actual computational savings. If one does not implement such a scheme, the amount of work done by the algorithm is bounded below by $\beta(G)$ and bounded above by $\alpha_k(G)$. The lower bound is achieved in the all terminal problem and in the two terminal problem where the two vertices are either adjacent or members of the same separation pair.

4. Backtrack Algorithms with Wheatstone Bridge Reductions

REL4 describes A4, a class of algorithms for the k-terminal network reliability problem. Algorithms in this class perform Wheatstone bridge reductions as well as parallel edges reductions and degree two vertex reductions. It is assumed that graphs input to these algorithms are simple and biconnected.

REL4(G, w)

1. Perform parallel edges reductions, degree two vertex reductions, and Wheatstone bridge reductions updating w appropriately.
2. If G is a single edge f then return(wp_f).
3. Select an edge e such that both G^*e and $G-e$ are biconnected.
4. Return(REL4$(G^*e, p_e w)$ + REL4$(G-e, q_e w)$).

The reductions in step 1 are interrelated and cannot be done independently. A

Wheatstone bridge reduction may lead to the creation of parallel edges and/or degree two vertices which may allow additional reductions. Additional research is needed to find the best way to do all these reductions although it is possible to perform these reductions in $O[n^2]$ time. Step 2 may be performed in constant time. In the all terminal problem, step 3 may be performed in $O[m+n]$ time using a variant of a triconnected component decomposition algorithm or in $O[n]$ time using a lexicographic edge selection rule. The $O[m+n]$ method may be preferable since it gives information helpful in the location and creation of Wheatstone bridges. In the k-terminal case, one must modify step 3 to include the phrase "if possible." If it is possible to meet the conditions of step 3, it should be possible to find the edge in $O[m+n]$ time.

The edge selection strategy specified in step 3 does not give sufficient conditions for generating an optimal search structure even in the all terminal case. This is easily demonstrated by showing that the lexicographic selection rule is not optimal. See Figure 3.4. Even less is known about optimal edge selection rules for the k-terminal problem.

Bounds are found for the number of leaves generated by A4 algorithms applied to all terminal problems. These bounds are derived by using a particular edge selection strategy when solving the all terminal problem on complete graphs. By performing a suitable sequence of deletions from K_n to obtain G and considering the search structure that develops one can show that $L4_A(G) \leqslant L4_A(K_n)$.

Theorem 3.11 There exists an algorithm $A \in$ A4 such that $L4_A(G) \leqslant \frac{11}{24}(n-2)!$ for the all terminal problem.

Proof. Using a lexicographic edge selection rule, degree two vertex reductions, and parallel edges reductions, one can solve a complete graph on n vertices by generating a search structure with exactly $\frac{(n-2)!}{4!}$ nodes that represent the graph K_6. Using the edge selection strategy as per Figure 3.4, each K_6 may be solved generating 11 leaves. □

5. Backtrack Algorithms with Triconnected Component Reductions

Let A5 be a class of algorithms for the k-terminal network reliability problem that incorporates triconnected component reductions into the framework of A3 algorithms. These algorithms generalize A4 algorithms in that all triconnected components, not just Wheatstone bridges, are considered for reduction. Predictably, the description of this class of algorithms by REL5 differs little from REL4.

REL5(G, w)

1. Perform parallel edges reductions, degree two vertex reductions, and triconnected component reductions updating w appropriately.
2. If G is a single edge f then return(wp_f).
3. Select an edge e such that both G^*e and $G-e$ are biconnected.
4. Return(REL5$(G^*e, p_e w)$ + REL5$(G-e, q_e w)$).

The reductions in step 1 are interrelated and cannot be done independently. A triconnected component reduction may lead to the creation of parallel edges and/or degree two vertices which may allow additional reductions. Additional research is needed to find the best way to do all these reductions and to determine if all methods produce a unique irreducible graph. Step 2 may be performed in constant time. In the all terminal problem, step 3 may be performed in $O[m+n]$ time using a variant of a triconnected component decomposition algorithm or in $O[n]$ time using a lexicographic edge selection rule. In the k-terminal case, one must modify step 3 to include the phrase "if possible." If it is possible to meet the conditions of step 3, it should be possible to find the edge in $O[m+n]$ time.

Since the optimal edge selection strategy is not known, it is not possible to determine the number of leaves generated by algorithms of this class although its worst case bound is certainly no worse than that of A4 algorithms. Neither is it known what combinatorial objects can be associated with the leaves generated by these algorithms. It is not obvious that one should select edges in such a way as to create additional triconnected components although it is clear that these reductions should be performed if reducible triconnected components exist.

I speculate that it is best to use edge selection rules that tend to create dense graphs or degree two vertices rather than to try to create additional large triconnected components. If the graph is fairly dense initially, I further speculate that it is best to use some edge selection rule (i.e. lexicographic) that minimizes the number of triconnected components with five or more edges.

6. Remarks

Five classes of backtrack algorithms were presented along with some analyses of their complexity. The classification scheme was posed to help highlight the essential differences among these algorithms. In the next chapter computational experience with algorithms from these classes is described.

