Monte-Carlo Algorithms for the Planar Multiterminal Network Reliability Problem*

RICHARD M. KARP†

Computer Science Division, University of California, Berkeley, California 94720

AND

MICHAEL LUBY

Computer Science Department, University of Toronto, Toronto, Ontario, Canada

This paper presents a general framework for the construction of Monte-Carlo algorithms for the solution of enumeration problems. As an application of the general framework, a Monte-Carlo method is constructed for estimating the failure probability of a multiterminal planar network whose edges are subject to independent random failures. The method is guaranteed to be effective when the failure probabilities of the edges are sufficiently small. © 1985 Academic Press, Inc.

1. INTRODUCTION

This paper investigates a classic problem in reliability theory: computing the failure probability of a network whose edges are subject to random failure. Efficient solution methods for this problem have been devised only in special cases (Satyanarayana and Wood, 1982; Wood, 1982) and there are strong indications from computational complexity theory that the general problem is intractable (Provan and Ball, 1981; Valiant, 1979). Building on our previous work on Monte-Carlo algorithms for enumeration and reliability problems (Karp and Luby, 1983, 1985; Luby, 1983), we present a Monte-Carlo method for the approximate solution of this problem when the network is planar. Our method is guaranteed to be effective when the failure probabilities of the edges are sufficiently small.

In the multiterminal network reliability problem one is given an undirected graph, within which certain vertices are designated as terminals. Each edge

† Research supported by the National Science Foundation under Grant DCR-8411954.
of this graph is in one of two states; good or bad. The states of the edges are assumed to be independent random variables, and each edge has a given probability of being bad. The network is said to fail if there exist two terminals that are not connected by any path consisting entirely of good edges; equivalently, the network fails if there exists a cut that separates two of the terminals and consists entirely of bad edges. The problem is to compute the probability that the network fails.

Our Monte-Carlo algorithm provides an estimate of the failure probability of a given planar network. The performance guarantee demanded of our algorithm is expressed in terms of two input parameters, \( \epsilon \) and \( \delta \); it is required that, with probability at least \( 1 - \delta \), the algorithm estimates the failure probability of the network with a relative error less than or equal to \( \epsilon \). We show that the computation time required to achieve this guarantee is polynomial bounded, provided the failure probabilities of the edges are sufficiently small; a more precise statement of this condition on the failure probabilities is given in Section 8.

The present paper is a continuation of previous work by the authors (Karp and Luby, 1983, 1985). Sections 2-4 are, in large part, a new exposition of the contents of these earlier papers. Sections 5-8, which are concerned specifically with the planar case of the multiterminal reliability problem, are new. Other Monte-Carlo algorithms for estimating the failure probability of a network are given in Easton and Wong (1980), Kumamoto, Tanaka, and Inoue (1977), and Van Slyke and Frank (1972).

2. Abstract Structure of the Monte-Carlo Method

A Monte-Carlo algorithm is a randomized computational method for the estimation of some quantity. The problem of estimating the volume of a region \( R \) in \( n \)-dimensional Euclidean space provides a simple example of a Monte-Carlo method. It is assumed that an algorithm is available to test whether any given point \( x \) lies in \( R \). The idea of the method is to approximate \( R \), which may be a rather complicated region, by a "nice" region \( S \) that encloses \( R \). The method requires that \( \text{vol}(S) \), the volume of \( S \), be known, and that a method be available for drawing a point at random from \( S \). A single trial of the method consists of drawing a point \( x \) at random from \( S \) and testing whether \( x \) lies in \( R \). The outcome of the trial is a random variable \( X \) which is equal to 1 if \( x \) lies in \( R \), and to 0 otherwise. Clearly, the expected value of \( X \) is equal to \( \text{vol}(R)/\text{vol}(S) \). The Monte-Carlo algorithm carries out a sequence of \( N \) independent trials, and then estimates \( \text{vol}(R) \) by the random variable \( Y = ((X_1 + X_2 + \cdots + X_N)/N) \text{vol}(S) \), where \( X_i \) is the outcome of the \( i \)th trial. Since \( E[X_i] = \text{vol}(R)/\text{vol}(S) \) we see easily that \( E[Y] = \text{vol}(R) \); i.e., \( Y \) is an unbiased estimator of \( \text{vol}(R) \). \( N \), the number of trials required, depends on the performance demanded of the algorithm. We assume that the
estimator $Y$ is required to satisfy the following condition, where $\epsilon$ and $\delta$ are positive real parameters specified by the user: $P[|Y - \text{vol}(R)/\text{vol}(R)| > \epsilon] < \delta$. The parameter $\epsilon$ specifies how small a relative error is acceptable, and $\delta$ bounds the probability with which the relative error may lie outside the acceptable range. Using elementary probability theory one can show that, if $N > \text{vol}(S)/\text{vol}(R) \ln(2/\delta)4.5/\epsilon^2$, then the algorithm will meet the performance requirement. Thus the effectiveness of the approach depends critically on the ratio of the volume of the enclosing region to the volume of the region $R$, and the art of constructing an effective Monte-Carlo algorithm lies in choosing a nice region $S$ that tightly encloses $R$, the region of interest.

The Monte-Carlo methods considered in this paper are quite similar in spirit to the volume-estimation method described in the last paragraph, but they are concerned not with geometric problems but rather with the estimation of the probabilities of certain events and the weights of certain discrete sets with weighted elements. The algorithms conform to the following scheme. A finite set $R$ is given, together with a function $a$ which assigns a weight $a(x)$ to any element $x$ in $R$. The purpose of the algorithm is to estimate $a(R)$, the sum of the weights of the elements of $R$. The algorithm constructs a "nice" set $S$ which contains $R$ as a subset, such that the weight function $a$ is defined for all elements of $S$. It is required that

(i) $a(S)$, the sum of the weights of the elements of $S$, be known;
(ii) a sampling algorithm be available which draws elements from $S$; at each trial, the probability of drawing element $x$ is $a(x)/a(S)$;
(iii) an algorithm be available for testing whether any given element $x \in S$ lies in $R$.

When these requirements are met the triple $(S, R, a)$ is called a Monte-Carlo structure with universe $S$, target set $R$, and weight function $a$.

A trial of the algorithm consists of drawing an element $x$ from $S$ and testing whether $x$ lies in $R$; $X$, the outcome of the trial, is equal to 1 if $x$ lies in $R$ and to 0 otherwise. The trial draws a sequence of $N$ samples and then estimates $a(R)$ by $Y = (X_1 + X_2 + \cdots + X_N)/N \cdot a(S)$, where $X_i$ is the outcome of the $i$th trial.

Let $\epsilon$ and $\delta$ be positive real parameters. Then the estimator $Y$ is called an $\epsilon$-$\delta$ approximation to $a(R)$ if $P[|(Y - a(R))/a(R)| > \epsilon] < \delta$.

**Theorem 1.** Let $\epsilon$ and $\delta$ be positive real numbers such that $\epsilon \leq 1 - a(R)$. If $N > a(S)/a(R) \ln(2/\delta)4.5/\epsilon^2$ then $P[|(Y - a(R))/a(R)| > \epsilon] < \delta$.