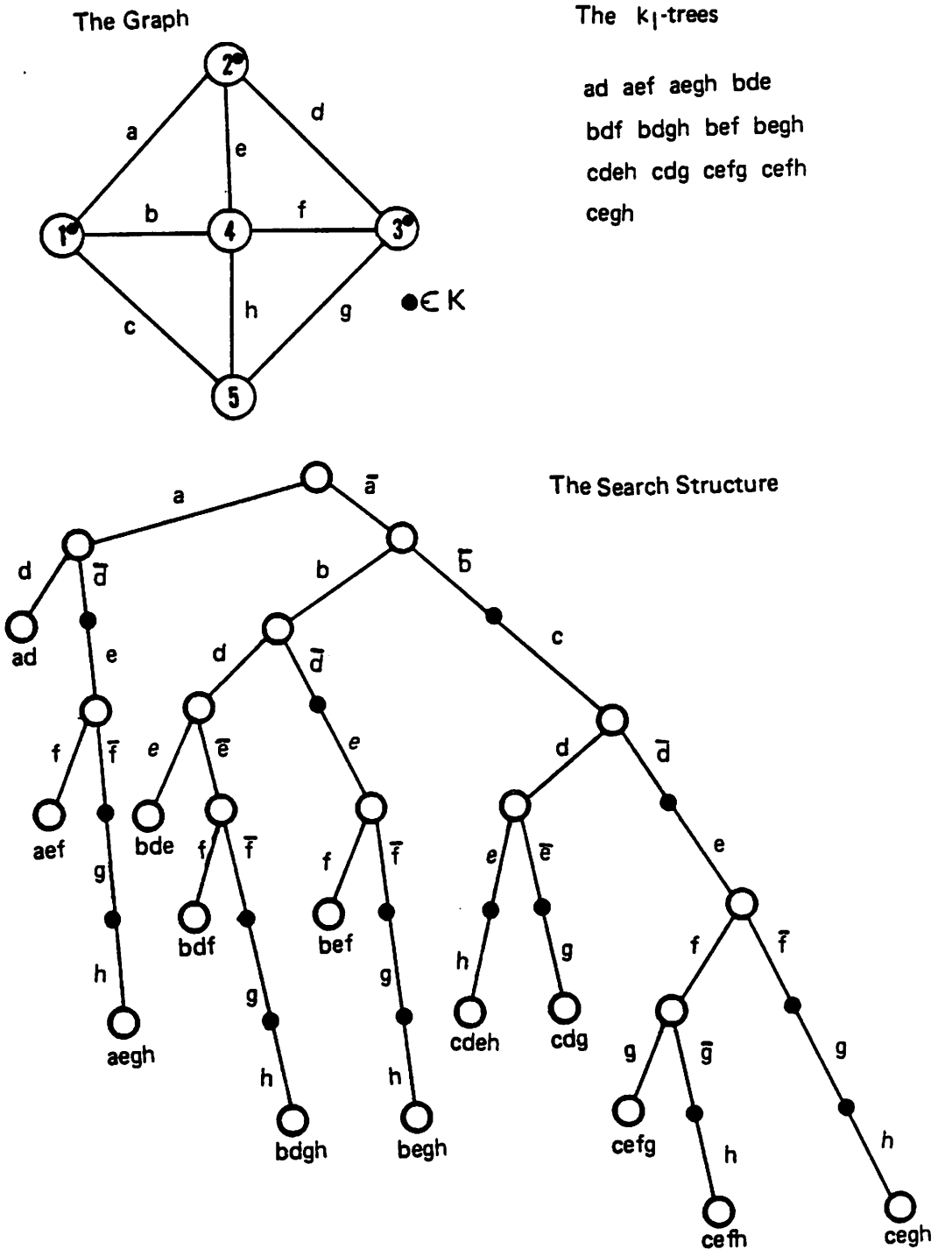


Figure 3.1 Ordered List of k_1-trees and Associated Search Structure

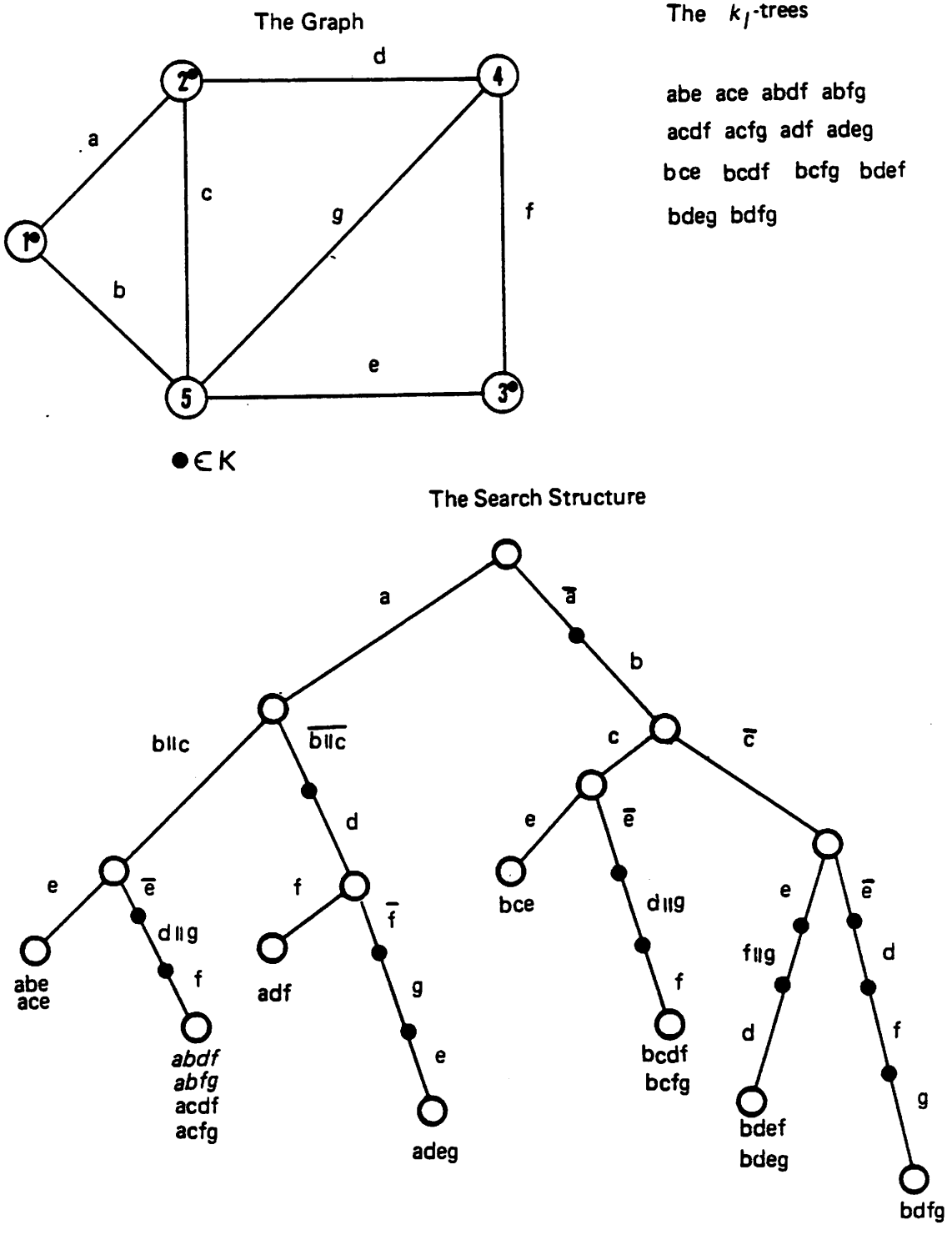


Figure 3.2 Coded Enumeration of k_l-trees

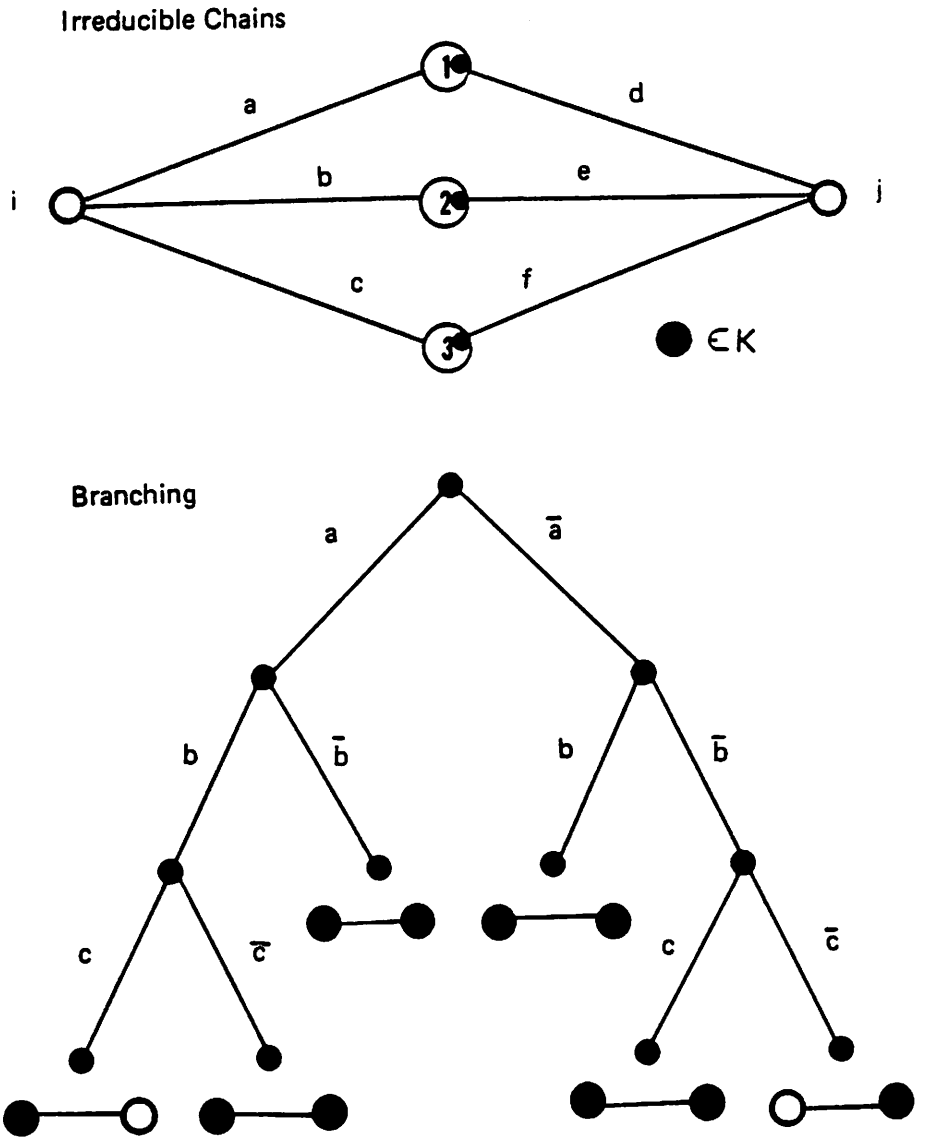


Figure 3.3 Irreducible Chains and Branching

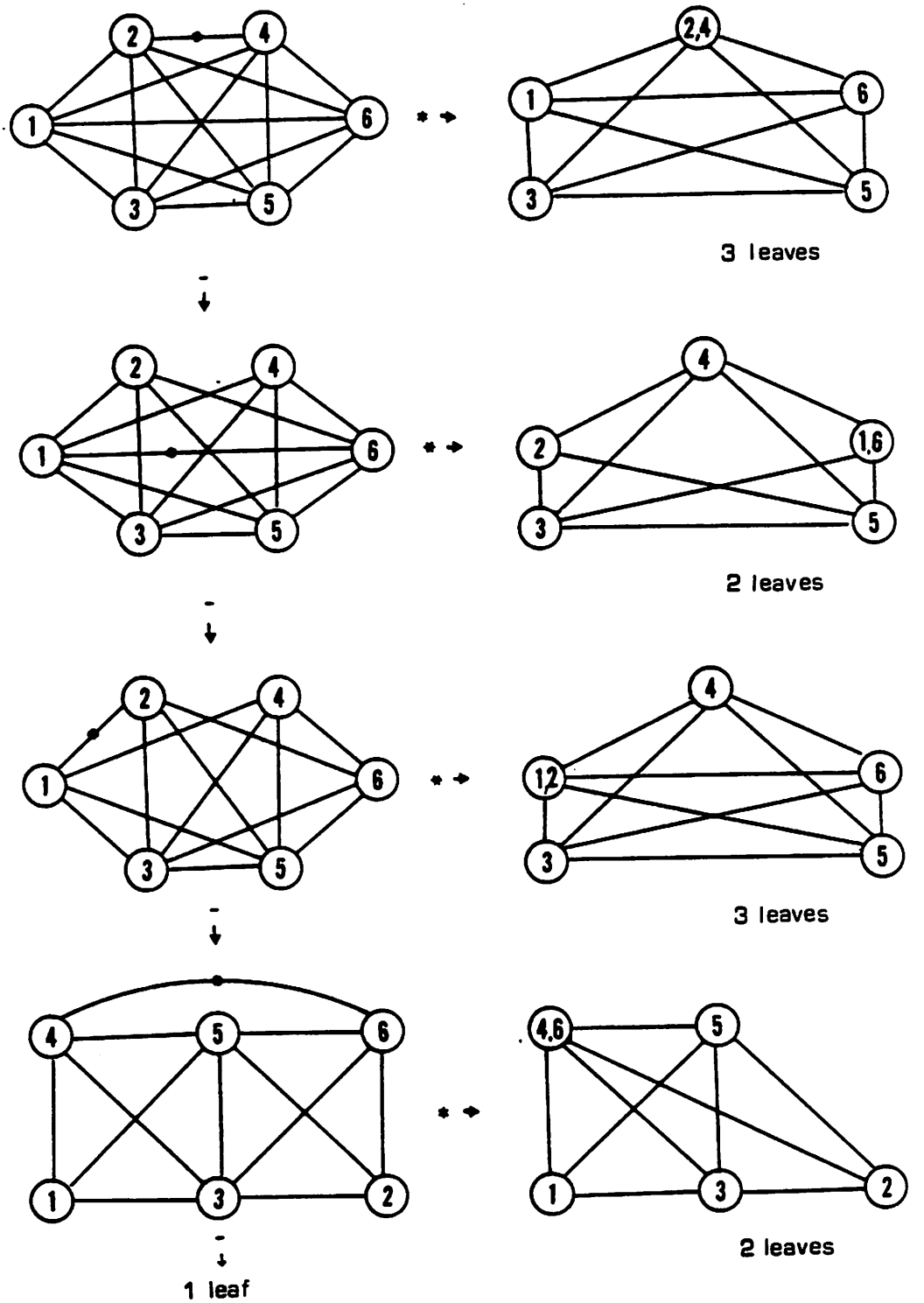


Figure 3.4 Non-Lexicographic Rule is Optimal

Chapter Four: Computational Experience

This chapter describes computational experience gained with an experimental program developed for solving k-terminal network reliability problems. The data structures used to implement the algorithms are described. Different edge selection strategies that were implemented are described. Examples with answers, run times, and various statistics about the program's operation are given. These examples are used to highlight certain insights gained about the operation of the various algorithms when applied to different kinds of networks. Suggestions for improved implementations are offered.

1. Data Structures

The network data structure was implemented as a set of doubly linked edge lists (one for each vertex) wherein each undirected edge is represented as a pair of antiparallel directed edges. Each edge appears on two edge lists. For any such directed edge the field *link* contains the address of the associated antiparallel edge. The head and tail vertices of each edge are stored in the arrays *head* and *tail*. The addresses of the previous edge and the next edge are stored as *prev* and *next*. These five arrays (*link*, *head*, *tail*, *prev*, and *next*) contain the information necessary to describe the graph and are indexed by *gadr* (graph address).

Associated with each edge of the graph is an edge label (*elbl*). The same edge label is associated with each edge of an antiparallel edge pair. This label is used to index the array p which contains the working probability of each edge. The arrays *elbl* and *gadr* allow one to find the edge label associated with a graph address and vice versa.

More information is associated with the network data structure. It is assumed that graph addresses $1, \cdots, n$ of *head* are reserved to be the heads of the edge lists for each vertex.

Graph address 0 heads the list of unused memory locations. By convention, if the *next* and *prev* associated with the head of each list are equal to the address of the list head, then the list is empty. In *head*[*i*] is the degree of vertex *i*. In *tail*[*i*] is the index of the lowest labeled vertex to which vertex *i* is connected by working edges. Vertices become connected after edges are contracted or after certain reductions. For $i > n$ these arrays store the vertices associated with an edge as was described. See Figure 4.1.

Other useful information includes the number of K-vertices, the number of vertices, the number of edges, and the weight associated with each graph. It is important to note that the vertices labeled 1 to $|K|$ are the K-vertices. Most of these variables were global and continually available to all routines. Global variables were also used to store the internal statistics collected. They are described in the section where the computational results are discussed.

2. Edge Selection Strategies

The optimal edge selection strategies often dictated that one perform complicated graph algorithms to find a good edge on which to branch. Practical considerations ruled out time consuming edge selections at every call of the routine. Efforts were made to find fast edge selection strategies that were optimal or nearly so.

The edge selection rules that were implemented are as follows.

1. An edge adjacent to the vertex labeled 1 was chosen. The edge chosen is simply the first one on the edge list of vertex 1. If the edges are originally entered in lexicographic order then this rule is the lexicographic edge selection rule. In the two terminal case, it is almost always non-optimal since the first edge it contracts (edge (1,2)) creates irrelevant edges. The labeling scheme and reductions assure that vertex 1 will always be a non-isolated K-vertex so there will always be an edge adjacent to the vertex labeled 1. This rule takes constant time.

2. The first edge on the list of a random K-vertex is chosen. This rule takes constant time.

3. The first edge on the edge list of the lowest labeled vertex of maximum degree is chosen. This rule requires $O[n]$ time.

4. The first edge on the edge list of the lowest labeled vertex of minimum degree is chosen. This rule tends to create a degree two vertex. If that vertex is irreducible, this rule will change such a vertex to a pendant. This rule requires $O[n]$ time.

5. This is an interactive edge selection rule with prompting. At each call of the routine, a list of the edges is printed and the user of the program is requested to select an edge.

6. The first edge on the edge list of the lowest labeled K-vertex of maximum degree is chosen. This rule differs from ESR3 in that it does not concern itself with non-K-vertices. Unlike ESR3, at least one of the vertices of the edge selected will be in K. This rule requires $O[k]$ time.

7. The first edge on the edge list of the lowest labeled K-vertex of minimum degree is chosen. This rule requires $O[k]$ time.

8. An edge of the form $(1, u)$ is chosen where u is maximized. Since K-vertices have the lowest labels, this rule tends not to cause all K-vertices to be coalesced early in the branching process. This rule avoids the problem of creating irrelevant components early in the search structure as is done by ESR1. This rule requires $O[n]$ time.

9. This rule selects an edge $(1, u)$ where u is the lowest labeled vertex adjacent to vertex 1. This is an $O[n]$ lexicographic edge selection rule. This rule requires $O[n]$ time.

10. This rule allows the user to select the edges chosen although there is no prompting. This rule is useful after having used ESR5. One can select those same edges and still be able to have the run statistics printed without seeing an edge list at every call of the procedure.

11. An edge (u, v) is selected where u is the lowest labeled K-vertex of maximum degree and v is the lowest labeled vertex of maximum degree on the edge list of u. This edge selection rule requires $O[n]$ time.

12. An edge (u, v) is selected where u is the lowest labeled K-vertex of minimum degree and v is the lowest labeled vertex of minimum degree on the edge list of u. This edge selection rule requires $O[n]$ time. It is felt that this rule would tend to create more degree two vertices than ESR11.

3. Discussion of Results

Implementations of algorithms of classes A1, A2, A3, and A4 were used to solve problems whose underlying graphs were complete graphs, cubic graphs, quartic graphs, and others. At the end of the chapter is a set of figures containing examples of problems solved. Each figure includes a description of the network analyzed and a set of problem instances that were solved. The solution of each instance required a run of some algorithm. A run is characterized by three parameters: the class of algorithm run (AC); the number of vertices in K (k: the vertices $1, \cdots, k$ are in K); and the edge selection rule used (esr). The output from each run includes the answer to the reliability question (mrp[k]) as well as internal statistics generated by the program and the amount of computer time used to solve each instance.

The various statistics collected were used to help verify algorithm correctness and to monitor and compare the performances of different algorithms. These statistics were output under various columns. The column titled "mrp[k]" contains the multi-terminal network reliability probability for the k vertices of concern. The next two columns, *calls* and *leaves*, record the number of calls to the main recursive routine and the number of good leaves generated in the solution of the problem instance. There are deadends in the search structure if the number of leaves is as few as half the number of calls. The column *cpusec* indicates the number of seconds the central processing unit of the computer used in a run. These times are to the nearest $\frac{1}{60}$ second and do not include the small amount of time used to input the data. The last four columns contain information describing the number of times each of the various reductions was successfully performed. Counted are the number of degree one vertex reductions(dors), parallel edges reductions(pers), degree two vertex reductions(dtrs), and Wheatstone bridge

reductions(wbrs).

Complete graphs were chosen as one class of graphs to analyze because of the possibility of verifying that the right number of leaves was generated. For complete graphs it is known that $\beta(K_n) = (n-2)!$, $\alpha(K_n) = (n-1)!$, and $\tau(K_n) = n^{n-2}$. If an optimal edge selection strategy is used, a correct algorithm should generate a number of leaves equal to the appropriate graph invariant.

Some cubic and quartic graphs were analyzed as representatives of sparse graphs. By counting the number of trees it was possible to determine the best possible performance to expect from an A1 algorithm for the all terminal problem. It was not clear how close the other algorithms were to optimum or to determine a priori how many leaves to expect since the best ways to calculate $\alpha(G)$ or $\beta(G)$ involve work proportional to solving the related reliability problem.

Comparing the number of leaves generated by A1 algorithms and the number of trees of the graph was easily done. For the complete graphs K_6 (Figure 4.3) and K_7 (Figure 4.4), it was shown that the number of leaves equaled the number of trees. For other graphs (i.e. Figure 4.7), the matrix tree theorem was used to find the number of trees and again it was seen that the number of trees was equal to the number of leaves generated, irrespective of the edge selection strategy. One can also see that the number of leaves generated while solving k-terminal problems is less than the number of leaves generated while solving the all terminal problem on the same network. Small examples done by hand demonstrate that the number of leaves generated is the number of k_f-trees; it is harder to count the number of these trees in larger networks. Algorithms from this class were not run on most of the larger problems because A1 algorithms are very slow.

Runs of A2 algorithms also corresponded to theoretical results. Complete graphs were solved with $(n-1)!$ leaves. It was shown that it is possible to generate as few as $(k-1)(n-2)!$ leaves when solving k-terminal problems on complete graphs(see Chang[1981]). For small graphs, it could be seen that these algorithms generated a number of leaves equal to the

number of acyclic orientations with all sinks in K.

The degree two vertex reduction is what distiguishes class A3 algorithms from those of class A1 and A2. This reduction significantly decreases the amount of computer time used in solving various problem instances. The reasons for this are twofold: first, fewer calls are made and fewer leaves are generated when using this reduction and secondly, it is no longer necessary to use the time consuming routine that finds bridges since no bridges are created when using the optimal edge selection strategy for A3 algorithms. For both complete graphs and the two classes of sparse graphs examined, A3 algorithms were faster than A2 algorithms by multiplicative factors ranging from n to $2n$. See Figures 4.4, 4.7, 4.8, 4.10, and 4.11.

For A3 algorithms, it is shown that it is possible to solve the all terminal or the two terminal problem on a complete graph, K_n, while generating as few as $(n-2)!$ leaves. In the runs done, this result was not always obtained because the edge selection rules implemented were not always optimal. See Figure 4.2. One also notices that the number of leaves generated while solving problems on the same graph tend to increase as the absolute value of the difference between the number of K-vertices and the number of vertices not in K decreases. See Figure 4.9. Theorem 2.9 suggests that this should be true. When the vertices are neither mostly in K nor mostly not in K, one is least likely to be able to perform triconnected component reductions of which degree two vertex reductions are a special case.

Exhaustive case analysis on all graphs smaller than K_6 was used to determine the best edge selection strategy for A4 algorithms on small graphs. One can see that it is not possible to generate fewer than eleven leaves while solving the all terminal problem although 11 is achievable. The strategies that generate this minimal search structure seemed to defy easy characterization that would have allowed a fast edge selection rule to be written. For most of the runs, a suboptimal strategy that solved a K_6 in twelve leaves was used. For this reason, the A4 algorithms applied to complete graphs generated $\frac{1}{2}(n-2)!$ leaves (and not $\frac{11}{24}(n-2)!$ leaves) while solving the all terminal problem. See Figure 4.6. (Figure 4.3 contains a run in which a K_6 was solved while generating only 11 leaves using ESR10.) In general, A4 algorithms took about

half as much time as A3 algorithms solving the same problem.

The optimal edge selection strategy for A4 algorithms is not yet known although it seems clear that it should be more restrictive than the optimal strategy for A3 algorithms. The A3 strategy is to select edges that belong to triconnected components with at least five edges. Not selecting such edges seems to increase the size of the A4 search structure although one could follow this rule and not produce the minimal search structure. Empirical evidence seems to indicate a number of properties of good edge selection rules for A4 algorithms. As is evidenced by the consistently good performance of edge selection rule 8 (ESR8), it seems likely that one should try to create graphs with vertices either mostly in K or mostly not in K. ESR8 tends to do this by branching on edges of the form $(1, u)$ where u is maximized. Since K-vertices have the lowest labels each contraction decreases the number of vertices not in K (unless all were in K already) while each deletion helps make u a reducible degree two vertex. It might be even better to try and select edges adjacent to no K-vertex.