**Proof.** The following inequality due to Bernstein is cited in Rényi (1970, p. 387): Let $A$ be one of the possible results of an experiment, suppose $p = P[A] > 0$ and put $q = 1 - p$. Let the random variable $Y(n)$ denote the relative frequency of $A$ in an experiment consisting of $n$ independent trials. Then, for $0 < a \leq pq$ we have $P[|Y(n) - p| > a] \leq
$2\exp(-na/2pq(1+a/2pq)^2)$. Taking $A$ to be the event that $x$ lies in $R$, $p$, the probability of $A$, to be $a(R)/a(S)$, and $a$ to be $ea(R)/(S)$, and noting that $q < 1$ and $a/pq \leq 1$, the desired result follows from the Bernstein inequality.

Theorem 1 shows that the efficiency of the method is determined by $a(S)/a(R)$, the ratio of the weight of the universe $S$ to the weight of the target set $R$ whose weight is to be estimated.

3. Application to a Problem in Boolean Algebra

In preparation for our main objective, the construction of Monte-Carlo algorithms for the solution of network reliability problems, we consider the simpler problem of estimating the number of truth-value assignments that satisfy a Boolean formula in disjunctive normal form (abbreviated DNF). Let $x_1, x_2, \ldots, x_n$ be Boolean variables, each of which can be true or false. Let $\overline{x}_i$ be the complement of $x_i$; then $x_i$ is true if and only if $\overline{x}_i$ is false. The elements of the set $x_1, x_2, \ldots, x_n, \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n$ are called literals. A clause is the conjunction (logical "and") of a set of literals, and a DNF formula is the disjunction (logical "or") of a set of clauses. An example of a DNF formula is $x_1x_3 \lor x_1x_2x_5 \lor x_4x_5$. This formula is the disjunction of four clauses.

A truth-value assignment is a function $f$ from the set of variables into \{true, false\}. The truth-value assignment $f$ is said to satisfy a given DNF formula if, upon substitution of the truth value $f(x_i)$ for each variable $x_i$, a Boolean formula is obtained that evaluates to "true." Clearly, a truth-value assignment satisfies a DNF formula if and only if it satisfies some clause, and it satisfies a given clause if and only if it assigns the value "true" to all the literals in that clause. For example, the following truth-value assignment satisfies the second clause in the above formula, and thus satisfies the formula

$$f(x_1) = \text{true}, f(x_2) = \text{false}, f(x_3) = \text{true}, f(x_4) = \text{false}, f(x_5) = \text{true}.$$  

Let $\#F$ denote the number of truth-value assignments satisfying the DNF formula $F$. We are interested in methods of determining $\#F$, either exactly or approximately. One standard approach is based on the Principle of Inclusion and Exclusion, which can be stated as follows. Let $|S|$ denote the number of elements in the finite set $S$, and let $E_1, E_2, \ldots, E_m$ be finite sets. Then

$$|E_1 \cup E_2 \cup \cdots \cup E_m| = \sum_{I \subseteq \{1,2,\ldots,m\}} (-1)^{|I|+1} \bigcap_{i \in I} E_i|.$$  

In the application to DNF formulas $E_i$ denotes the set of truth-value assignments that satisfy clause $i$, $E_1 \cup E_2 \cup \cdots \cup E_m$ is the set of truth-value
assignments satisfying the DNF formula, and thus \( \#F = |E_1 \cup E_2 \cup \cdots \cup E_m| \). The number of truth-value assignments satisfying a given set of clauses is easily determined by inspection: if two of the clauses contain complementary literals then there are no such assignments; otherwise the number of satisfying assignments is \( 2^k \), where \( k \) is the number of variables that occur in none of the clauses. Using this rule, each term in the summation (1) can easily be evaluated. For example, if \( F = x_1 x_4 \cup x_1 \overline{x}_2 x_3 \cup x_1 x_2 x_5 \cup x_1 x_3 \overline{x}_4 \overline{x}_5 \) we have \( |E_1| = 8, \ |E_2| = 4, \ |E_3| = 4, \ |E_4| = 2, \ |E_1 \cap E_2| = 2, \ |E_1 \cap E_3| = 2, \ |E_2 \cap E_4| = 1 \); all the other intersections are empty. It follows that \( \#F = (8 + 4 + 4 + 2) - (2 + 2 + 1) = 13 \).

If the DNF formula \( F \) contains \( m \) clauses then the Inclusion-Exclusion formula entails \( 2^m - 1 \) terms, and thus is not a practical computational tool when \( m \) is large. Monte-Carlo methods provide an attractive alternative, provided that a reliable estimate of \( \#F \) is acceptable. The most straightforward Monte-Carlo method simply takes the target set \( R \) as the set of all truth assignments that satisfy \( F \) and the universe \( S \) as the set of all \( 2^n \) truth assignments to the \( n \) variables; each element of \( S \) is assigned a weight of 1. In this case the set \( S \) is certainly nice; its weight is known to be \( 2^n \), and it is trivial to sample from \( S \). However, this simple method is quite ineffectual when the ratio \( a(S)/a(R) \) is extremely large.

Fortunately, there is an alternate method which works well even when the number of truth assignments satisfying \( F \) is extremely small compared to the total number of truth assignments. In this alternate method the universe \( S \) is taken to be the set of all ordered pairs \((i, x)\) such that \( x \) is a truth assignment satisfying the \( i \)th clause. Each pair \((i, x) \in S\) has weight one. The target set \( R \) consists of those pairs \((i, x)\) such that the \( i \)th clause is the lowest-numbered clause satisfied by \( x \). Hence, the elements of \( R \) are in one-to-one correspondence with the truth assignments that satisfy \( F \). For each fixed \( i \), the number of pairs \((i, x) \in S\) is equal to \( |E_i| \), where \( E_i \) is the set of truth-value assignments satisfying clause \( i \). Hence \(|S| = \sum_{i=1}^{m} |E_i| \). Also, \(|R| = \sum_{i=1}^{m} |E_i| \), and it follows that the critical ratio \( a(S)/a(R) = |S|/|R| \approx m \).

Assuming that clause \( i \) contains no contradictory pair of literals, \(|E_i| = 2^{n-k}\), where \( k \) is the number of literals occurring in the clause. Thus \(|S|\) is easily determined by inspection of the formula \( F \). There is a simple two-stage process for drawing a pair \((i, x)\) at random from \( S \). In the first stage clause \( i \) is selected with probability \(|E_i|/|S|\). Given \( i \), a truth-value assignment satisfying clause \( i \) is chosen as follows. If the literal \( x_i \) occurs in the clause then \( x_i \) is assigned the value true; if \( \overline{x}_i \) occurs in the clause then \( x_i \) is assigned the value false; if neither \( x_i \) nor \( \overline{x}_i \) occurs in the clause then the truth value assigned to \( x_i \) is chosen at random from the set, independently of the truth values assigned to the other variables. All elements of \( S \) are equally likely to be selected by this two-stage sampling process.

We have now demonstrated that all the requirements for the successful application of the Monte-Carlo method are satisfied: the weight of \( S \) is known;
there is an efficient method for sampling from \( S \); and the ratio \( a(S)/a(R) \) is bounded above by \( m \), the number of clauses. For fixed \( \varepsilon \) and \( \delta \) the number of trials required to obtain an \( \varepsilon - \delta \) approximation to the number of satisfying assignments is \( O(m) \). The overall Monte-Carlo algorithm begins with a preprocessing phase in which the numbers \( |E_i| \) are calculated and reference tables are prepared to facilitate the execution of Monte-Carlo trials. The time required to execute the preprocessing stage is \( O(mn) \), and the time per trial thereafter is \( O(mn) \). The overall execution time of the algorithm is \( O(m^2n) \). Thus we have a polynomial-time randomized algorithm for the approximate solution of a counting problem that is complete in Valiant's complexity class \( \#P \) (Valiant, 1979). It should be pointed out, however, that our success depended critically on the fact that the Boolean formula was in disjunctive normal form. The existence of a polynomial-time \( \varepsilon - \delta \) approximation algorithm for estimating the number of truth-value assignments satisfying a Boolean formula in conjunctive normal form is not possible unless \( P = NP \).