Other edge selection rules that were tested included selecting edges between vertices of large degree (ESR11) and selecting edges between vertices of small degree (ESR12). See Figures 4.7, 4.8, and 4.10. It was thought that selecting edges between vertices of large degree would tend to create dense graphs and graphs likely to have large triconnected components. The other strategy seemed more likely to create degree two vertices and less likely to create larger triconnected components. Comparing runs that used these different strategies supports the speculation that it is better to try to select edges in such a way as to create reducible degree two vertices than to try to select edges so as to create Wheatstone bridges or larger triconnected components.

After noting that this program requires roughly six minutes of computer time to solve a complete graph on ten vertices and fractions of a second to solve graphs smaller and sparser, little attention should be paid to the times. No claim is made that the most efficient program has been implemented. More attention should be paid to the internal statistics generated than to the times.

4. Suggestions for Better Programs

The backtrack algorithms described in the previous chapter offer little hope to those who wish analytic solutions to reliability problems on large or dense networks. This is clear from the complexity analyses of Chapter Three and the computational experience described in this chapter. Although one should not be too optimistic about prospects for good (or even drastically better) algorithms and programs, it is possible to implement programs much like the one described that are more likely to be able to handle problems on large and dense networks.

More flexibility would go a long way in improving this kind of backtrack program. Such flexibility is achieved by a modular design and is useful to the researcher. Independent modules allows for ease in testing ideas and routines. This is helpful in proving program correctness and in checking the performance of different routines and algorithms on various types of graphs. If one is designing a program to solve practical problems, it is necessary to do more than just use routines with the best asymptotic worst case bounds; one must also consider heuristics and try to find those routines that tend to perform best.

Changes in the implementation of the reduction techniques could cause the program to run faster. Efficient routines must be used to implement each reduction. Data structures should be (re)designed to make frequently performed operations easy. For example, it might be worthwhile to maintain a list of three neighboring vertices for each vertex so that it will be easier to search for Wheatstone bridges. It may be necessary to first solve a large number of problems in order to determine which operations are used most frequently. It was found that some computational effort could be saved by not invoking the Wheatstone bridge reduction routine unless the network had six or fewer vertices since with most edge selection rules such reductions are extremely rare if the graph has more than six vertices.

The idea of implementing adaptive algorithms to perform reductions can be extended. If the degree two vertex reduction routine knew which edge selection strategy was used, then it might be possible to perform this reduction in constant time since some strategies prevent all but one vertex from ever being degree two. Making routines adaptive should help in the

solution of larger problems.

More versatile edge selection strategies would be useful in both research and in solving larger problems. It should be possible to select edges based on functions depending on the degrees of both vertices, the characteristics of the network, and even perhaps the working probability of the edge. One should be able to vary the rule as a function of the network's characteristics. Although such a strategy would be complicated, it would allow more experimentation and perhaps suggest what types of rules work best with various kinds of graphs. Ultimately, such experimentation would lead to the creation of heuristics that allow one to solve problems more effectively.

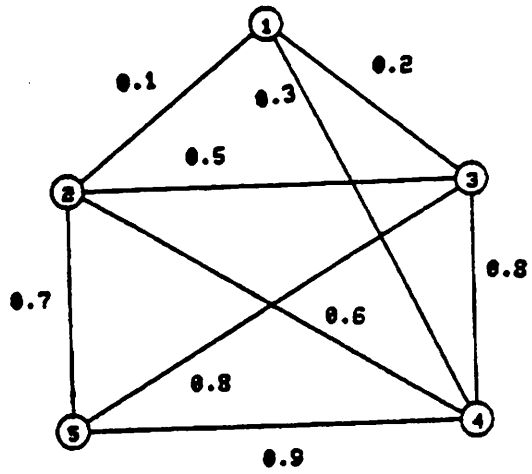
Although only one program was used to generate the results presented, other programs were implemented. One of these early programs used very simple data structures and performed only parallel edges reductions. The lack of overhead allowed this small program to outperform the more sophisticated one for small graphs (K_6 and smaller). It should be possible to extend the range of the program by incorporating the simple version of the reliability algorithm into the more sophisticated version and invoking the simpler routine when the problem or subproblem to be solved is small.

If one expects a program to be useful for arbitrary problems, it would be wise to modify the program so that it can either find approximate solutions or provide bounds. Pruning the search structure in various ways allows one to give approximate solutions and bounds although great care and work would have to be invested to determine how good the bounds or approximations might be. Such techniques will be necessary when the best edge selection rules, reduction techniques, and intelligent programming fail to stop the exponential growth of the search structure.

5. Remarks

The computer used was a Digital Equipment Corporation VAX 11/780 with a floating point accelerator. The programs were written in the language C using double precision (64 bits)

arithmetic operations. Those interested in the actual program should contact the author.



i	head	tail	elbl	link	next	prev
0	0	0	0	0	24	0
1	3	1	0	0	10	6
2	4	2	0	0	16	7
3	4	3	0	0	20	9
4	4	4	0	0	22	11
5	3	5	0	0	23	17
6	2	1	1	7	1	8
7	1	2	1	6	2	12
8	3	1	2	9	6	10
9	1	3	2	8	3	13
10	4	1	3	11	8	1
11	1	4	3	10	4	15
12	3	2	4	13	7	14
13	2	3	4	12	9	18
14	4	2	5	15	12	16
15	2	4	5	14	11	19
16	5	2	6	17	14	2
17	2	5	6	16	5	21
18	4	3	7	19	13	20
19	3	4	7	18	15	22
20	5	3	8	21	18	3
21	3	5	8	20	17	23
22	5	4	9	23	19	4
23	4	5	9	22	21	5
24	0	0	0	25	0	24

indexed by gadr

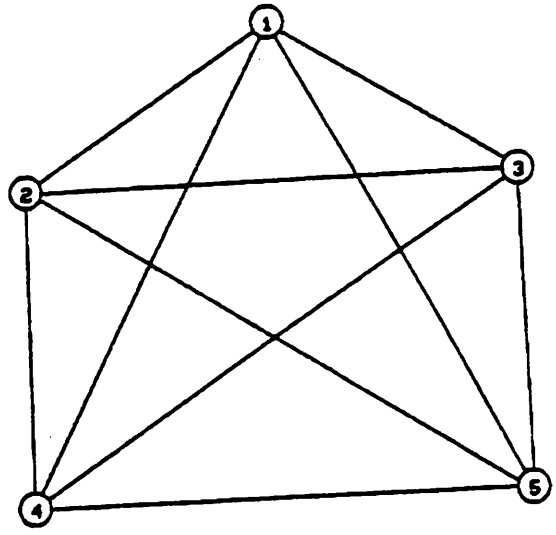
j	p	q	gadr
0	0.000000	1.000000	10
1	0.100000	0.900000	6
2	0.200000	0.800000	8
3	0.300000	0.700000	10
4	0.500000	0.500000	12
5	0.600000	0.400000	14
6	0.700000	0.300000	16
7	0.800000	0.200000	18
8	0.800000	0.200000	20
9	0.900000	0.100000	22
10	0.000000	1.000000	11
11	0.000000	1.000000	12
12	0.000000	1.000000	13
13	0.000000	1.000000	14
14	0.000000	1.000000	15
15	0.000000	1.000000	16
16	0.000000	1.000000	17
17	0.000000	1.000000	18
18	0.000000	1.000000	19
19	0.000000	1.000000	20
20	0.000000	1.000000	21
21	0.000000	1.000000	22
22	0.000000	1.000000	23
23	0.000000	1.000000	24
24	0.000000	1.000000	24

indexed by elbl

Figure 4.1 Network Data Structure

Multi-Terminal Network Reliability
The network has 5 vertices
and 10 edges as follows:

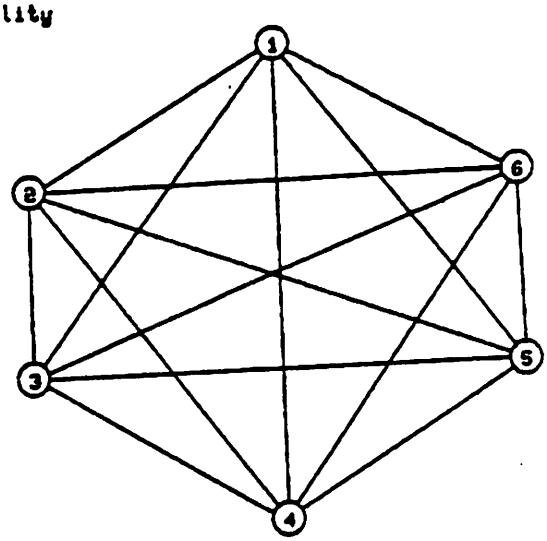
edge	head	tail	P[working]
1	1	2	0.250
2	1	3	0.250
3	1	4	0.250
4	1	5	0.250
5	2	3	0.250
6	2	4	0.250
7	2	5	0.250
8	3	4	0.250
9	3	5	0.250
10	4	5	0.250



AC	k	esr	mrp[k]	calls	leaves	cpusec	dors	pers	dtrs	ubrs
3	5	1	0.157692	11	6	0.317	6	21	10	0
3	2	1	0.421415	13	7	0.167	7	21	9	0
3	2	2	0.421415	13	7	0.217	6	21	10	0
3	2	3	0.421415	11	6	0.133	6	21	10	0
3	2	4	0.421415	13	7	0.167	7	21	9	0
3	2	6	0.421415	11	6	0.183	6	21	10	0
3	2	7	0.421415	13	7	0.133	7	21	9	0
3	2	8	0.421415	13	7	0.150	7	21	9	0
3	2	9	0.421415	19	10	0.267	7	21	9	0
3	2	11	0.421415	11	6	0.150	6	22	12	0
3	2	12	0.421415	13	7	0.117	7	21	9	0
3	2	2	0.421415	13	7	0.167	6	21	10	0
3	2	2	0.421415	13	7	0.150	7	21	9	0
3	2	2	0.421415	13	7	0.200	6	22	11	0
3	2	2	0.421415	13	7	0.167	6	22	11	0

Figure 4.2 Five Vertex Complete Graph Computations

```
Multi-Terminal Network Reliability
 The network has   6 vertices
  and  15 edges as follows:
 edge head tail  P[working]
    1    1    2       0.250
    2    2    3       0.300
    3    3    4       0.350
    4    4    5       0.400
    5    5    6       0.450
    6    1    6       0.500
    7    1    3       0.550
    8    1    4       0.600
    9    1    5       0.650
   10    2    6       0.700
   11    2    5       0.750
   12    2    4       0.800
   13    4    6       0.850
   14    3    6       0.900
   15    3    5       0.950
```

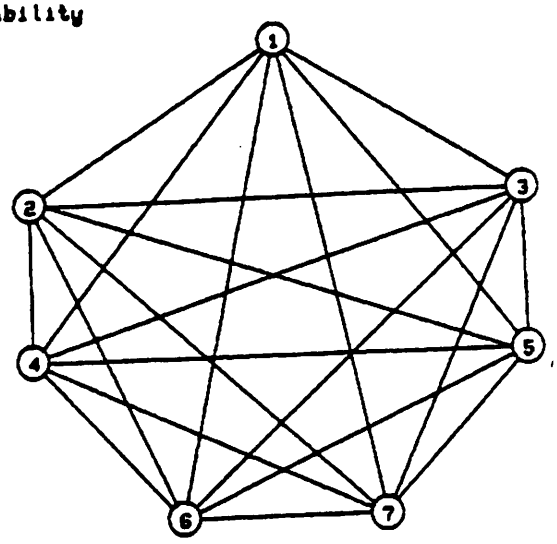


AC	k	esr	rrp[k]	calls	leaves	cpusec	dcrs	pcrs	dtrs	ubrs
1	2	8	0.966655	1887	944	26.317	276	0	0	0
1	4	8	0.960549	2527	1264	34.933	501	0	0	0
1	6	2	0.958219	2591	1296	36.383	774	0	0	0
1	6	11	0.958219	2591	1296	35.817	977	0	0	0
2	2	9	0.966655	129	65	1.933	65	94	0	0
2	4	9	0.960549	227	114	3.500	152	154	0	0
2	6	1	0.958219	239	120	3.150	206	160	0	0
2	6	12	0.958219	239	120	3.267	206	160	0	0
3	2	1	0.966655	61	31	0.617	29	94	36	0
3	2	2	0.966655	61	31	0.717	28	98	41	0
3	2	3	0.966655	53	27	0.833	24	96	47	0
3	2	4	0.966655	61	31	0.717	29	94	36	0
3	2	6	0.966655	59	30	0.683	30	100	46	0
3	2	7	0.966655	61	31	0.767	29	94	36	0
3	2	8	0.966655	57	29	0.400	29	94	36	0
3	2	9	0.966655	81	41	0.983	29	94	36	0
3	2	11	0.966655	55	28	0.717	28	107	57	0
3	2	12	0.966655	61	31	0.433	29	94	36	0
3	4	8	0.960549	47	24	0.417	24	94	41	0
3	4	12	0.960549	103	52	0.767	60	144	63	0
3	6	1	0.958219	47	24	0.367	24	94	41	0
4	2	8	0.966655	25	13	0.267	13	46	4	12
4	2	12	0.966655	29	15	0.267	13	46	4	12
4	4	8	0.960549	23	12	0.233	12	46	5	12
4	4	12	0.960549	39	20	0.317	22	63	7	18
4	6	10	0.958219	21	11	0.183	11	42	4	12
4	6	12	0.958219	23	12	0.217	12	46	5	12

Figure 4.3 Six Vertex Complete Graph Computations

Multi-Terminal Network Reliability
The network has 7 vertices
and 21 edges as follows:

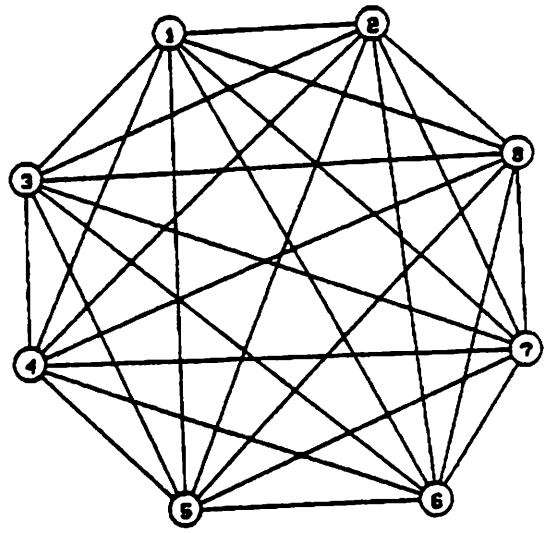
edge	head	tail	P[working]
1	1	2	0.300
2	1	3	0.400
3	1	4	0.500
4	1	5	0.600
5	1	6	0.700
6	1	7	0.800
7	2	3	0.900
8	2	4	0.300
9	2	5	0.400
10	2	6	0.500
11	2	7	0.600
12	3	4	0.700
13	3	5	0.800
14	3	6	0.900
15	3	7	0.300
16	4	5	0.400
17	4	6	0.500
18	4	7	0.600
19	5	6	0.700
20	5	7	0.800
21	6	7	0.900



AC	k	esr	mrp[k]	calls	leaves	cpusec	dors	pers	dtrs	wbrs
1	2	7	0.988612	6737	3369	109.567	758	0	0	0
2	2	1	0.988612	651	326	9.833	326	485	0	0
3	2	8	0.988612	291	146	2.150	146	485	180	0
4	2	12	0.988612	131	66	1.167	66	245	20	60
1	7	4	0.973626	33613	16807	489.100	4173	0	0	0
2	7	1	0.973625	1439	720	24.750	1237	975	0	0
3	7	1	0.973625	239	120	2.433	120	485	206	0
4	7	1	0.973625	119	60	1.250	60	245	26	60
4	7	11	0.973625	149	75	1.333	75	311	127	40
4	7	12	0.973625	119	60	1.167	60	245	26	60
3	2	9	0.988612	411	206	2.817	146	485	180	0
3	3	9	0.988314	659	330	4.683	299	747	288	0
3	4	9	0.975919	791	396	5.433	427	879	342	0
3	5	9	0.974253	807	404	5.700	487	917	356	0
3	6	9	0.974036	667	334	4.700	420	844	331	0
3	7	9	0.973625	239	120	1.967	120	485	206	0

Figure 4.4 Seven Vertex Complete Graph Computations

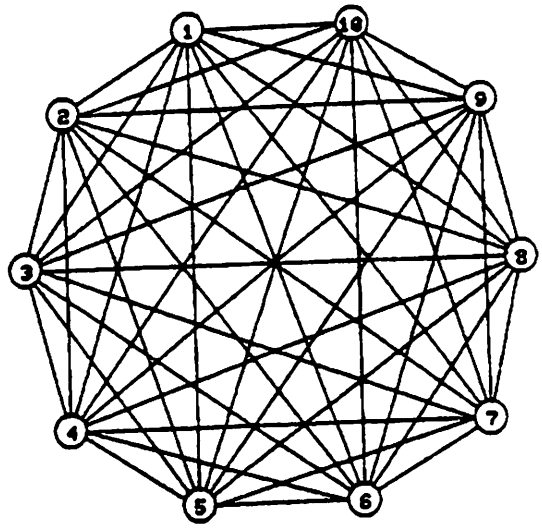
```
 Multi-Terminal Network Reliability
The network has    8 vertices
 and   28 edges as follows:
edge head tail  P[working]
   1    1    2       0.500
   2    1    3       0.400
   3    1    4       0.400
   4    1    5       0.300
   5    1    6       0.300
   6    1    7       0.300
   7    1    8       0.200
   8    2    3       0.200
   9    2    4       0.200
  10    2    5       0.200
  11    2    6       0.100
  12    2    7       0.100
  13    2    8       0.200
  14    3    4       0.200
  15    3    5       0.300
  16    3    6       0.300
  17    3    7       0.300
  18    3    8       0.400
  19    4    5       0.400
  20    4    6       0.400
  21    4    7       0.400
  22    4    8       0.500
  23    5    6       0.500
  24    5    7       0.600
  25    5    8       0.600
  26    6    7       0.600
  27    6    8       0.700
  28    7    8       0.800
```



AC	k	esr	arp[k]	calls	leaves	cpusec	dors	pers	ditrs	ubrs
3	2	8	0.787963	1753	877	13.267	877	2931	1080	0
3	2	9	0.787963	2473	1237	16.400	877	2931	1080	0
4	2	6	0.787963	859	430	7.433	401	1455	270	325
4	2	9	0.787963	1033	517	8.533	397	1491	120	360
3	8	9	0.650525	1439	720	12.067	720	2931	1237	0
4	8	9	0.650525	719	360	7.167	360	1491	157	360
4	8	11	0.650525	893	447	8.967	447	1820	836	247
4	8	12	0.650525	719	360	6.867	360	1491	157	360
4	8	2	0.650525	781	391	7.733	391	1633	320	329

Figure 4.5 Eight Vertex Complete Graph Computations

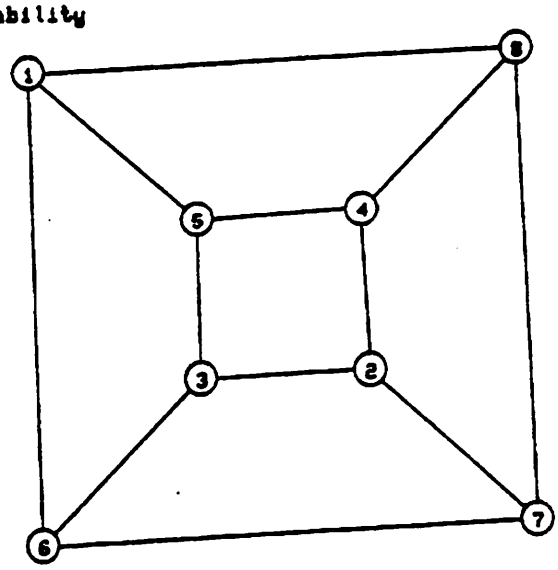
```
. Multi-Terminal Network Reliability
The network has  10 vertices
 and  45 edges as follows:
edge head tail  P[working]
   1    1    2       0.400
   2    1    3       0.500
   3    1    4       0.600
   4    1    5       0.500
   5    1    6       0.400
   6    1    7       0.300
   7    1    8       0.400
   8    1    9       0.500
   9    1   10       0.600
  10    2    3       0.600
  11    2    4       0.700
  12    2    5       0.600
  13    2    6       0.500
  14    2    7       0.400
  15    2    8       0.300
  16    2    9       0.200
  17    2   10       0.200
  18    3    4       0.300
  19    3    5       0.400
  20    3    6       0.500
  21    3    7       0.600
  22    3    8       0.700
  23    3    9       0.800
  24    3   10       0.800
  25    4    5       0.700
  26    4    6       0.600
  27    4    7       0.500
  28    4    8       0.400
  29    4    9       0.300
  30    4   10       0.300
  31    5    6       0.200
  32    5    7       0.100
  33    5    8       0.200
  34    5    9       0.300
  35    5   10       0.400
  36    6    7       0.400
  37    6    8       0.500
  38    6    9       0.600
  39    6   10       0.700
  40    7    8       0.700
  41    7    9       0.800
  42    7   10       0.800
  43    8    9       0.900
  44    8   10       0.600
  45    9   10       0.500
```