4. The Multiterminal Network Reliability Problem

A multiterminal network is specified by

(i) a connected undirected graph \( G \) with vertex set \( V \) and edge set \( E \);
(ii) a set \( K \subseteq V \) called the set of terminals;
(iii) an assignment to each edge \( e \) of a failure probability \( p(e) \), where \( 0 < p(e) < 1 \); we say that edge \( e \) is bad with probability \( p(e) \) and good with probability \( 1 - p(e) \).

The network is said to fail if some pair of terminals cannot be joined by a path consisting entirely of good edges. The multiterminal network reliability problem is the problem of determining the failure probability of a multiterminal network.

To formalize the problem we introduce the concept of state. A state of the network is a function \( s: E \rightarrow \{ \text{good, bad} \} \). The probability of state \( s \) is

\[
b(s) = \prod_{\{e\}: s(e) = \text{bad}} p(e) \prod_{\{e\}: s(e) = \text{good}} (1 - p(e)).
\]

Edge \( e \) is good in state \( s \) if \( s(e) = \text{good} \), and otherwise is bad in state \( s \). State \( s \) is a failure state of the network if, for some pair \( x, y \subseteq K \), every path between \( x \) and \( y \) contains an edge that is bad in state \( s \). The failure probability of the network is the sum of the probabilities of the failure states.

In order to discuss the computation of the failure probability of a network we define a few basic graph-theoretic terms. We assume that the concepts of connected graph and connected component are known. A walk from vertex \( x \) to vertex \( y \) in graph \( G \) is a sequence of edges \( \{i_1, i_2\}, \{i_2, i_3\}, \ldots, \{i_{k-1}, i_k\} \) such that \( i_1 = x \) and \( i_k = y \). A path is a walk such that the vertices \( i_1, \)
i_2, \ldots, i_k$ are distinct. A walk is cyclic if $i_1 = i_k$, and a cyclic walk is called a cycle if the vertices $i_1, i_2, \ldots, i_{k-1}$ are distinct. A cut is a minimal set of edges whose removal from the connected graph $G$ results in a graph with exactly two connected components. A cut $c$ is said to separate vertices $x$ and $y$ if $x$ and $y$ lie in different components of the two-component graph obtained by deleting the edges of $c$ from $G$. A cut is called $K$-separating if it separates two vertices in $K$. State $s$ of network $(G, K, p)$ is a failure state of the network if there is a $K$-separating cut, all of whose edges are bad in state $s$.

Our goal is to estimate the probability that the network is in a failure state. Let $F$ be the set of failure states. Since $b(s)$ is the probability of state $s$, the failure probability of the network is $b(F) = \sum_{s \in F} b(s)$. We will derive a Monte-Carlo algorithm for the estimation of $b(F)$. The algorithm will require us to specify, for each failure state $s$, a $K$-separating cut $g(s)$ whose edges are all bad in state $s$. The cut $g(s)$ is called the canonical failing cut in state $s$.

A computationally simple way of specifying the cut $g(s)$ is as follows: Let the elements of $K$, the set of terminals, be designated, in an arbitrary but fixed order, as $t(1), t(2), \ldots, t(K)$. Let $G(s)$ be the graph whose vertex set is $V$, the vertex set of $G$, and whose edge set is the set of edges of $G$ that are good in state $s$. Let $C(i)$ be the vertex set of the connected component of $G(s)$ containing the terminal $t(i)$. Since $s$ is a failure state, at least one terminal lies outside component $C(i)$. There exists at least one $i$ such that the subgraph of $G$ induced by all vertices not in $C(i)$ is connected. Let $i^*$ be the least such index, and choose $g(s)$ as the $k$-separating cut of $G$ consisting of those edges having exactly one of their two endpoints in $C(i^*)$.

Armed with the specification of the canonical failing cut $g(s)$ we can specify the universe $S$, the target set $R$, and the weight function $a$ required for a Monte-Carlo structure. Let $S$ be the set of pairs $(c, s)$, where $s$ is a failure state and $c$ is a $K$-separating cut that fails in state $s$. Define the weight function $a$ by $a(c, s) = b(s)$, and let $R$ be the set of all pairs $(g(s), s)$ such that $s$ is a failure state. Then $a(R) = \sum_{s \in F} a(s, g(s)) = \sum_{s \in F} b(s) = b(F)$. The triple $(S, R, a)$ will be called the Monte-Carlo structure based on cuts for the network $(G, K, p)$.

Four requirements must be satisfied in order for the Monte-Carlo structure based on cuts to yield an efficient Monte-Carlo algorithm for estimating the failure probability of a multiterminal network: it must be easy to recognize whether a pair $(c, s) \in S$ lies in $R$, $a(S)$ must be known, it must be easy to sample from $S$, and the ratio $a(S)/a(R)$ must be small. We consider these requirements in turn. Testing whether $(c, s)$ lies in $R$ is simple. One simply computes $g(s)$, the canonical cut that fails in state $s$, and tests whether $c = g(s)$.

Computing $a(S)$ and sampling from $S$ are easy provided an explicit list of all $K$-separating cuts is available. If $c$ is a $K$-separating cut, then the sum of the probabilities of the states in which all edges of $c$ are bad is given by $P(c) = \prod_{e \in c} p(e)$. Hence $a(S) = \sum c P(c)$. To sample from $S$ one simply
chooses $K$-separating cut $c$ with probability $P(c)/a(S)$ and then chooses a state $s$ in which all edges of $c$ are bad as follows: if $e \in c$ then $s(e) =$ bad; if $e \notin c$ then $s(e)$ is bad with probability $p(e)$ and good with probability $1 - p(e)$. The randomizations required to determine $s(e)$ for edges $e$ not in the cut $c$ are performed independently. It follows that the probability of drawing the pair $(c, s)$ is $a(c, s)/a(S) = b(s)/a(S)$.

Finally, we consider the ratio $a(S)/a(R)$. For each failure state $s$, the number of pairs $(c, s)$ in $S$ is not greater than the number of $K$-separating cuts. All such pairs have the same weight $a(c, s) = b(s)$, and exactly one of these pairs lies in $R$, namely the pair $(g(s), s)$. These remarks show that the number of $K$-separating cuts is an upper bound on the ratio $a(S)/a(R)$. The following theorem gives another upper bound on this ratio.

**Theorem 2.** Let $(S, R, a)$ be the Monte-Carlo structure based on cuts associated with a network $(G, K, p)$. Then $a(S)/a(R) \leq \Pi_{e \in E}(1 + p(e))$.

**Proof.** Without loss of generality let the edge set of $G$ be $\{1, 2, \ldots, m\}$. We define a function $D$ whose domain is the set of all tuples of the form $(b(1), b(2), \ldots, b(i))$, such that $0 \leq i \leq m$ and $b(j) \in \{\text{good, bad}\}$, for $j = 1, 2, \ldots, i$. The function $D$ is defined by the following rule: $D(b(1), b(2), \ldots, b(i))$ is the conditional probability that state $s$ is a failure state, given that $s(1) = b(1), s(2) = b(2), \ldots, s(i) = b(i)$. In particular, $D(s(1), s(2), \ldots, s(m))$ is equal to 1 if $s$ is a failure state, and is equal to 0 otherwise, and, letting $\lambda$ denote the null sequence, $D(\lambda)$ is the failure probability of the network. The function $D$ satisfies the recursion $D(b(1), b(2), \ldots, b(i-1)) = p(i)D(b(1), b(2), \ldots, b(i-1), \text{bad}) + (1 - p(i))D(b(1), b(2), \ldots, b(i-1), \text{good})$. Also, $D$ satisfies the monotonic property $D(b(1), b(2), \ldots, b(i-1), \text{bad}) \geq D(b(1), b(2), \ldots, b(i-1), \text{good})$. Note that $D(\lambda) = a(R)$, since each of these quantities is equal to the failure probability of the network. Define a function $N$ with the same domain as the function $D$, by the following rules: if $s$ is a state then $N(s(1), s(2), \ldots, s(m))$ is equal to 1 if $s$ is a failure state, and is equal to 0 otherwise; $N(b(1), b(2), \ldots, b(i-1)) = p(i)N(b(1), b(2), \ldots, b(i-1), \text{bad}) + N(b(1), b(2), \ldots, b(i-1), \text{good}))$. It is easily proved by induction that $N(b(1), b(2), \ldots, b(i-1))/D(b(1), b(2), \ldots, b(i-1)) \leq \Pi_{e \in E}(1 + p(j))$. In particular $N(\lambda)/D(\lambda) \leq \Pi_{e \in E}(1 + p(j))$.

We complete the proof by showing that $N(\lambda) \geq a(S)$. This follows from the observation that $N(\lambda) = \Sigma_s \Pi_e p(e)$, where the summation is over failure states $s$ and the product is over edges $e$ that are bad in state $s$, whereas $A(S) = \Sigma_s \Pi_e p(e)$, where the summation is over the $K$-separating cuts $c$, and the product is over edges $e$ that occur in cut $c$. To every $K$-separating cut $c$ there corresponds a failure state $s$ in which the edges of $c$ are bad and all other edges are good. From this correspondence it follows that the summation for $N(\lambda)$ includes all those terms occurring in the summation for $a(S)$, so that $N(\lambda) \geq a(S)$. 
Theorem 2 establishes that the critical ratio \( a(S)/a(R) \) which determines the number of samples required in the Monte-Carlo method is small when the failure probabilities of the edges are sufficiently small.

We see that the Monte-Carlo structure based on cuts leads to an efficient algorithm for estimating the failure probability of a multiterminal network once a list of the \( K \)-separating cuts is available. Without such a list it seems difficult to compute \( a(S) \) or to sample from \( S \). In the following sections we shall show that, despite these difficulties, efficient Monte-Carlo methods that do not require the listing of cuts can be constructed, provided we restrict attention to the planar case of the multiterminal reliability problem.

5. A Dual Monte-Carlo Structure

For the rest of the paper we assume that the graph \( G = (V, E) \) is planar. Let \( V \), the number of vertices of \( G \), be denoted by \( n \); let \( E \), the number of edges of \( G \), be denoted by \( m \); and let the number of faces of \( G \) be denoted by \( n' \).

The Monte-Carlo methods we develop for planar graphs will be based on considering \( G' = (V', E) \), the planar dual of \( G \); a definition of the concept of a planar dual can be found in any text on graph theory. If \( G \) is not a triply connected graph then \( G' \) is not uniquely defined, but in that case any planar dual of \( G \) will work. Note that \( |V'| = n' \). Contrary to the usual convention, which is to give different names to the edge sets of \( G \) and \( G' \) and then employ a natural one-to-one correspondence between these sets, we adopt the convention that the edge sets of the two graphs are one and the same; every edge of \( G \) is also an edge of \( G' \), every cut of \( G \) is also a cycle of \( G' \), every state of \( G \) is a state of \( G' \), every failure state of \( G \) is a failure state of \( G' \), etc.. In developing a Monte-Carlo structure that does not require an explicit listing of cuts we shall find it easier to work with \( G' \) than with \( G \), since cycles are computationally more manageable than cuts. Our next step is to obtain a characterization of the \( K \)-separating cuts of \( G \). Let \( x \) and \( y \) be two vertices of \( G \), and let \( T \) be any path between \( x \) and \( y \). Then a cut of \( G \) separates \( x \) from \( y \) if and only if it has an odd number of edges in common with \( T \). In order to extend this observation to the case of more than two terminals, define a test set for \( K \) as a set \( \text{TEST} \) of paths of \( G \) such that

(i) every path in \( \text{TEST} \) joins two vertices in \( K \);
(ii) for each pair of vertices in \( K \), there is a walk between \( x \) and \( y \) which is a concatenation of paths contained in \( \text{TEST} \).

**Lemma 1.** Let \( \text{TEST} \) be a test set for \( K \). Then a cut \( c \) in \( G \) is a \( K \)-separating cut if and only if, for some \( T \in \text{TEST} \), \( c \) has an odd number of elements in common with \( T \).

**Proof.** If \( c \) has an odd number of elements in common with \( T \) then \( c \) separates the two endpoints of \( T \); since these endpoints lie in \( K \), \( c \) is a
K-separating cut. Conversely, if \( c \) is a \( K \)-separating cut then it separates two vertices, \( x \) and \( y \), from \( K \). There is a walk between \( x \) and \( y \) which is a concatenation of paths from \( \text{TEST} \). This walk crosses \( c \) an odd number of times, and hence one of the paths in the concatenation crosses \( c \) an odd number of times.

We give three important examples of test sets:

1. If \( x \) and \( y \) are the only two elements of \( K \) and \( T \) is a path of \( G \) between \( x \) and \( y \) then the singleton set \( \{ T \} \) is a test set for \( K \);
2. If \( K = \{ x(1), x(2), \ldots, x(k) \} \) and \( T(i) \) is a path of \( G \) between \( x(i) \) and \( x(i+1) \), \( i = 2, 3, \ldots, k \) then \( \{ T(2), T(3), \ldots, T(k) \} \) is a test set for \( K \);
3. If \( K \) consists of all vertices of \( G \) and \( \text{TREE} \) is a spanning tree of \( G \) then the set of all edges of \( \text{TREE} \) is a test set for \( K \).

In general, the minimum number of paths in a test set for \( K \) is \( |K| - 1 \).

Now suppose we are given a network \( (G, K, p) \) where the graph \( G \) is planar, together with a test set \( \text{TEST} \) for \( K \). Let \( G' = (V', E) \) be the dual of \( G \). Define a minimal failure set as a cycle in \( G \) which has an odd number of edges in common with some \( T \in \text{TEST} \); note that a minimal failure set is exactly the same as a \( K \)-separating cut, but it is now described in terms of \( G' \) rather than \( G \), and in terms of the test set \( \text{TEST} \) rather than the set of terminals \( K \). Let \( s: E \rightarrow \{ \text{good}, \text{bad} \} \) be a state; it is equally valid to view \( s \) as a state of \( G \) or as a state of \( G' \), but, in keeping with our new dual viewpoint, it is more useful to view it as a state of \( G' \). Then \( s \) is a failure state if and only, for some minimal failure set \( c \), \( s(e) = \text{bad} \) for all \( e \in c \). The failure probability of \( G' \) is the sum of the probabilities of all the failure states.