AC	k	esr	erp[k]	calls	leaves	cpusec	dors	pers	dtrs	ubrs
4	9	12	0.979756	40319	20160	391.417	20160	83756	8801	20160
4	10	12	0.979447	40319	20160	380.533	20160	83756	8801	20160

Figure 4.6 Ten Vertex Complete Graph Computations

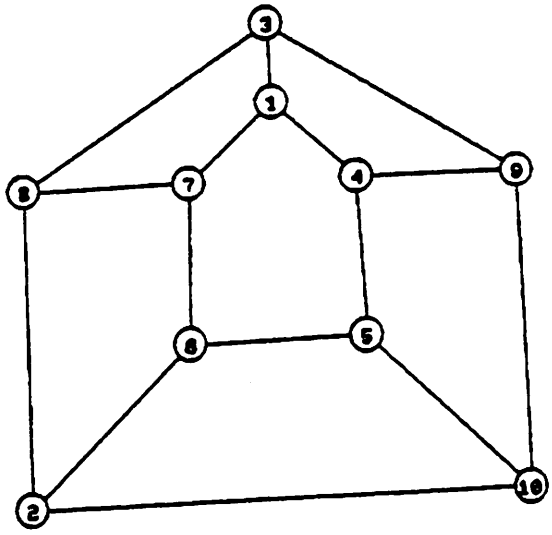
```
% Multi-Terminal Network Reliability
 The network has    8 vertices
  and   12 edges as follows:
 edge head tail  P[working]
    1    1    5       0.600
    2    1    6       0.500
    3    1    8       0.400
    4    2    3       0.600
    5    2    4       0.100
    6    2    7       0.200
    7    3    5       0.700
    8    3    6       0.400
    9    4    5       0.500
   10    4    8       0.200
   11    6    7       0.300
   12    7    8       0.100
```



AC	k	esr	mrp[k]	calls	leaves	cpusec	dors	pers	dtrs	wbrs
1	2	8	0.372830	659	330	9.750	240	0	0	0
1	4	8	0.216623	767	384	10.717	318	0	0	0
1	8	2	0.055568	767	384	11.483	417	0	0	0
1	8	11	0.055568	767	384	11.400	490	0	0	0
2	2	9	0.372830	141	71	2.067	105	31	0	0
2	4	9	0.216622	225	113	3.317	179	66	0	0
2	8	1	0.055568	265	133	3.750	275	89	0	0
2	8	12	0.055568	265	133	3.767	259	116	0	0
3	2	8	0.372830	31	16	0.233	17	43	38	0
3	2	12	0.372830	35	18	0.267	17	44	38	0
3	4	8	0.216623	33	17	0.283	19	44	38	0
3	4	12	0.216623	55	28	0.433	36	50	37	0
3	8	1	0.055568	21	11	0.217	11	32	28	0
4	2	8	0.372830	19	10	0.217	11	21	20	6
4	2	12	0.372830	19	10	0.200	11	21	20	6
4	4	8	0.216623	19	10	0.200	11	19	18	6
4	4	12	0.216623	19	10	0.200	11	15	14	8
4	8	11	0.055568	11	6	0.117	6	12	13	5
4	8	12	0.055568	11	6	0.117	6	13	13	5

Figure 4.7 Eight Vertex Cubic Graph Computations

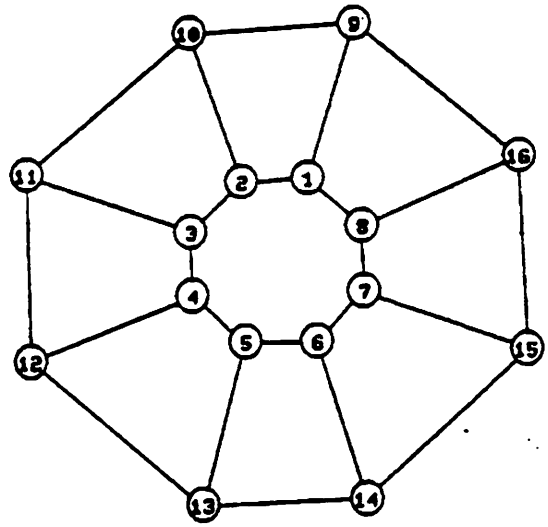
```
% Multi-Terminal Network Reliability
 The network has  10 vertices
  and  15 edges as follows:
 edge head tail  P[working]
    1    1    3       0.200
    2    1    4       0.300
    3    1    7       0.400
    4    2    6       0.300
    5    2    8       0.400
    6    2   10       0.500
    7    3    8       0.300
    8    3    9       0.400
    9    4    5       0.400
   10    4    9       0.500
   11    5    6       0.500
   12    5   10       0.600
   13    6    7       0.600
   14    7    8       0.700
   15    9   10       0.600
```



AC	k	esr	mrp[k]	calls	leaves	cpusec	dors	pors	dtrs	wbrs
1	2	9	0.292308	767	384	11.883	441	0	0	0
1	4	9	0.134926	1447	724	23.533	903	0	0	0
1	8	1	0.091596	3609	1805	54.217	2110	0	0	0
1	10	1	0.082546	3609	1805	50.567	2217	0	0	0
2	2	9	0.292308	483	242	7.367	363	100	0	0
2	4	9	0.134926	855	428	12.433	712	219	0	0
2	8	1	0.091596	1127	564	14.950	1162	321	0	0
2	10	1	0.082546	1127	564	15.867	1212	321	0	0
2	10	2	0.082546	1127	564	16.233	1244	386	0	0
3	2	12	0.292308	97	49	0.867	50	115	101	0
3	4	8	0.134926	109	55	1.133	60	116	125	0
3	4	12	0.134926	129	65	1.133	80	143	118	0
3	8	1	0.091596	125	63	1.067	79	118	121	0
3	10	1	0.082546	51	26	0.500	26	78	71	0
4	2	8	0.292308	51	26	0.583	29	52	57	15
4	2	12	0.292308	49	25	0.517	28	52	54	16
4	4	8	0.134926	71	36	0.783	39	54	70	19
4	4	12	0.134926	79	40	0.833	54	70	64	18
4	8	11	0.091596	73	37	0.783	44	40	68	22
4	8	12	0.091596	39	20	0.467	22	41	42	14
4	10	11	0.082546	29	15	0.283	15	34	38	11
4	10	12	0.082546	29	15	0.317	15	39	37	11

Figure 4.8 Ten Vertex Cubic Graph Computations

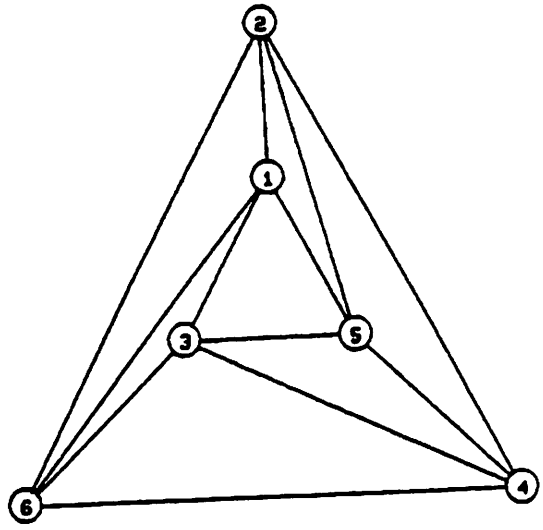
```
% Multi-Terminal Network Reliability
 The network has  16 vertices
  and  24 edges as follows:
 edge head tail  P[working]
    1    1    2       0.300
    2    2    3       0.500
    3    3    4       0.600
    4    4    5       0.600
    5    5    6       0.300
    6    6    7       0.200
    7    7    8       0.100
    8    1    8       0.600
    9    9   10       0.200
   10    9   16       0.300
   11   10   11       0.300
   12   11   12       0.800
   13   12   13       0.500
   14   13   14       0.300
   15   14   15       0.400
   16   15   16       0.700
   17    1    9       0.300
   18    2   10       0.400
   19    3   11       0.300
   20    4   12       0.200
   21    5   13       0.400
   22    6   14       0.800
   23    7   15       0.300
   24    8   16       0.400
```



AC	k	esr	mrp[k]	calls	leaves	cpusec	dors	pers	ctrs	ubrs
3	2	8	0.328570	745	373	8.100	395	1096	1158	0
3	2	12	0.328570	605	303	6.183	366	586	672	0
4	2	12	0.328570	337	169	4.533	189	194	354	126
4	2	11	0.328570	377	189	6.617	189	452	672	106
3	16	1	0.004134	493	247	6.633	247	793	771	0
4	8	8	0.017165	253	127	3.867	127	261	359	120
4	16	8	0.004134	253	127	3.883	127	261	359	120
4	16	12	0.004134	317	159	4.733	159	462	506	88
3	2	2	0.328570	617	309	6.000	362	672	763	0
3	2	9	0.328570	567	284	5.733	388	487	607	0
3	3	2	0.180407	941	471	9.083	581	936	1068	0
3	3	9	0.180407	1121	561	10.383	786	872	1023	0
3	4	2	0.121166	1433	715	13.617	935	1257	1473	0
3	4	9	0.121166	1809	905	15.633	1309	1285	1416	0
3	5	9	0.084071	2639	1320	23.000	1978	1758	1824	0
3	6	9	0.044717	3601	1801	30.017	2802	2308	2280	0
3	7	9	0.020059	4645	2323	38.800	3765	2908	2788	0
3	8	9	0.017165	5623	2812	50.950	4763	3355	3268	0
3	9	9	0.010303	6793	3397	63.817	5827	4063	4035	0
3	10	9	0.007371	7289	3645	64.967	6320	4341	4323	0
3	11	9	0.005873	6397	3196	58.167	5498	3963	3965	0
3	12	9	0.005527	4959	2476	42.667	4160	3236	3281	0
3	13	9	0.004866	3539	1767	30.850	2828	2531	2591	0
3	14	9	0.004683	2331	1166	20.800	1710	1965	2001	0
3	15	9	0.004297	1415	708	13.367	933	1532	1522	0
3	16	9	0.004134	495	248	4.983	248	923	877	0