6. A MONTE-CARLO STRUCTURE BASED ON PATHS AND CYCLES

The planar multiterminal network reliability problem has now been recast in the following form. One is given a graph \( G' = (V', E) \) in which each edge \( e \) has an associated failure probability \( p(e) \). Also given is \( \text{TEST} \), a collection of subsets of \( E \). A cycle in \( G' \) is called a minimal failure set if it has an odd number of edges in common with some set \( T \in \text{TEST} \). A state is a function \( s: E \rightarrow \{ \text{good}, \text{bad} \} \), and the probability of state \( s \) is defined as

\[
b(s) = \prod_{\{ e \mid s(e) = \text{good} \}} (1 - p(e)) \prod_{\{ e \mid s(e) = \text{bad} \}} p(e).
\]

State \( s \) is called a failure state if, for some minimal failure set \( c \), \( s(e) = \text{bad} \) for all \( e \in c \). The problem is to compute the sum of the probabilities of all failure states.

The Monte-Carlo structure that was specified in the last section can now be presented entirely in terms of this new formulation of the problem. We assume that there is available an easy-to-compute function \( g \), such that, for any failure state \( s \), \( g(s) \) is a minimal failure set whose edges are all bad in state
s. Then $S$ is the set of all pairs $(c, s)$ such that $s$ is a failure state and $c$ is a minimal failure set whose edges are all bad in state $s$, $a(c, s) = b(s)$, and $R$ is the set of all pairs $(g(s), s)$ within $S$. The sum of the probabilities of all the failure states is given by $a(R)$.

The Monte-Carlo structure just described is not a suitable starting point for the construction of an efficient Monte-Carlo algorithm. The reason is that, in the absence of an explicit listing of all the minimal failure sets, there is no convenient way to compute $a(S)$ or sample from $S$. In order to alleviate these problems we define a new Monte-Carlo structure which builds more information into the description of each element of $S$. In the new structure, each element of $S$ not only specifies a minimal failure set $c$ and a failure state $s$ but also gives a "reason" why $c$ is a minimal failure set. From now on it will be convenient to assume that $E$, the edge set of $G$, is $\{1, 2, \ldots, m\}$. Let the endpoints of edge $i$ be $x(i)$ and $y(i)$. If cycle $c$ is a minimal failure set then $c$ has an odd-cardinality intersection with some set $T \in \text{TEST}$. If $i$ is the lowest-numbered edge in $T \cap c$, then $c$ consists of the edge $i$ together with a path $P$ between $x(i)$ and $y(i)$. Also, $P$ has no edges in common with $T \cap \{1, 2, \ldots, i\}$ and has an even number of edges in common with $T \cap \{i + 1, \ldots, m\}$. This characterization of the minimal failure sets suggests the following Monte-Carlo structure, in which the universe $S$ consists of all quadruples $(T, i, P, s)$ such that:

(i) the set of edges $T$ is an element of $\text{TEST}$ and edge $i$ is an element of $T$;
(ii) $P$ is a path between $x(i)$ and $y(i)$;
(iii) $P \cap T \cap \{1, 2, \ldots, i\}$ is empty and $P \cap T \cap \{i + 1, \ldots, m\}$ is of even cardinality;
(iv) all edges of the cycle $P \cup \{i\}$ are bad in state $s$.

The weight function $a$ is specified by the rule $a(T, i, P, s) = b(s)$. The target set $R$ is required to have the property that, for each failure state $s$, exactly one quadruple $(T, i, P, s)$ lies in $R$. Except for this requirement we leave open for the moment the question of specifying $R$.

The Monte-Carlo structure we have now specified is admittedly complicated, but, in return for its complication a natural approach to sampling from the universe $S$ suggests itself. Define $\text{WEIGHT}(P)$, the weight of path $P$, as the product of the failure probabilities of the edges in $P$. For each pair $(T, i)$ such that $T \in \text{TEST}$ and $i \in T$, let $\text{PATHS}(T, i)$ be the set of all paths $P$ between $x(i)$ and $y(i)$ such that $P \cap T \cap \{1, 2, \ldots, i\}$ is empty and $P \cap T \cap \{i + 1, \ldots, m\}$ is of even cardinality, and let $\text{PATHWEIGHT}(T, i)$ be the sum of the weights of the paths in $\text{PATHS}(T, i)$. Let

$$\text{TOTALWEIGHT} = \sum_{T \in \text{TEST}} p(i)\text{PATHWEIGHT}(T, i),$$
where \( p(i) \) denotes the failure probability of edge \( i \). Suppose that \( \text{PATH-WEIGHT}(T, i) \) were known for all \( T \) and \( i \) and that we had a method of sampling from \( \text{PATHS}(T, i) \) so that the probability of drawing any given path \( P \) is \( \text{WEIGHT}(P) / \text{PATHWEIGHT}(T, i) \). Then we could sample from the universe \( S \) as follows.

**Hypothetical sampling method.**

1. Draw a pair \((T, i)\) using a sampling method in which the probability of drawing any given pair \((T, i)\) is \( p(i) \text{PATH-WEIGHT}(T, i) / \text{TOTALWEIGHT} \);
2. Draw a path \( P \) from \( \text{PATHS}(T, i) \) using a sampling method in which the probability of drawing any given path \( P \) is \( \text{WEIGHT}(P) / \text{PATHWEIGHT}(T, i) \);
3. Choose a state \( s \) by the following randomized procedure: \( s(i) = \text{bad} \); if \( e \) lies in \( P \) then \( s(e) = \text{bad} \); else, \( s(e) = \text{bad} \) with probability \( p(e) \), and \( s(e) = \text{good} \) with probability \( 1 - p(e) \).

The result of the hypothetical sampling method is a quadruple \((T, i, P, s)\) that lies in the universe \( S \) associated with the current Monte-Carlo structure. It is easy to verify that each quadruple \((T, i, P, s)\) is drawn with the correct probability, \( a(T, i, P, s) / a(T) \).

The above sampling method is called hypothetical because efficient methods are unlikely to exist for computing \( \text{PATHWEIGHT}(T, i) \) or sampling from \( \text{PATHS}(T, i) \); indeed, these problems can be shown to be NP-hard. The difficulty of computing \( \text{PATHWEIGHT}(T, i) \) stems from the fact that we want to sum up the weights of all relevant paths, without including the weights of any walks that repeat vertices; no efficient method is known for capturing the desired paths while screening out the unwanted walks. To get around this difficulty we enlarge the universe associated with our Monte-Carlo structure in such a way that certain walks are included along with the paths that we are really interested in.

### 7. A Monte-Carlo Algorithm Based on Walks

In the following discussion a walk is viewed as a sequence of edges, and the length of a walk is the number of edges it contains. If walk \( A \) occurs as a consecutive subsequence of walk \( B \) then \( A \) is called a subwalk of \( B \). If \( A \) and \( B \) are walks and the final vertex of \( A \) is the initial vertex of \( B \), then \( AB \) denotes the walk obtained by concatenating \( A \) and \( B \) together. If \( W = e_1 e_2 \ldots e_t \), then \( \text{WEIGHT}(W) \), the weight of walk \( W \), is defined as \( \Pi_{j=1}^t p(e_j) \). Note that the edges \( e_j, j = 1, 2, \ldots, t \), are not necessarily distinct, and that, in computing the weight of walk \( W \), the failure probability of each edge is multiplied in as many times as the edge occurs in \( W \). In the case where \( W \) is a path, the
new definition of weight agrees with the earlier definition of the weight of a path.

Let $k$ be a positive integer that will be fixed throughout the following discussion. Walk $W$ is called a $k$-walk if the length of $W$ is less than or equal to $n' - 1$ and every subwalk of $W$ of length less than or equal to $k$ is a path. Thus, repetitions of a vertex may occur in $W$, but they must be separated by at least $k + 1$ edges. Call the $k$-walk $W (T, i)$-valid if the endpoints of $W$ are $x(i)$ and $y(i)$, and $x(i)$ and $y(i)$ do not occur as internal vertices in $W$, $W$ contains no edges from $T \cap \{1, 2, \ldots, i\}$, and $W$ contains an even number of occurrences of edges from $T \cap \{i + 1, \ldots, m\}$. In our enlarged Monte-Carlo structure the universe $S$ consists of all the quadruples $(T, i, P, s)$ included in the universe of the previous Monte-Carlo structure, together with all those triples $(T, i, W)$ such that $W$ is a $(T, i)$-valid $k$-walk but not a path. The weights of the triples and quadruples in the universe are defined as follows: $a(T, i, W) = p(i) \text{WEIGHT}(W)$, and $a(T, i, P, s) = b(s)$. The target set $R$ contains no triples and contains exactly one quadruple $(T, i, P, s)$ for each failure state $s$; hence, $a(R)$ is equal to the sum of the probabilities of all the failure states.

In order to obtain a Monte-Carlo algorithm from this new Monte-Carlo structure it is necessary to give methods for computing $a(S)$, the weight of the universe, and for sampling from $S$. We introduce several definitions for this purpose. Define $\text{WALKWEIGHT}(T, i, k)$ as the sum of the weights of all $(T: i)$-valid $k$-walks. Define

$$\text{TOTALWEIGHT}(k) = \sum_{i \in T} \sum_{T \in \text{TEST}} p(i) \text{WALKWEIGHT}(T, i, k).$$

It is easy to verify that $a(S)$, the total weight of the universe for our enlarged Monte-Carlo structure, is equal to $\text{TOTALWEIGHT}(k)$. Suppose that, for some fixed $k$, $\text{WALKWEIGHT}(T, i, k)$ were known for all pairs $(T, i)$ and that a sampling method were available that drew any given $(T, i)$-valid $k$-walk $W$ with probability $\text{WEIGHT}(W)/\text{WALKWEIGHT}(T, i, k)$. Then we could sample from the universe $S$ as follows:

**Sampling method.** (i) Choose the pair $(T, i)$ by a sampling method that selects each pair $(T, i)$ with probability $p(i)\text{WALKWEIGHT}(T, i, k)/\text{TOTALWEIGHT}(k)$;

(ii) Choose a $(T, i)$-valid $k$-walk $W$ by a sampling method that selects each $(T, i)$-valid $k$-walk $W$ with probability $\text{WEIGHT}(W)/\text{WALKWEIGHT}(T, i, k)$;

(iii) if $W$ is not a path then the selected element of the universe $S$ is the triple $(T, i, W)$;

(iv) if $W$ is a path then select state $s$ by the following randomized process: if $e \in W \cup \{i\}$ then $s(e) = \text{bad}$
the selected element of the universe is the quadruple \((T, i, W, s)\).

It is easy to verify that the given sampling method selects elements of the universe \(S\) with probabilities proportional to their weights. To show that the method is computationally feasible we give algorithms based on dynamic programming to compute \(\text{WALKWEIGHT}(T, i, k)\) and to perform the sampling required in step (ii). Let \(H(T, i)\) (hereafter abbreviated to \(H\)) be the graph \((V', E')\), where \(V'\) is the vertex set of \(G'\) and \(E' = E - (T \cap \{1, 2, \ldots, i\})\). Then the set of \((T, i)\)-valid \(k\)-walks is precisely the set of \(k\)-walks from \(x(i)\) to \(y(i)\) in \(H\) with an even number of occurrences of edges from \(\{i + 1, \ldots, m\}\) in which neither \(x(i)\) nor \(y(i)\) occurs as an internal vertex.

Let \(\lambda\) denote the empty path. Let \(\text{TAILS}(T, i, k)\) denote the set of all paths \(P\) in \(H\) such that

(i) \(x(i)\) either does not occur in \(P\) or occurs only as the initial vertex of \(P\);
(ii) \(y(i)\) either does not occur in \(P\) or occurs only as the final vertex of \(P\); and
(iii) either \(P\) is of length \(k\) or \(P\) starts at \(x(i)\) and is of length less than \(k\).

If \(y\) is any walk in \(H\) let \(\text{TAIL}(y)\) be the longest suffix of \(y\) that lies in \(\text{TAILS}(T, i, k)\). For any walk \(y\) let \(\text{len}(y)\) be the length of \(y\) and let \(\text{par}(y)\) be the residue modulo 2 of the number of occurrences in \(y\) of edges from \(\{i + 1, \ldots, m\}\). For any triple \((A, r, d)\) such that \(A\) is an element of \(\text{TAILS}(T, i, k)\), \(r\) is an integer between 0 and \(n' - 1\) and \(d\) is 0 or 1, let \(F(A, r, d)\) be the sum of the weights of all \(k\)-walks \(y\) in \(H\) which start at vertex \(x(i)\) such that \(\text{TAIL}(y) = A\), \(\text{par}(y) = d\) and \(\text{len}(y) = r\). Then \(\text{WALKWEIGHT}(T, i, k) = \sum F(A, r, 0)\), where the summation extends over all paths \(A\) in \(\text{TAILS}(T, i, k)\) that terminate at \(y(i)\), and over all \(r\) between 0 and \(n' - 1\). We give a recursive formula in the spirit of dynamic programming for the tabulation of the function \(F(A, r, d)\). For this purpose we require two more definitions.

Let \(\oplus\) denote addition modulo 2. For any path \(A\) in \(\text{TAILS}(T, i, k)\) define \(\text{pred}(A)\) as the set of pairs \((B, e)\) such that \(B \in \text{TAILS}(T, i, k)\), \(e\) is an edge of \(H\), \(Be\) is a walk, \(\text{TAIL}(Be) = A\), and \(\text{len}(A) = \max(k, 1 + \text{len}(B))\). Then

\[
F(A, 0, d) = 1 \quad \text{if} \ A = r \ \text{and} \ d = 0
\]
\[
= 0 \quad \text{otherwise}
\]

and, for \(r > 0\).

\[
F(A, r, d) = \sum p(e)F(B, r - 1, d + \text{par}(e)),
\]
where the summation extends over all pairs \((B, e)\) in \(\text{pred}(A)\). The number of computation steps required to tabulate the function \(F(A, r, d)\) and then compute \(\text{WALKWEIGHT}(T, i, k)\) is \(O(|\text{TAILS}(T, i, k)|m n')\). Once the function \(F(A, r, d)\) has been tabulated and \(\text{WALKWEIGHT}(T, i, k)\) has been computed it is an easy matter to draw a walk from \(\text{WALKS}(T, i, k)\) in such a way that each walk \(W\) is drawn with probability \(\text{WEIGHT}(W)/\text{WALKWEIGHT}(T, i, k)\). This is done with the help of a randomized procedure \(\text{SAMPLE}(A, r, d)\) which takes as input a triple \((A, r, d)\) in the domain of the function \(F(A, r, d)\) and selects a \(k\)-walk \(W\) of length \(r\) such that \(\text{TAIL}(W) = A\) and \(\text{par}(W) = d\), such that the probability of choosing any such \(k\)-walk \(W\) is \(\text{WEIGHT}(W)/F(A, r, d)\). This procedure executes recursively as follows.