Figure 4.9 Sixteen Vertex Cubic Graph Computations

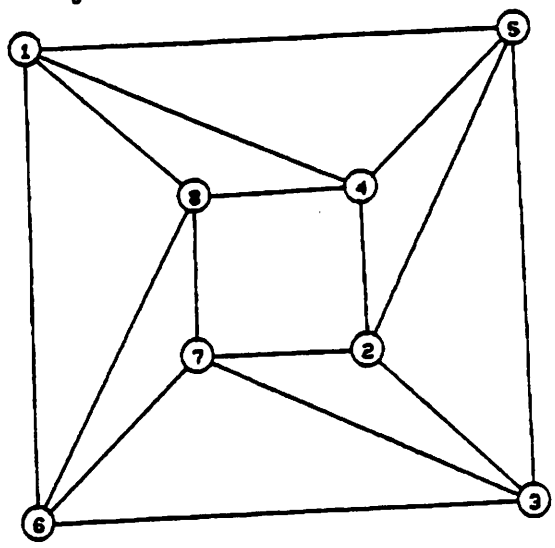
```
% Multi-Terminal Network Reliability
 The network has   6 vertices
  and  12 edges as follows:
 edge head tail  P[working]
    1    1    2        0.400
    2    1    3        0.100
    3    1    5        0.200
    4    1    6        0.300
    5    2    4        0.400
    6    2    5        0.500
    7    2    6        0.600
    8    3    4        0.700
    9    3    5        0.600
   10    3    6        0.800
   11    4    5        0.900
   12    4    6        0.500
```



AC	k	esr	mrp[k]	calls	leaves	cpusec	dors	pers	dtrs	ubrs
1	2	9	0.650331	145	73	1.917	58	0	0	0
1	3	9	0.599095	287	144	3.650	114	0	0	0
1	4	9	0.589955	417	209	5.667	158	0	0	0
1	5	9	0.583208	497	249	6.267	182	0	0	0
2	2	9	0.650331	63	32	0.867	36	35	0	0
2	3	9	0.599095	109	55	1.433	69	63	0	0
2	4	9	0.589955	127	64	1.700	93	72	0	0
2	5	9	0.583208	127	64	1.617	104	72	0	0
3	2	8	0.650331	27	14	0.200	14	44	23	0
3	3	8	0.599095	27	14	0.200	14	43	25	0
3	3	12	0.599095	37	19	0.233	21	49	26	0
3	4	8	0.589955	25	13	0.200	13	42	24	0
3	5	8	0.583208	23	12	0.167	12	41	23	0
3	6	1	0.572831	21	11	0.150	11	41	23	0
4	2	8	0.650331	15	8	0.133	8	24	9	4
4	2	12	0.650331	15	8	0.100	7	19	5	5
4	3	8	0.599095	13	7	0.117	7	18	4	7
4	3	1	0.599095	19	10	0.133	10	22	6	7
4	4	8	0.589955	13	7	0.117	7	20	6	6
4	4	11	0.589955	15	8	0.133	8	22	10	6
4	5	8	0.583208	13	7	0.117	7	22	8	5
4	6	1	0.572831	15	8	0.117	8	29	14	3
4	6	12	0.572831	11	6	0.100	6	19	6	5
4	6	12	0.572831	11	6	0.083	6	19	6	5
4	6	12	0.572831	11	6	0.117	6	19	6	5

Figure 4.10 Six Vertex Quartic Graph Computations

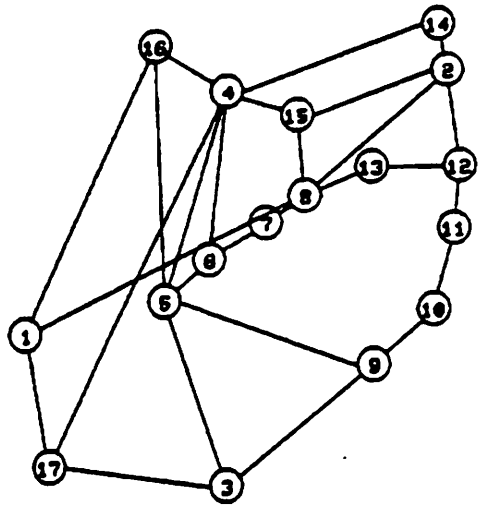
```
% Multi-Terminal Network Reliability
 The network has    8 vertices
  and   16 edges as follows:
 edge head tail  P[working]
    1    1    4       0.300
    2    1    5       0.600
    3    1    6       0.500
    4    1    8       0.400
    5    2    3       0.600
    6    2    4       0.100
    7    2    5       0.400
    8    2    7       0.200
    9    3    5       0.700
   10    3    6       0.400
   11    3    7       0.500
   12    4    5       0.500
   13    4    8       0.200
   14    6    7       0.300
   15    6    8       0.600
   16    7    8       0.100
```



AC	k	esr	mrp[k]	calls	leaves	cpusec	dors	pers	dtrs	wbrs
2	8	9	0.243053	851	426	12.450	816	438	0	0
3	8	9	0.243053	75	38	0.667	38	141	88	0
4	8	9	0.243053	45	23	0.517	23	81	43	15
2	2	8	0.585933	509	255	7.133	287	278	0	0
3	2	8	0.585933	111	56	0.850	58	194	118	0
4	2	8	0.585933	77	39	0.683	41	129	69	16
3	4	8	0.419775	117	59	1.083	61	191	121	0
3	4	12	0.419775	189	95	1.600	107	220	132	0
4	4	12	0.419775	85	43	0.817	49	107	49	26
4	8	12	0.243053	43	22	0.450	22	73	34	16
3	8	12	0.243053	75	38	0.583	38	137	82	0
4	8	11	0.243053	47	24	0.467	24	80	56	14

Figure 4.11 Eight Vertex Quartic Graph Computations

```
 Multi-Terminal Network Reliability
The network has  17 vertices
 and  27 edges as follows:
edge head tail  P[working]
   1    1   17       0.400
   2    1   16       0.500
   3    1    8       0.700
   4    2   15       0.800
   5    2   14       0.900
   6    2   12       0.700
   7    2    8       0.600
   8    3   17       0.300
   9    3    9       0.500
  10    3    5       0.400
  11    4   17       0.500
  12    4   16       0.800
  13    4   15       0.900
  14    4   14       0.400
  15    4    6       0.700
  16    4    5       0.700
  17    5   16       0.600
  18    5    9       0.700
  19    5    6       0.800
  20    6    7       0.800
  21    7    8       0.900
  22    8   15       0.900
  23    8   13       0.800
  24    9   10       0.600
  25   10   11       0.700
  26   11   12       0.800
  27   12   13       0.900
```



AC	k	esr	srp[k]	calls	leaves	cpusec	dors	pers	dtrs	ubrs
4	2	9	0.849860	403	202	4.267	191	500	409	90
4	17	12	0.298481	155	78	1.667	78	274	109	77
4	17	11	0.298481	207	104	2.483	104	332	310	57

Figure 4.12 A "Practical" Example

Bibliography

A.V. Aho, J.E. Hopcroft, and J.D. Ullman

[1974] **The Design and Analysis of Computer Algorithms,** Addison-Wesley, Reading, MA.

Michael O. Ball

[1977] **Network Reliability Analysis: Algorithms and Complexity,** Ph. D. thesis, Cornell University.

[1979] *Computing Network Reliability* , **Operations Research 27** : 823-838.

[1980] *Complexity of Network Reliability Computations,* **Networks 10** : 153-165.

M.O. Ball and G.L. Nemhauser

[1979] *Matroids and a Reliability Analysis Problem,* **Mathematics of Operations Research 4** : 132-143.

M.O. Ball and J.S. Provan

[1981a] *The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected,* Working Paper 81-002, University of Maryland at College Park.

[1981b] *Bounds on the Reliability Polynomial for Shellable Independence Systems,* Working Paper 81-013, University of Maryland at College Park.

[1981c] *Calculating Bounds on Reachability and Connectedness in Stochastic Networks,* Working Paper 81-012, University of Maryland at College Park.

M.O. Ball and R. Van Slyke

[1977] *Backtracking Algorithms for Network Reliability Analysis,* **Annals of Discrete Mathematics 1** : 49-64.

Richard E. Barlow

[1982] *Set Theoretic Signed Domination for Coherent Systems,* University of California at Berkeley.

R.E. Barlow and F. Proschan

[1975] **Statistical Theory of Reliability and Life Testing,** Holt, Rinehart, and Winston, New York.

Z.W. Birnbaum and J.D. Esary

[1965] *Modules of Coherent Binary Systems,* **SIAM Journal on Applied Mathematics 13** : 444-462.

Z.W. Birnbaum, J.D. Esary, and S.C. Saunders

[1961] *Multi-component Systems and Structures and their Reliability,* **Technometrics 3** : 55-77.

R.C. Buck

[1943] *Partition of Space,* **Amer. Math. Monthly 50** : 541-544.

J.A. Buzacott

[1976] *A Recursive Algorithm for Finding the Probability that a Graph is Disconnected,* Working Paper 76-016, Department of Industrial Engineering, University of Toronto.

[1980] *A Recursive Algorithm for Finding Reliability Measures Related to the Connection of Nodes in a Graph,* **Networks 10** : 311-327.

J.A. Buzacott and S.K. Chang

[1979] *Partitions and Partitioning-Decomposition for Network Reliability Analysis,* Working Paper 79-017, Department of Industrial Engineering, University of Toronto.

Mark Chang

[1981] **A Graph Theoretic Appraisal of the Complexity of Network Reliability Algorithms,** Ph. D. thesis, University of California at Berkeley.

Nicos Christofides

[1975] **Graph Theory: An Algorithmic Approach,** Academic Press, New York.

Henry H. Crapo

[1967] *A Higher Invariant for Matroids,* **Journal of Combinatorial Theory 2** : 406-417.