\textbf{Procedure SAMPLE} \((A, r, d)\)

\begin{enumerate}
  \item if \(r = 0\) then if \(A = \lambda\) and \(d = 0\) then output \(\lambda\)
  \item else output "error"
  \item else choose a pair \((B, e)\) in \(\text{pred}(A)\) by a sampling method that selects any such pair \((B, e)\) with probability \(p(e)F(B, r - 1, d + \text{par}(e))/F(A, r, d)\);
  \item \(z \gets\) the output of \(\text{SAMPLE}(B, r - 1, d + \text{par}(e))\);
  \item output \(\leftarrow ze\).
\end{enumerate}

The procedure for selecting a walk from \(\text{WALKWEIGHT}(T, i, k)\) using Procedure \(\text{SAMPLE}(A, r, d)\) is as follows.

\textbf{Procedure SELECTWALK}

Choose a pair \((A, r)\) such that \(A \in \text{TAILS}(T, i, k)\), \(A\) terminates at \(y(i)\), and \(0 \leq r \leq n' - 1\) by a sampling method that selects any such pair with probability \(F(A, r, 0)/\text{WALKWEIGHT}(T, i, k)\);
\begin{enumerate}
  \item call Procedure \(\text{SAMPLE}(A, r, 0)\)
\end{enumerate}

The efficient implementation of procedures \(\text{SAMPLE}(A, r, d)\) and \(\text{SELECTWALK}\) depends on a standard trick for sampling from a discrete probability distribution. In order to draw a random variable \(X\) with a given probability distribution over the set \(\{1, 2, \ldots, N\}\) one constructs an \(N\)-element array \(A\) whose \(j\)th element gives the probability that \(X \leq j\). In order to sample one draws a random number \(x\) from the uniform distribution over \([0, 1]\) and then sets \(X\) equal to the least \(j\) such that \(A[j] \geq x\); binary search can be used to find \(j\) using \(O(\log N)\) comparisons between the random number \(x\) and elements of the array. If the distribution is defined over some arbitrary \(N\)-element set the trick can still be applied by introducing a bijection between this set and the set \(\{1, 2, \ldots, N\}\). This trick is used within Procedure \(\text{SAMPLE}(A, r, d)\) to select a pair \((B, e)\) in \(\text{pred}(A)\) and within Procedure \(\text{SELECTWALK}\) to select \(A\) and \(r\).

The time required to execute Procedure \(\text{SELECTWALK}\) is dominated by the time required for the recursive calls on procedure \(\text{SAMPLE}(A, r, d)\). The number of such calls is at most \(n'\) and, with the aid of the sampling trick
mentioned in the last paragraph, the execution time of each call is \( O(\log m) \). Hence the execution time of Procedure SELECTWALK is \( O(n' \log m) \). In the Monte-Carlo structure we are considering it is required that the target set \( R \) contain exactly one quadruple \((T, i, P, s)\) for each failure state \( s \). Although we have been working primarily with paths and cycles in the dual graph \( G' \), it is convenient to specify \( R \) in terms of the planar graph \( G = (V, E) \) and terminal set \( K \) in terms of which the multiterminal planar reliability problem was originally specified. Let \( s \) be a given failure state. Let \( g(s) \) be the canonical cut in \( G \) that fails in state \( s \); \( g(s) \) is also a cycle in \( G' \). A computationally efficient method of selecting \( g(s) \) is given in Section 4. Let the elements of the test set \( \text{TEST} \) be linearly ordered in a fixed but arbitrary way, and let \( T \) be the first element of \( \text{TEST} \) that has an odd number of edges in common with \( g(s) \). Let \( i = \min\{j | j \in T \cap g(s)\} \). Let \( P \) be the path in \( G' \) obtained by deleting edge \( i \) from \( g(s) \). Then the quadruple \((T, i, P, s)\) is the unique element of \( R \) corresponding to the failure state \( s \).

8. THE CHOICE OF THE PARAMETER \( k \)

Finally, it is necessary to select the integer parameter \( k \), which is of central importance in the specification of our Monte-Carlo structure. Recall that the universe \( S \) consists of all quadruples \((T, i, P, s)\) such that \( P \) is a \((T, i)\)-valid path and \( s \) is a state in which all edges of \( P \cup \{i\} \) are bad, together with all triples \((T, i, W)\) such that \( W \) is a \((T, i)\)-valid \( k \)-walk which is not a path. The triples \((T, i, W)\) represent "padding" that is added to the universe \( S \) in order to facilitate sampling from \( S \) and computing \( a(S) \), the weight of \( S \). The overall Monte-Carlo algorithm consists of a preprocessing phase and a sampling phase. In the preprocessing phase dynamic programming calculations are performed to determine the value of \( a(S) \) and to construct the tables used later to draw samples from the universe \( S \). In the sampling phase elements are drawn from \( S \) and a tally is kept of how many of these elements lie in the target set \( R \). As \( k \) increases the definition of a \( k \)-walk becomes stricter, the amount of padding decreases, the weight of the universe \( S \) decreases, and accordingly, the ratio \( a(S)/a(R) \) that determines the sample size required in the Monte-Carlo algorithm also decreases. Thus an increase in \( k \) reduces the computation time required for the sampling phase of the Monte-Carlo algorithm. On the other hand, as \( k \) increases, the cardinality of each set \( \text{Tails}(T, i, k) \) increases, leading to an increase in the execution time and memory requirements for the tabulation of the function \( F(A, r, d) \). Thus, increasing \( k \) leads to an increase in the time and storage requirements of the preprocessing phase.

The optimal choice of \( k \) must represent a trade-off between the complexity of the preprocessing phase and the complexity of the sampling phase. In
exploring this trade-off we consider two parameters associated with the graph \( G' \): \( D \), the maximum degree, and \( C \), the maximum, over all vertices \( v \) of \( G' \), of the sum of the failure probabilities of the edges of \( G' \) incident with \( v \). In terms of the planar graph \( G \) occurring in the original statement of the multi-terminal planar network reliability problem, \( D \) represents the maximum number of edges bounding a face and \( C \) represents the maximum, over all faces, of the sum of the failure probabilities of the edges bounding the face.

We shall say that our Monte-Carlo structure satisfies the Padding Condition if \( \sum a(T, i, W) \leq \sum a(T, i, P, s) \), where the first summation is over all the triples in the universe \( S \), and the second summation is over all quadruples \((T, i, P, s)\) in the universe \( S \). This condition means that the "padding" introduced by considering \( k \)-walks as well as paths at most doubles the weight of the universe \( S \). The following theorem guarantees that a certain choice of \( k \) will cause the Padding Condition to be satisfied.

**Theorem 3.** If \( C < \frac{1}{2} \) and the parameter \( k \) is chosen so that \((1/C)^k \geq 4n'\) then the Padding Condition will be satisfied.

**Proof.** We assemble some facts required for the proof.

**Fact 1.** Let \( T \) be a set of edges and let \( M \) be a cyclic walk having an odd number of occurrences of edges from \( T \). Then \( M \) has a subwalk which is a cycle having an odd number of edges from \( T \).

**Proof.** If \( M \) is not a cycle then it can be split into two cyclic subwalks, \( M' \) and \( M'' \), such that the number of occurrences of any edge in \( M' \) and \( M'' \) together is equal to the number of occurrences of that edge in \( M \). Since \( M \) has an odd number of occurrences of elements of \( T \), either \( M' \) or \( M'' \) must have an odd number of occurrences of elements of \( T \). The result now follows by induction.