N. Deo

[1974] **Graph Theory with Applications to Engineering and Computer Science,** Prentice-Hall.

L.R. Ford and D.R. Fulkerson

[1962] **Flows in Networks,** Princeton University Press, NJ.

Howard Frank

[1975] *Computer Networks: Art to Science to Art,* **Networks 5** : 7-32.

H. Frank and Wushow Chou

[1972] *Topological Optimization of Computer Networks,* **Proceedings of the IEEE 60,** no. 11 : 1385-1397.

H. Frank and I.T. Frisch

[1970a] *Analysis and Design of Survivable Networks,* **IEEE Transactions on Communication Technology COM-18** : 501-519.

[1970b] *Network Analysis,* **Scientific American 223,** July : 94-103.

L. Fratta and U. Montanari

[1973] *A Boolean Algebra Method for Computing Terminal Reliability in a Communication Network,* **IEEE Transactions on Circuit Theory CT-20** : 203-211.

Harold N. Gabow

[1977a] *Finding All Spanning Trees of Undirected and Directed Graphs,* Working Paper CU-CS-103-77, Department of Computer Science, University of Colorado at Boulder.

[1977b] *Two Algorithms for Generating Weighted Spanning Trees in Order,* **SIAM Journal on Computing 6** : 139-150.

M.R. Garey and D.S. Johnson

[1979] **Computers and Intractibility: A Guide to the Theory of NP-Completeness,** Freeman, San Francisco, CA.

Curtis Greene

[1977] *Acyclic Orientations,* **Higher Combinatorics,** ed. M. Aigner : 65-68.

Jane N. Hagstrom

[1980] **Combinatoric Tools for Computing Network Reliability,** Ph. D. thesis, University of California at Berkeley.

Eberhard Hänsler

[1972] *A Fast Recursive Algorithm to Calculate the Reliability of a Communication Network,* **IEEE Transactions on Communication COM-20** : 637-640.

E. Hänsler, G.K. McAuliffe, and R.S. Wilkov

[1974] *Exact Calculation of Computer Network Reliability,* **Networks 4** : 95-112.

F. Harary

[1969] **Graph Theory,** Addison-Wesley, Reading, MA.

F. Harary and E.M. Palmer

[1973] **Graphical Enumeration,** Academic Press, New York.

J.E. Hopcroft and R.E. Tarjan

[1973] *Dividing a Graph into Triconnected Components,* **SIAM Journal on Computing 2** : 135-158.

Mark Jerrum

[1981] **On the Complexity of Evaluating Multivariate Polynomials,** Ph. D. thesis, University of Edinburgh.

Rubin Johnson

[1981a] *Network Reliability and Permutation Partitioning*, working paper.

[1981b] *Network Reliability and Acyclic Orientations*, working paper.

[1981c] *The Wheatstone Bridge Reduction*, working paper.

Richard M. Karp

[1972] *Reduciblity Among Combinatorial Problems*, **Complexity of Computer Computations**, eds. R.E. Miller and J.W. Thatcher, Plenum Press, New York : 85-103.

[1975] *On the Complexity of Combinatorial Problems*, **Networks 5** : 45-68.

A.K. Kel'mans

[1967] *Connectivity of Probabilistic Networks*, **Automation and Remote Control 3** : 444-460. Translated from **Avtomatika i Telemekhanika 3** : 98-116.

A. Kershenbaum and R. Van Slyke

[1973] *Recursive Analysis of Network Analysis*, **Networks 3** : 81-94.

Donald E. Knuth

[1975] *Estimating the Efficiency of Backtrack Programs*, **Mathematics of Computation 129** : 121-136.

Eugene L. Lawler

[1976] **Combinatorial Optimization: Networks and Matroids**, Holt, Rinehart, and Winston, New York.

J.D. Leggett

[1968] **Synthesis of Reliable Communication Networks**, Ph. D. thesis, University of Pennsylvania.

J.D. Leggett and S.D. Bedrosian

[1969] *Synthesis of Reliable Networks*, **IEEE Transactions on Circuit Theory**, 384-385.

G.J. Minty

[1965] *A Simple Algorithm for Listing all the Trees of a Graph*, **IEEE Transactions on Circuit Theory CT-12**: 120.

K.B. Misra

[1970] *An Algorithm for the Reliability Evaluation of Redundant Networks*, **IEEE Transactions on Reliability R-19**: 146-151.

E.F. Moore and C.E. Shannon

[1956] *Reliable Circuits Using Less Reliable Relays*, **Journal of the Franklin Institute 262**, no. 3 : 191-208, no. 4 : 281-297.

Fred Moskowitz

[1958] *The Analysis of Redundancy Networks,* **AIEE Transactions on Communications and Electronics 77, Part I** : 627-632.

J.D. Murchland

[1975] *Fundamental Concepts and Relations for Reliability Analysis of Multi-state Systems,* **Reliability and Fault Tree Analysis,** eds. R.E. Barlow, J.B. Fussell, and N.D. Singpurwalla, SIAM : 581-618.

J.D. Murchland and R.D. Shier

[1973] *Calculating the Probability that an Undirected Graph is Disconnected,* unpublished working paper.

Rita Procesi-Ciampi

[1981] *A Minimality Property for Acyclic Orientations,* working paper, University of California at Berkeley.

R.C. Read and R.E. Tarjan

[1975] *Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees,* **Networks 5** : 237-252.

Arnon Rosenthal

[1974] **Computing Reliability of Complex Systems,** Ph. D. thesis, University of California at Berkeley.

[1975] *A Computer Scientist looks at Reliability Computations,* **Reliability and Fault Tree Analysis,** eds. R.E. Barlow, J.B. Fussell, and N.D. Singpurwalla, SIAM : 133-152.

[1977] *Computing the Reliability of Complex Networks,* **SIAM Journal of Applied Mathematics 32** : 384-393.

A. Rosenthal and D. Frisque

[1977] *Transformations for Simplifying Network Reliability Calculations,* **Networks 7** : 97-111.

A. Satyanarayana

[1980] *Multi-terminal Network Reliability,* Technical Report ORC80-11, University of California at Berkeley.

A. Satyanarayana and M. Chang

[1981] *Network Reliability and the Factoring Theorem,* Technical Report ORC81-12, Operations Research Center, University of California at Berkeley.

A. Satyanarayana, M. Chang, and Z. Khalil

[1981] *Some Results on the Overall Reliability of Undirected Graphs,* Technical Report ORC81-2, Operations Research Center, University of California at Berkeley.

A. Satyanarayana and J.N. Hagstrom

[1980a] *Combinatoric Properties of Directed Graphs Useful in Network Reliability,* Technical Report ORC80-8, Operations Research Center, University of California at Berkeley, also in **Networks 11** : 357-366.

[1980b] *A New Algorithm for the Reliability Analysis of Multi-Terminal Networks,* Technical Report ORC80-11, Operations Research Center, University of California at Berkeley, also in **IEEE Transactions on Reliability R-30,** no. 4 : 325-334.

A. Satyanarayana and A. Prabhakar

[1978] *New Topological Formula and Rapid Algorithm for Reliability Analysis of Complex Networks,* **IEEE Transactions on Reliability 27** : 82-100.

A. Satyanarayana and R. Procesi-Ciampi

[1981] *On Some Acyclic Orientations of a Graph,* Technical Report ORC81-11, Operations Research Center, University of California at Berkeley.

A. Satyanarayana and R. Kevin Wood

[1982] *Polygon-to-Chain Reductions and Network Reliability*

Andrew S. Shogan

[1976] *Sequential Bounding of the Reliability of a Stochastic Network,* **Operations Research 24** : 1027-1044.

[1978] *A Decomposition Algorithm for Network Reliability Analysis,* **Networks 8** : 231-251.

Richard P. Stanley

[1973] *Acyclic Orientations of Graphs,* **Discrete Mathematics 5** : 171-178.

[1977] *Cohen-McCauley Complexes,* **Higher Combinatorics,** ed. M. Aigner : 51-62.

Kenneth Steiglitz, Peter Weiner, and D.J. Kleitman

[1969] *The Design of Minimum-Cost Survivable Networks,* **IEEE Transactions on Circuit Theory CT-16,** no. 4 : 455-460.

R. Endre Tarjan

[1972] *Depth-first Search and Linear Graph Algorithms,* **SIAM Journal on Computing 1** : 146-160.

[1974] *A Note on Finding the Bridges of a Graph,* **Information Processing Letters 2** : 160-161.

Leslie G. Valiant

[1977a] *The Complexity of Computing the Permanent,* Report no. CSR-14-77, Computer Science Department, University of Edinburgh, Scotland, also in **Theoretical Computer Science 8** : 189-201 [1979].

[1977b] *The Complexity of Enumeration and Reliability Problems,* Report no. CSR-15-77, Computer Science Department, University of Edinburgh, Scotland, also in **SIAM Journal on Computing 8** : 410-421 [1979].

R. Van Slyke and H. Frank

[1972] *Network Reliability Analysis: Part I,* **Networks 1** : 279-290.

R. Van Slyke, H. Frank, and A. Kershenbaum

[1975] *Network Reliability Analysis: Part II,* **Reliability and Fault Tree Analysis,** eds. R.E. Barlow, J.B. Fussell, and N.D. Singpurwalla, SIAM : 619-650.

J. Von Neumann

[1952] *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components,* **Automata Studies,** [1956] eds. C.E. Shannon and J. McCarthy, Princeton University Press : 43-98.

D.J.A. Walsh

[1976] **Matroid Theory,** Academic Press, London.

Robert S. Wilkov

[1972] *Analysis and Design of Reliable Computer Networks,* **IEEE Transactions on Communication COM-20** : 660-678.

O. Wing and P. Demetriou

[1964] *Analysis of Probabilistic Networks,* **IEEE Transactions on Communication Technology COM-12** : 34-40.

S. Winograd and J.D. Cowan

[1963] **Reliable Computation in the Presence of Noise,** The MIT Press, MA.

R. Kevin Wood

[1980] *Efficient Calculation of the Reliability of Lifeline Networks Subject to Seismic Risk,* Technical Report ORC80-13, Operations Research Center, University of California at Berkeley.

Thomas Zaslavsky

[1975] *Facing Up to Arrangements : Face-Count Formulas for Partitions of Space by Hyperplanes,* **Memoirs of the American Mathematical Society 1,** issue 1, no. 154 : 1-102.