**Fact 2.** The sum of the weights of all walks of length \( t \) beginning at a given vertex \( v \) is at most \( C^t \). Hence the sum of the weights of all walks of at least \( k \) beginning at \( v \) is at most \( C^k/(1 - C) \).

**Proof.** Induction on \( t \).

Given two cyclic walks \( M' \) and \( M'' \) with a common vertex \( v \), one can form a cyclic walk which starts at \( v \), returns to \( v \) by following all the occurrences of edges in \( M' \), and then returns to \( v \) again by following all the occurrences of edges in \( M'' \). Such a cyclic walk is said to result from joining \( M' \) and \( M'' \) at \( v \). Let \( CY \) be a simple cycle of \( G' \) and let \( WA \) be a cyclic walk in \( G' \). Then \( CY \) is called a \( k \)-seed for \( WA \) if there is a (possibly empty) set of cyclic walks, each of length at least \( k \) and having a vertex in common with \( CY \), such that \( WA \) is created by successively joining these walks with \( CY \).
Fact 3. If $C$ and $k$ are chosen as in the statement of the theorem then, for any simple cycle $CY$, the sum of the weights of the cyclic walks having $CY$ as a $k$-seed is at most twice the weight of $CY$.

Proof. Let $\text{SUM}(CY, v)$ denote the sum of the weights of the cyclic walks of length at least $k$ having vertex $v$ in common with $CY$. Then the sum of the weights of the cyclic walks having $CY$ as a $k$-seed is bounded above by $\Pi_v(1 + \text{SUM}(CY, v))$. But $\text{SUM}(CY, v) \leq C^k/(1 - C) \leq 2C^k$, so

$$\prod_v (1 + \text{SUM}(CY, v)) \leq \prod_v (1 + 2C^k) \leq \exp(2n'C^k) \leq \exp(1/2) < 2.$$ 

Now we are ready to prove the theorem. The universe $S$ consists of triples $(T, i, W)$ and quadruples $(T, i, P, s)$. For each $T \in \text{TEST}$, let $\text{WA}(T)$ denote the set of cyclic $k$-walks containing an odd number of occurrences of edges from $T$, let $\text{CY}(T)$ denote the set of cycles containing an odd number of edges from $T$, let $\text{WAWEIGHT}(T)$ denote the sum of the weights of the walks in $\text{WA}(T)$ and let $\text{CYWEIGHT}(T)$ denote the sum of the weights of the cycles in $\text{CY}(T)$. Then $a(S)$, the weight of the universe $S$, is equal to $\sum_{T \in \text{TEST}} \text{WAWEIGHT}(T)$ and the sum of the weights of the quadruples $(T, i, P, s)$ in the universe $S$ is equal to $\sum_{T \in \text{TEST}} \text{CYWEIGHT}(T)$. For each fixed $T$, $\text{WAWEIGHT}(T) \leq 2\text{CYWEIGHT}(T)$. This follows from Fact 3 by noting that every walk in $\text{WA}(T)$ has as a $k$-seed a cycle in $\text{CY}(T)$. It follows that the weight of the universe $S$ is at most twice the sum of the weights of the quadruples in $S$. This completes the proof of the theorem.

Assuming that the parameter $k$ is chosen so as to guarantee the Padding Condition, we analyze the time required for the dynamic programming calculations used in the preprocessing phase of the algorithm. For each $T \in \text{TEST}$ and each $i \in T$, the time required to tabulate the function $F(A, r, d)$ and to compute $\text{WALKWEIGHT}(T, i, k)$ is $O(m n' |\text{TAILS}(T, i, k)|)$. Noting that $|\text{TAILS}(T, i, k)| < (n' + 1)D^k$ we find that the time required for the preprocessing associated with the pair $(T, i)$ is $O(m(n')^2 D^k)$. If the set of terminals in graph $G$ is $K$ then the test set $\text{TEST}$ can be chosen so that it has at most $|K| - 1$ elements, each of which is a path in $G$, and hence is of cardinality at most $n - 1$. Thus the overall execution of the preprocessing phase of the Monte-Carlo algorithm is $O(m(n')^2 nK D^k)$. In order to satisfy the condition of Theorem 3 we require that $(1/C)^k \geq 4n'$, which is equivalent to $D^k \geq (4n')^{k \log_2 C/D}$. The results of this analysis can be summarized in the following theorem.

Theorem 4. If $C < \frac{1}{4}$ then the parameter $k$ can be chosen so that the Padding Condition is satisfied and the execution time of the preprocessing phase of the Monte-Carlo algorithm is $O(m(n')^2 nK D (4n')^{k \log_2 C/D})$.

Finally, we investigate the number of trials required by the Monte-Carlo algorithm, assuming that the Padding Condition holds. It suffices for the
number of trials to be at least \( a(S)/a(R) \ln(2/\delta)4.5/e^2 \). Because of the Padding Condition \( a(S) \leq 2 \sum a(T, i, P, s) \). Moreover, \( a(R) = \sum b(s) \) and \( \sum a(T, i, P, s) = \sum_{(c,s)} b(s) \), where \( s \) ranges over all failure states and \( (c,s) \) ranges over all pairs such that \( s \) is a failure state and \( c \) is a \( K \)-separating cut whose edges are all bad in state \( s \). By Theorem 2, \( \sum_{(c,s)} b(s) \leq (K - 1)\Pi(1 + p(e))\Sigma b(s) \). Hence it follows that the number of trials required in order for the Monte-Carlo algorithm to guarantee an \( \epsilon - \delta \) approximation is at most \((K - 1)\Pi(1 + p(e)) \ln(2/\delta)4.5/e^2 \).

The following statement summarizes our claims about the efficiency of our Monte-Carlo algorithm for the planar multiterminal network reliability problem.

**Theorem 5.** Let an instance of the planar multiterminal reliability problem be specified by a planar graph \( G \), a set of terminals \( K \), and an assignment of failure probabilities \( p \). Let \( G \) have \( m \) edges, \( n \) vertices, and \( n' \) faces. Let \( D \) denote the maximum number of edges on any face of \( G \), and let \( C \) denote the maximum sum of failure probabilities around any face of \( G \). Our Monte-Carlo algorithm yields an \( \epsilon - \delta \) approximation to the failure probability of \( G \). If \( C < \frac{1}{3} \) then the execution time of the algorithm is

\[
O\left( m(n')^2 n K D (4n')^{\log_2/C} D + K \prod_{e}(1 + p(e)) n' \ln m \ln \left( \frac{2}{\delta} \right) (4.5/e^2) \right).
\]

**Corollary 1.** For any class of instances of the multiterminal planar network reliability problem in which \( C < \frac{1}{3} \), \( \Pi (1 + p(e)) \) is bounded above by a polynomial in \( n \) and \( \log_2/C D \) is bounded above by a constant, and for every \( \epsilon \) and \( \delta \), the Monte-Carlo method of the present paper provides an \( \epsilon, \delta \) approximation algorithm that runs in polynomial time.

Luby (1983) gives a PASCAL program based on the Monte-Carlo algorithm given in the present paper and and gives the results of running the algorithm on six examples. This limited computational experience suggests that the performance of the algorithm in practice is considerably better than the worst-case guarantee of Theorem 5 would indicate.

**References**


PROVAN, J. S., AND BALL, M. O. (1981), The complexity of counting cuts and of computing the probability that a graph is connected, working paper.


